Heuristic Approaches to Minimize the Number of Rehandlings for Export Containers

FATMA VİRDİL

KOC UNIVERSITY
AUGUST 2012

Heuristic Approaches to Minimize the Number of Rehandlings for Export Containers

by

Fatma Virdil

A Thesis Submitted to the

Graduate School of Engineering

in Partial Fulfillment of the Requirements for

the Degree of

Master of Science

in

Industrial Engineering

Koc University

August 2012

Koc University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Fatma Virdil

and have found that it is complete and satisfactory in all respects,

and that any and all revisions required by the final

examining committee have been made.

Committee Members:

____

Prof. Dr. Ceyda Oguz, (Advisor)

____

Asst. Prof. Dr. Sibel Salman

____

Assoc. Prof. Dr. Tonguc Unluyurt

Date: ______________________

ABSTRACT

In the growing international trade, the number of import, export and transit containers is ever increasing. Therefore, the importance of the container terminals and their efficient managements are highlighted. In this thesis, we study stacking policies in a container terminal for export containers, due to their characteristics. Different than the literature, we considered both the space allocations for containers arrived recently and the pick up operations before the departure of the containers. We assume that an initial configuration exists in the storage yards for already stored containers. The containers arrived recently are allocated to the blocks in the storage yard based on this initial configuration by taking the departure time of the containers into account, which are known beforehand since they are export containers. Hence the retrieval sequences of the containers within blocks are known. Another important aspect of our study is the inclusion of the remarshaling operation, which is used to speed up the retrieval of the export containers. Throughout these stacking operations (allocation, remarshaling, and pick up), some containers already stored in the blocks might be moved into other positions and these moves are known as relocations. These relocations cause major time and cost expenses in the container terminals since any relocation may result in several rehandling operations. Hence the main focus of this thesis is to deal with the rehandlings. We propose several heuristic approaches to estimate the locations for the containers arrived recently and maybe relocated export containers in order to minimize the number of rehandlings. We then analyze the performance of these proposed heuristic algorithms with a set of randomly generated problem instances considering different initial configurations under different container terminal scenarios.

Key words: container terminals, export containers, remarshaling, heuristic algorithms

ÖZET

Büyüyen uluslararası ticarette, artan ithalat, ihracat ve transit konteynerlerin sayısı konteyner terminallerin verimli yönetilmesinin önemini her geçen gün arttırmaktadır. Bu çalışmada, ihracat konteynerlerinin konteyner terminallerindeki istifleme yöntemlerini incelemekteyiz. Literatürden farklı olarak, hem yeni gelen konteynerlerin yerleştirilmelerindeki hem de ayrılmalarındaki yerleştirme haraketleri ele alınmaktadır. Konteynerlerin depolama alanına atanmaları sonucu bloklar oluşmakta ve bu nedenle bilinen bir başlangıç konfigürasyonun olduğu varsayılmaktadır. Yeni gelen konteynerler depolama alanındaki bloklara bu varsayılan konfigürasyona ve terminalden ayrılma zamanlarına göre atanmaktadır. Bu ayrılma zamanları, ihracat konteynerleri ele alındığı için bilinmektedir. Buna bağlı olarak herhangi bir bloktaki konteynerlerin terminalden ayrılma sıraları bilinmektedir. Çalışmanın bir diğer özelliği de yeniden düzenleme (remarshaling) operasyonunun ihracat konteynerlerinin terminalden ayrılmalarını hızlandırmak amacıyla kullanılmasıdır. İstifleme operasyonları (atama, yeniden düzenleme ve toplama) esnasında, blokta bulunan bazı konteynerlerin yeniden yerleştirilmeleri (relocation), konteynerlerin yeniden elleçlenmesine (rehandling) yol açtığı için, büyük ölçüde zaman ve para kaybına neden olmaktadır. Dolayısıyla, bu çalışmanın amacı elleçleme sayılarının enazlanmasıdır. Bu amaca ulaşmak için tez çalışmasında yeni gelen veya yeri değişen ihracat konteynerlerinin yerlerini hesaplamak için değişik sezgisel yöntemler önerilmektedir. Daha sonra, geliştirilen yöntemlerin performansları, farklı başlangıç konfigürasyonlarına sahip rastgele oluşturulmuş örnekler kullanılarak farklı konteyner terminal senaryoları üzerinden karşılaştırılmıştır.

Anahtar kelimeler: konteyner terminalleri, ihracat konteynerleri, yeniden düzenleme, sezgisel yöntemler

ACKNOWLEDGEMENTS

I would like to thank all the faculty members of the Industrial Engineering Department at Koc University for their support, encouragement and excellent teaching during both my undergraduate and graduate studies. I thank my advisor Prof. Dr. Ceyda Oğuz for her guidance during both my senior project in undergraduate study and my research in graduate study.

I gratefully acknowledge TUBITAK Graduate Scholarship Programme (Code: 2210) for their financial support.

I appreciate the friendship and continuous support of Hacer Ayaz, İdil Su Göktan, Utku Boz and Engin Çukuroğlu during whole my university and private life.

Most special thanks go to Mehmet Ayyıldız for his outstanding company, friendship, support and patience.

Finally, I would like to thank my parents, Ahmet and Seyyare Virdil for being there whenever I need them throughout my life. None of this would have been possible without their love, patience and moral support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Motivation

In recent years, the flow of cargo has increased steadily due to the growth of international trade. Most of such international cargo use containers as the medium of transportation. Since containers are solid structures with standard dimensions, they are easy to carry and less prone to damages. In today's container transportation generally 20, 40 and 45 feet sized containers are used. TEU (twenty-foot equivalent unit) is considered as a comparison unit in container transportation. Containers of 20 feet are known as 1 TEU, 40 feet containers are known as 2 TEU and 45 feet containers are also considered as 2 TEU, rather than 2.25 TEU. Hence, containers are accepted as standard unit loads for international cargo globally.

Modern container shipping has started in 1956 and the usage of containers has increased rapidly during years. The growth in world-wide container traffic is nearly 140% between 1990 and 2000, and 433% between 1990 and 2010 [1]. As the international trade increases, the importance of container transshipment also increases, which results in building larger vessels and larger fleet sizes to accommodate the increasing volume of containers to be transshipped. Fleet size growth is parallel with the growth of container traffic, which has increased nearly 133% between 1990 and 2000, and 333% between 1990 and 2010 [2]. Table 1 gives the growth in container traffic and fleet size in detail.

Table 1: Growth of container traffic and fleet size between 1990 and 2010.

	Year									
	1991	**1992**	**1993**	**1994**	**1995**	**1996**	**1997**	**1998**	**1999**	**2000**
World Container Traffic (Million TEU)	31.3	34.1	37.1	41.9	46	49.1	54	56.3	61.6	68.3
End-Year Fleet Size (Million)	6.9	7.6	8.1	8.8	9.73	10.6	11.5	12.4	13.5	14.9
	2001	**2002**	**2003**	**2004**	**2005**	**2006**	**2007**	**2008**	**2009**	**2010**
World Container Traffic (Million TEU)	70.7	78.9	91.9	105.3	115.5	127	142.4	149	134.6	153
End-Year Fleet Size (Million)	15.5	16.6	18.1	20	21.4	23.3	26.2	28.1	27.1	27.6

It is apparent that the expanding number of containers and vessels necessitates an efficient management of containers at container terminals for higher service levels.

1.2 Container Terminals and Operations

Container terminals include facilities at the sea-side (such as berths) and on the land (such as storage yard area). Figure 1 gives a general top view of a container terminal and flow of transportation [3]. Containers are transported into these terminals by vessels, trucks or trains. Detailed descriptions and classifications of the main processes and operations in container terminals are given in [3] and [4] and displayed in Figure 2.

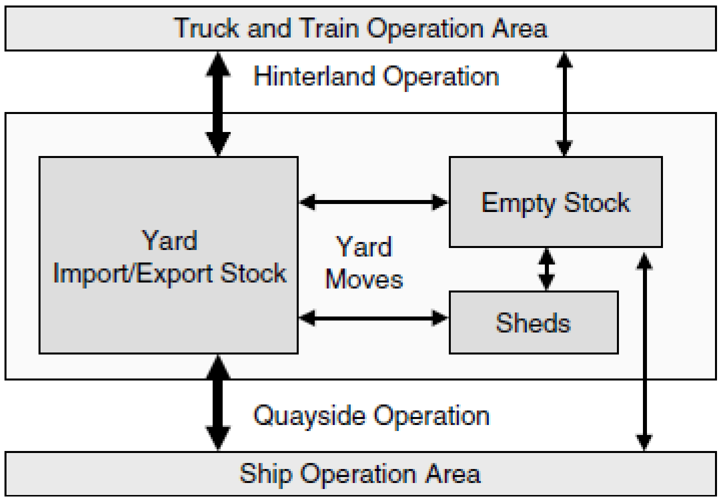


Figure 1: Operation areas and flow of transports in a container terminal [3].

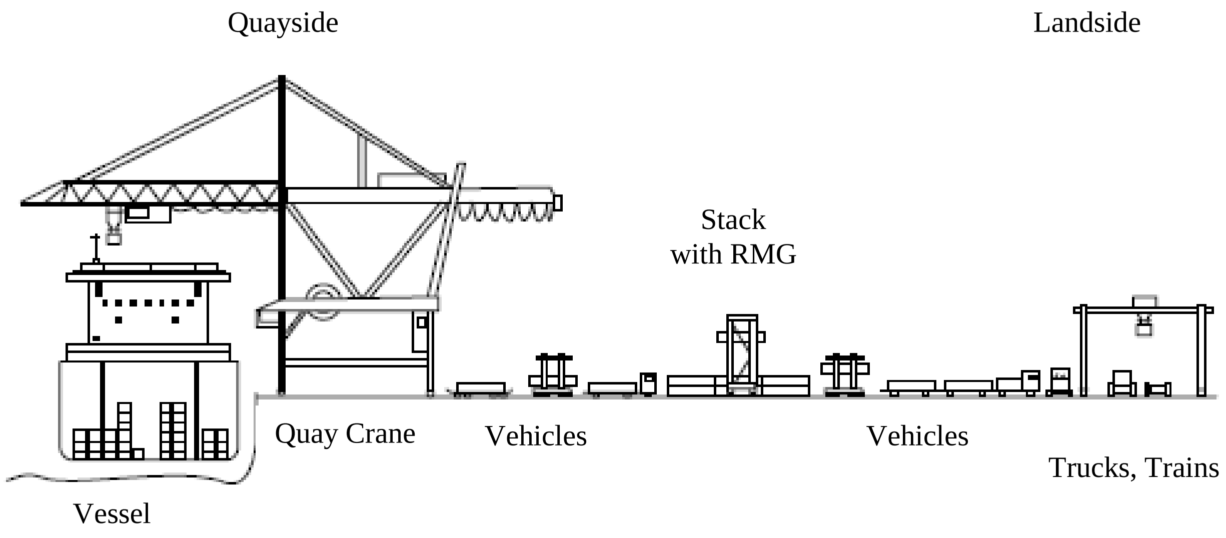


Figure 2: A schematic side view of a container terminal system (not in scale) [3] and [4].

The flow of containers and their related operations in a container terminal depend on the type of the containers. There are three types of containers stored in the terminals: import, export and transit. Import containers arrive by vessels to the terminals and are stored in terminals until they are claimed by a truck or train for land transportation. On the other hand, export containers follow the opposite route; they arrive by land transportation and are stored in the terminal until they are loaded into vessels. Transit containers use only sea transportation; they arrive at terminals by a vessel, and are stored in the terminal until they are loaded onto another vessel.

Container terminals include several operations such as loading, unloading, storage and handling. Operations for import and export containers occur both in quayside and hinterland. On the other hand, operations for transit containers only include quayside operations. Each container category type follows a similar operation route. This route starts with the arrival of the container into container terminal followed by unloading operation. Inter-transportation handling of containers into storage yard and stacking containers into blocks is the following step. Any stored container is picked up from the blocks and transported with inter-transportation handling and loaded on vehicles that claimed it.

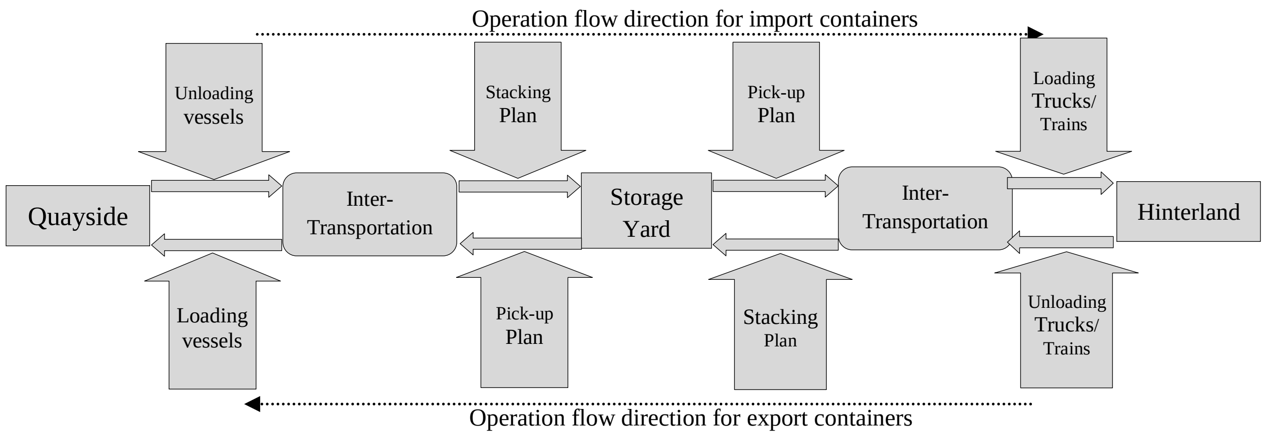


Figure 3: Diagram of operations in a container terminal.

Figure 3 shows a diagram of the main operations occurred in a container terminal. Each operation in the container terminals is highly dependent on each other.

Figure 4 displays vessels assigned to berths for unloading import or transit containers and loading export or transit containers. Quay cranes are assigned to vessels for these loading and unloading operations occurred after they are positioned at a berth (Figure 5).

Figure 4: A berthed vessel.

Figure 5: A berthed vessel and a quay crane assigned to that vessel.

Transportation between quayside and storage yard or hinterland and storage yard is handled with vehicles (Figure 6). Containers which have arrived into the storage yard area are stacked in the blocks by using yard cranes or straddle carriers (Figure 7 and 8). Containers are stored in the blocks until they are claimed. If an export or transit container is claimed by a vessel, it is picked up by yard cranes or straddle carriers, loaded on the trucks, transported to the quayside and loaded on the vessels by quay cranes. On the other hand, depending on the container terminal policy a claimed import container is either directly loaded on the truck that has claimed the container or loaded on a transportation vehicle, transported to the yard area and then loaded on the truck that claims it by transportation trucks directly (Figure 9) or by top-lift handling equipments (Figure 10).

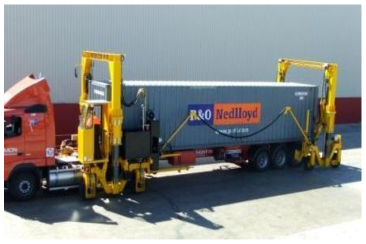


Figure 6: Storage yard area transportation.

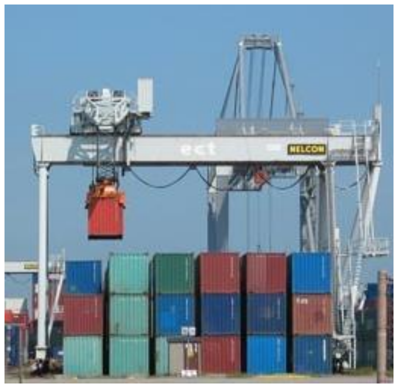


Figure 7: Yard crane (rail tired gantry crane).

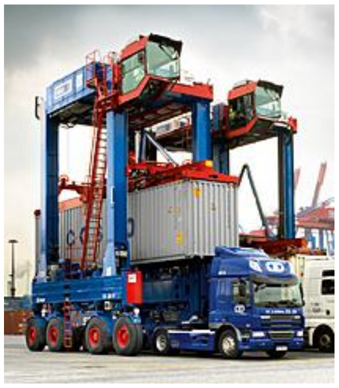


Figure 8: Straddle crane.

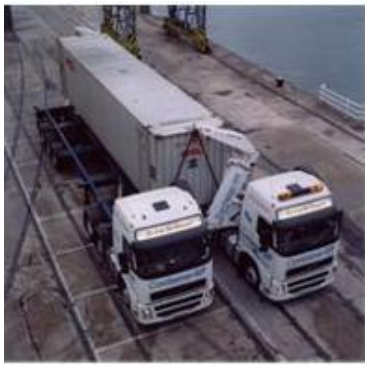


Figure 9: Loading of a container from truck to truck.

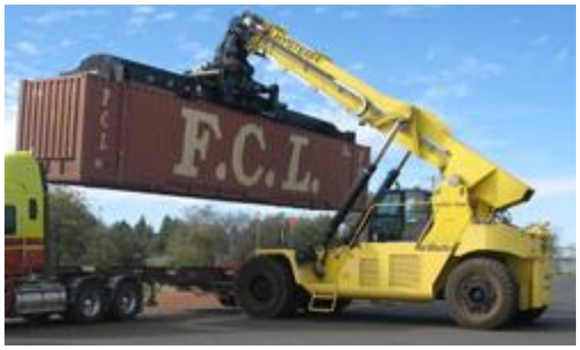


Figure 10: Truck loading by a top-lift type material handling equipment.

1.2.1 International Container Terminals

Vessels do not always travel on the same route during their transportations. Hence, both the countries and the container terminals the vessels docked show diversity depending on the type of container load they transport. Each container terminal in a country is localized in a position based on geographic characteristics of the region. These characteristics affect both the quayside and hinterland capacity of the container terminals. Moreover, storage yards, material handling equipment and personnel depend on the capacity of those container terminals.

Container terminals' importance is increasing parallel with the growth of worldwide container traffic. Major container terminals, which have higher transportation traffic, are located on different continents. Table 2 shows some of the major container terminals located in Asia or in Europe and their worldwide rankings between 2005 and 2009 [5].

Table 2: Rankings and container traffics of some major container terminals.

	Year									
	2005		**2006**		**2007**		**2008**		**2009**	
Terminal	**Ranking**	**Container Traffic (TEU)**	**Ranking**	**Container Traffic (TEU)**	**Ranking**	**Container Traffic (TEU)**	**Ranking**	**Container Traffic (TEU)**	**Ranking**	**Container Traffic (TEU)**
Singapore	1	23,192,000	1	24,792,400	1	27,932,000	1	29,918,200	1	25,866,400
Hong Kong	2	22,602,000	2	23,538,580	3	23,881,000	3	24,248,000	3	20,983,000
Rotterdam	7	9,286,757	7	9,654,508	6	10,790,604	9	10,783,825	10	9,743,290
Hamburg	8	8,087,545	9	8,861,804	9	9,889,792	11	9,737,110	15	7,007,704

Singapore and Hong Kong, each has a major container terminal in Asia continent since the container traffic is significantly higher in these cities. On the other hand, Rotterdam and Hamburg have the major container terminals, which are located in the Western Europe.

These container terminals can be either managed by terminal management itself, in which case are accounted as single terminals like in Hong Kong and in Rotterdam, or managed by commercial terminal operators such as in Singapore and in Hamburg, in which case are categorized based on their managements.

Singapore container terminals are managed by two commercial terminal operators. The first operator, PSA Singapore, manages 44 berths with a quay length of 12,800 meters and 143 quay cranes are used in between. Additionally, the terminal area is 436 hectares, which is designed to have a capacity of 24,700 kTEU. On the other hand, second operator, Jurong terminal, manages 30 berths, where the total berth lengths are 5,629 meters. The terminal areas, which are managed by Jurong terminal, are divided into two zones: free trade zone, which is 124 hectares and non-free trade zone that is 28 hectares. In these zones in total 28 hectares area is designated as warehouse facilities, and number of the warehouses is 25. Hong Kong container terminal is smaller than Singapore container terminals both in the number of terminals, berths and terminal area. Hong Kong container terminal manages nine terminals, which includes 24 berths in total and holds 279 hectares of land area in total.

Hamburg container terminal is another terminal that contains several container terminals. The first container terminal is Eurogate, which includes six large-ship berths and 21 container cranes. HHLA Tollerort is another container terminal, which includes a container rail station with a length of 720 meters of track, different than the others. This container terminal manages four berths and uses eight container gantries as handling equipment. HHLA Burchardkai container terminal has ten berths, which is higher in the number than the other Hamburg container terminals. Hence the number of container gantries used as material handling equipment increases and 27 container gantries are used in this container terminal. Finally, HHLA Altenwerder container terminal includes four berths for large container ships and uses fifteen container gantry cranes. The major importance of this terminal is to use automated-driverless vehicles. On the other hand, Rotterdam container terminal has a terminal area of 10,500 hectares in total. 5,000 hectares is used as commercial site, 3,500 hectares is used in water and 2,000 hectares is used for road and railways. Having a capacity like this, Rotterdam terminal has over 400 million tons of goods per annum as goods throughput.

1.2.2 National Container Terminals

Turkey is a peninsula which is surrounded by four different seas: Blacksea, Marmara, Aegean and Mediterranean. Therefore, there are several container terminals on its coats. These terminals play an important role in the container traffic between parts of Europe and Asia. Moreover the Bosporus connects countries which have terminals at the Blacksea to the Mediterranean container traffic so each port of Turkey takes part in the global container traffic.

Although Turkey has a great advantage of its geographical location and container terminals, there is a decrease in the number of rehandled containers in years. Due to the decrease in the number of containers rehandled in the ports, some of these terminals have been privatized. This action is performed in order to increase the involvement of Turkey in the container traffic sector. For example, Mersin terminal has been private after May, 2007 and named as MIP, while Samsun terminal is managed by private sector under the name of Samsun terminal after April, 2010 and Bandırma terminal, which is now named as Çelebi, is handled by a private terminal management company after mid-May, 2011. Table 3 provides information on the number of rehandled containers in the container terminals of Turkey.

Table 3: Number of rehandled containers in Containers terminals in Turkey (TEU).

	Year							
Terminal	**2004**	**2005**	**2006**	**2007**	**2008**	**2009**	**2010**	**2011**
Haydarpaşa	316,982	340,629	400,067	396,637	356,272	187,365	176,468	63,752
Mersin	532,999	596,289	643,749	232,181	-	-	-	-
İskenderun	607	0	52	603	0	0	115	11
Samsun	0	0	0	2	0	254	122	-
Derince	1,509	550	609	488	402	251	800	747
Bandırma	36	0	0	9	69	34	0	-
İzmir	804,563	784,377	847,926	898,217	884,906	826,645	727,443	219,356

Table 4: Capacity of container terminals in Turkey.

Terminal	**Quay Length (m)**	**Terminal Area (*1000 m^2)**	**Maximum Depth (m)**	**Capacity of Vessel Acceptance (Vessel /Year)**	**Rehandling Capacity (*1000 Ton/Year)**	**Container Stacking Capacity (*1000 Ton/Year)**
Haydarpaşa	2765	320	12	2651	5889	269
İzmir	3386	525	13	3640	6419	343
Derice	1092	366	15	862	2288	100
İskenderun	1426	750	12	640	3247	146
Mersin	4725	1097	14	4692	8606	371
Samsun	1756	338	12	1130	2380	50
Bandırma	2706	250	12	4280	2771	50

The capacities of these terminals depend on the area they are located and their surroundings. The material handling equipments used in these terminals depend on both the seaside and yard capacity. The capacities of the terminals managed by the government, including the ones which are privatized, are given in Table 4.

Moreover, Ambarlı container terminal, which is located in European side of Istanbul, is the biggest international container traffic gate of Marmara region. The main container terminals, which are located within Ambarlı container terminal, are Marport (Main, West and East), Mardaş and Kumport. These terminals' general information is specified in Table 5 and their container traffic is given in Table 6.

Table 5: General information of Ambarlı container terminals.

Terminal	**Quay Length (m)**	**Terminal Area (*1000 m^2)**	**Maximum Depth (m)**	**Number of Reefer Socket**	**Rehandling Capacity (*1000 Ton/Year)**
Marport -Main	800	170	14,5	164	770
Marport -West	700	170	14,5	96	630
Marport -East	450	69	13,5	44	300
Martaş	910	194	15	150	550
Kumport	2080	400	15,5	144	1000

Table 6: Container traffic in Ambarlı container terminals (TEU).

	Year				
Terminal	**2004**	**2005**	**2006**	**2007**	**2008**
Marport (total)	770	791	963	1,298	1,541
Mardaş	137	162	198	276	360
Kumport	484	439	531	666	649

Table 7: Material handling equipment in Turkey container terminals.

Terminal	Third Party Equipment	Empty Container Stacking Equipment	Container Stacking Equipment	Spreader	RTG Crane	Palette Crane	Gantry Crane	Mobile Crane	Quay Crane
Bandırma	-	1:8 ton	3:25-42 ton	-	-	3	-	5:5-25 ton	15:3-35 ton
Derince	1 MHC:100 ton	4:8-10 ton	4:25-42 ton	-	-	1	-	8:5-25 ton	9:3-35 ton
Haydarpaşa	1 MHC:65 ton	7:8-10 ton	7:25-42 ton	-	18:40 ton	-	4:40 ton	6:5-25 ton	8:3-35 ton
İskenderun	-	3:10 ton	2:42 ton	-	-	5	-	8:5-25 ton	17:3-35 ton
İzmir	2 MHCs:100 ton	20:8-10 ton	19:25-42 ton	-	10:40 ton	-	5:40 ton	12:5-25 ton	7:3-25 ton
Mersin	2 MHCs: 100 ton	12:8-10 ton	11:40-42 ton	-	14:40 ton	-	3:40 ton	8:5-25 ton	16:3-35 ton
Samsun	1 QC with digger:8 ton	1:8 ton	3:25-42 ton	-	-	-	-	6:5-25 ton	19:3-35 ton
Marport - Main	-	-	2	13	17	-	6	2	-
Marport - West	-	-	2	11	18	-	3	5	-
Marport - East	-	7	9	11	-	-	-	8	-
Martaş	-	-	-	14	8	-	-	10	-
Kumport	-	4:8-9 ton	13:45 ton	11:45-50 ton	12:45 ton		-	12:55-100 ton	-

Finally, Table 7 presents the material handling equipments used in the container terminals of Turkey. These equipments are used for not only loading or unloading vessels, trucks or trains but also for the transportation of containers between storage areas and vessels, trucks or trains. Depending on the container terminal, some of the equipments' capacity is also given in the table. An example for explaining some numbers which are given in the table is like this: Quay Crane for Bandırma terminal is 15:3-35 ton means that there are 15 quay cranes which have capacities ranging between 3 and 35 ton.

Material handling equipments used in the terminals show diversity based on the location of the container terminal and the capacity of quay and yard sides of the container terminal. Table 7 shows that while in some terminals empty containers are stacked by special stacking equipments, in other terminals stacking for any type of containers are accomplished by the same stacking equipments. Moreover, in most of the container terminals which are managed by the government, equipments of third parties such as Mobile Harbor Crane (MHC) and Quay Crane (QC) are included since the capacities or equipments of those container terminals are not sufficient.

Consequently, each container terminal whether it is international or national, has a common layout, operation processes and material handling equipments. Therefore, any possible improvement at any level in a container terminal is applicable in others directly or with some modifications.

1.3 General Approach

Container terminals include several operations and areas interacting with each other. Each one of these operations and container terminal areas leads to a decision problem. Hence, each decision made in a container terminal might be classified under two categories depending on their levels, time periods that are done for and their consequences. Depending on the length of the decision periods, they are categorized as either strategic decisions, which are performed for longer periods, such as terminal location, material handling equipment selection and the number of berths, or operational decisions, which are applied for shorter periods like quay crane allocation to the vessels, storage space allocation and yard crane deployment.

Zhang et al. [6] give the hierarchical levels of operational decisions in a container terminal: berth allocation, schedule and stowage planning of vessels, quay crane (QC) allocation, storage space allocation, location assignment and yard crane (rail tired gantry crane, RTGC) and vehicle deployment. Figure 11 explains the hierarchical level between these decision levels. Each decision level is applied continuously and has an effect on other levels. In addition, while the decision level is getting lower, the frequency of the decision making is increasing.

In this hierarchy, we focus on the operational decisions at the location assignment level, which occur in the storage yard area. The operational decisions which are needed at this level have to be applied continuously, so it is imperative to use methods that require less computational time. The main problem that we study is to determine storage locations for both arrived and relocated containers (which are rehandled during retrieval of several containers or remarshaling) with the objective of minimizing the total cost, which is obtained by minimizing the total number of rehandling.

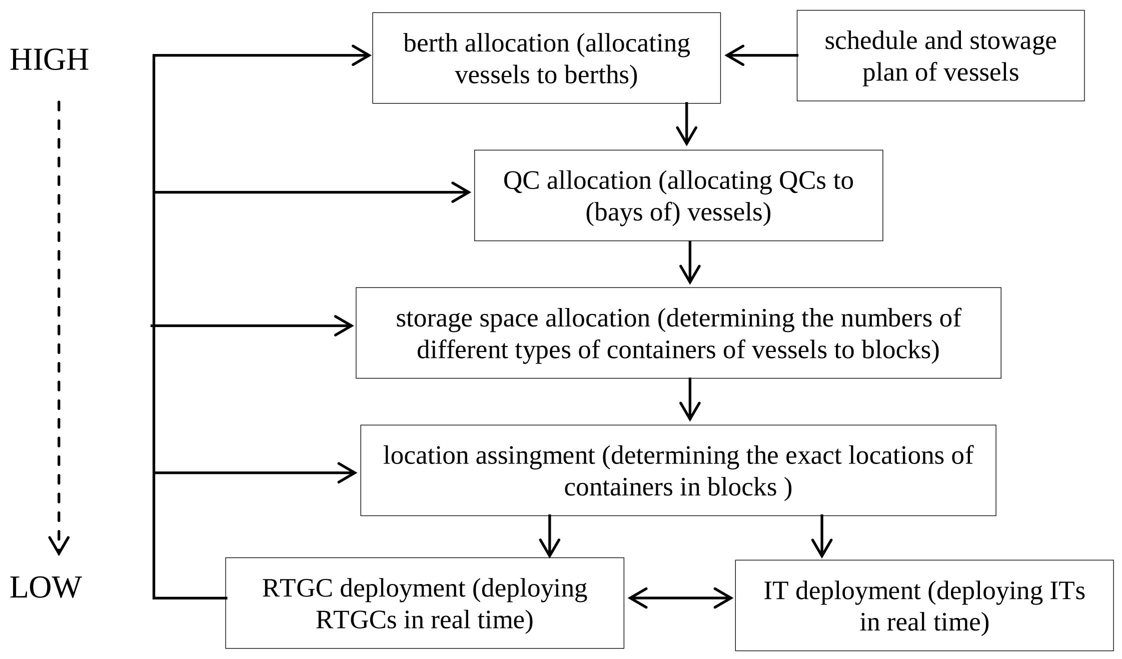

HIGH
LOW
berth allocation (allocating vessels to berths)
schedule and stowage plan of vessels
QC allocation (allocating QCs to (bays of) vessels)
storage space allocation (determining the numbers of different types of containers of vessels to blocks)
location assingment (determining the exact locations of containers in blocks)
RTGC deployment (deploying RTGCs in real time)
IT deployment (deploying ITs in real time)

Figure 11: Hierarchical structure of operational decisions in a container terminal [6].

The remainder of this thesis is organized as follows. In Chapter 2, a literature review about all container terminal operations is given. Chapter 3 provides a definition of the focused problem and lists the possible container terminal scenarios. Chapter 4 discusses the idea of the proposed heuristic algorithms. Chapter 5 presents the performance results of the heuristic algorithms. Conclusion and possible future studies are given in Chapter 6.

Chapter 2

LITERATURE REVIEW

Container terminal optimization is a very popular research area. There are several decision levels; hence it is difficult to solve problems at these different decision levels with an overall optimization model in a container terminal. Therefore, there are plenty of researches focusing on each decision level separately, and there are some new researches integrating two different levels of decisions.

In the literature, there are some review articles about container terminals [3], [4], [7], and [8]. Steenken, Voβ, and Stahlbock [3] review the history of containers, container services, structure of container terminals, types of handling equipment, logistics within the container terminals, optimization methods for container terminals and operations in detail. A literature update of [3] is given by Stahlbock and Voβ [4], in which the structure of the article is basically same with the previous one but additional subtitles are added.

Vis and Koster [7] focuses on the operation sequences in the container terminal, different than [3] and [4], and they review the literature based on optimization hierarchy. Operations starting with arrival of a vessel, later based on container types unloading and loading of a vessel, and finally stacking a container in the storage yard are until it is claimed and any necessary transportation within container terminals are primary topics which are focused in [7].

Vacca, Bierlaire and Salani [8] give an overview of decision problems which arise in the management of a container terminal. Furthermore, they identify several critical issues occurred in some of the busiest container terminals in the world and focuses on competition and cooperation issues that arise between decision makers and market players.

The rest of the literature review in this thesis continues according to the general research areas that are given in [4]. Since we will be dealing with storage allocation problem, this section is discussed in more detail while other sections are reviewed relatively brief.

2.1 Container Terminal Systems

In the literature, there are some studies explaining the container terminal systems in detail. In these studies, researchers take the container terminals as a whole. For example the following articles help us to understand the components of a container terminal with the help of explaining the corresponding and correlated decisions which are made in a container terminal.

Murty, et al. [9] describe a variety of interrelated decisions made during daily operations in a container terminal. In their study, they focus on combining these decisions with the objective of minimizing the overall workload and time during these daily operations in the container terminal. They propose several mathematical models and algorithms in their study and used support tools.

A heuristic approach is applied into container terminals' integration problems by Kozan and Preston [10]. In their research, they present a genetic algorithm (GA), a tabu search and a tabu search/genetic algorithm hybrid to solve the integration problem between container-transfer model and a container-location model to determine both optimal locations and the corresponding handling schedule.

A simulation model of a container terminal and its components are given as a help tool by Bielli, Boulmakoul and Rida [11]. They aim to increase the efficiency of the terminals and the operations with this model. On the other hand, Lau, Chan and Wong [12] formulate a simulation model which combines management and operation processes in the container terminals with the aim of providing a flexible environment for logistics in the container terminals.

2.2 Shipping Planning

The shipping operation starts with the allocation of the berth, which are locations used for loading and unloading operations of vessels. Berth allocation is performed before the arrival of the vessel to the container terminal. During this allocation, length of the vessel, types of handling equipment and positions of containers that are assigned to that vessel are considered. Whenever a vessel arrives to the container terminal later than the expected arrival time, its previous berth allocation has to be modified. Nishimura and Papadimitriou [13] focus on this dynamic problem. They develop a heuristic algorithm based on the genetic algorithm to deal with the berth allocation problem. Moreover, in another study [14], they study the same dynamic berth allocation problem at an extremely busy container terminal in a developing country. This terminal is located in a developing country where the berth capacity is very limited to handle a lot of calling ships. Under this case, they consider berth allocation at the container terminal, which allocates some ships to another container terminal with the objective of minimizing the total service time of ships at these external container terminals. A genetic algorithm based heuristic is developed for this problem and its well performance in reducing external terminal usage is displayed by numerical experiments.

Bae, Park and Kim [15] study the same dynamic berth allocation problem by taking account the real constraints and various dynamic situations. In their study, the main objective is to minimize the cost and they show the similarities and differences between the berth allocation problem, the median location problem and the facility layout problem. Meanwhile, Ganji, Babazadeh and Arabshahi [16] take this NP-hard allocation problem and solve it with branch and bound algorithm. They conclude that branch and bound algorithm is not usable with large sized problems. Then they suggest a method, which uses genetic algorithm as a base and compare the results of the method with results gathered from the branch and bound part.

After the berth allocation problem is managed, the decision has to be made on the stowage plan of the vessel. The vessel stowage planning is done according to the information gathered from the vessel captain and characteristics of the containers that will be unloaded from and loaded onto the vessel, such as their types and weights. The main objective during the stowage planning is to maximize the utilization of the vessel, by minimizing the number of shifts during loading and lowering the turn-around time of the vessels. Ambrosino, Sciomachen and Tanfani [17] propose a model for the stowage planning problem that they define as “Master Bay Plan Problem” with the aim of minimizing the loading time of all containers. Then, they define a three-phased algorithm as a solution procedure and propose methods to obtain stability in the stowage plan. The same problem is formulated as a three-dimensional bin packing problem by Sciomachen and Tanfani [18] for real test cases from the port of Genova, Italy. They propose a heuristic approach with the aim of minimizing the total loading time during stowage planning in these cases. The general idea behind the algorithm is considering containers as items and the vessels as bin while working with stowage plans.

Another problem occurred in container terminals is the assignment of the quay cranes to the vessels that are waiting in the berth locations to be loaded or unloaded, which is named as crane split problem in the literature. The general objective at this level is to minimize the completion time of loading and unloading operations, so that the turnover times of the vessels will be minimized. Conventional quay cranes have a lift capacity of one container and each study in the literature is based on this fact. Kim and Park [19] study this problem with the objective of minimization of the weighted sum of the makespan of the container vessel and the total completion time of all quay cranes. They determine a branch and bound algorithm in order to find an optimal solution to the problem. They also propose a heuristic approach, which is named as greedy randomized adaptive search procedure, to deal with the computational complexity of the branch and bound algorithm. Moccia et al. [20] deal with the same problem and work on the instances generated by [19]. However, different than [19], they propose a branch and cut algorithm, and concluded that it is better than the branch and bound algorithm on the medium sized problems. Sammarra et al. [21] divide the same problem into two problems: routing problem and scheduling problem. A tabu search algorithm is suggested for the routing problem, which results with a local search for the second problem. The results are compared with [19] and [20] to show that the tabu search outperforms the others.

2.3 Transport Optimization

In the container terminals, containers that are incoming from hinterland are transferred from vessels, trucks or trains to container storage yard areas. Moreover, import containers which are stored in the storage yard area transferred from their positions to their assigned vehicles, which might be vessels, trucks or trains. These transfers are named as inter-transportation and are done by the material handling equipment such as internal trucks, straddle carriers, and automated guided vehicles. In the review articles ([3], [4], [7], and [8]) a detailed classification of this problem is given.

Loading containers to and unloading containers from vessels are called as the quayside transportation. The main objectives are finding the schedules and sequences that minimize the overall operation time. For this problem, Bierwirth and Meisel [22] provide a list of applicable algorithms in the literature which are focusing on quayside transportation after they review berth allocation problem.

The transportation occurred in the hinterland side, which contains transferring containers to the trucks or trains from the storage yard, is known as landside transportation. The aim of landside transportation is to allocate a given number of material handling equipments to the operations that balance the workload and time requirements.

Crane transportation is another problem studied under this topic. Cranes used in the container terminal transportation has to be optimized since the number of cranes is fewer than the number of stacks in a storage yard area. Hence, yard cranes are moved between stacks in order to allocate containers to their assigned stack and pick up containers from their stored stacks. During these operations, the main aim is to minimize the waiting time of the transport vehicles and travel times of the cranes, which will be resulted in reduced time for the overall operations.

2.4 Storage and Stacking Logistics

Containers arrive to container terminals with vessels, trucks or trains and are assigned into blocks in the storage yard area. These stored containers remain in their assigned blocks between their arrival to the container terminal and their departure from the container terminal, which is a temporary time period. Containers that are stored in the container terminals might have different types: import, export or transit containers. According to the type of the containers, the storage times and operation flow within the container terminals will be different. However, regardless of their types, containers are stored in the blocks, which are allocated areas in the storage area. A block might be holding different types and number of containers depending on the container terminal characteristics. The assignment of containers to these blocks is done based on their types, lengths and special requirements, like needing electricity.

In a block two main operations are performed: allocation of a container into this block, which occurs with the assignment of a new container into this block, and retrieval of a container from this block, which starts when a container is claimed. Moreover, there may be some remarshaling operations for the containers that are already stored at this block, and rehandlings based on the relocations occurred during each operation. In order to move containers less within a block, their initial assignments in the blocks are important. So, the assignment of these containers to the storage locations within the blocks is another decision problem in the container terminals, which is referred as the storage and stacking problem.

A detailed review about stacking problem in container terminals is given by Dekker, Voogd and Asperen [23]. In this study, they focus on the stacking process within container terminal operations. They examine several variants of container stacking policies in an automated container terminal. They also consider the exchange of the containers during the loading processes in the container terminal. A general introduction on the container terminals, operations and trends are also given in their study. Then, they focus on the stacking policies from different perspectives where they define different performance measures and features.

2.4.1 Storage Allocation

The storage allocation problem has an important role in the container terminal processes. Each type of containers arriving into the storage yard are brings different effect into the container terminal. Export containers, for example, eliminate the uncertainty in the departure times since whenever they arrive to a container terminal, their departure times are known. Also whenever an export container assigned to a block, its position in the retrieval sequence is known. On the other hand, arrival times of the import containers into the container terminals are known even though their departure times are unknown.

Kim and Kim [24] study the storage allocation problem for import containers with the aim of minimizing the expected total number of rehandlings. They suggest mathematical models and solution procedures based on Lagrangian relaxation technique in order to obtain an optimal storage allocation. They analyze the cases where the arrival rates of import containers are constant, cyclic, or dynamic, while satisfying the space requirements. In the study, the presented formulation shows the relationship between the stack height and the number of rehandlings.

Kim, Park and Ryu [25] propose a dynamic programming model to find the storage locations for export containers while considering their weights. Their objective is to minimize the number of relocation movements expected for the loading operation. During the study, they assume that all information is known before the export containers have arrived, their departure times are affected by their arrival times, such as any container which has arrived before that is received before. Finally, relocation for a container is not allowed more than once in their system. The relocation is caused whenever a light container is located on top of a heavy container, since the heavier containers are loaded on the ships earlier. They also provide a decision tree for locating export containers in their system.

The storage space allocation problem in the storage yards of container terminals with the objectives of overall vessel berthing time is minimized and quay cranes throughput rate is maximized are studied in [26]. In their study, the authors divide the problem into two levels, which includes rolling horizon approach. In the first part they work with the assignment of total numbers of containers to blocks. The second part is about allocation of containers of each vessel to blocks. They focus on determining the number of import or transient containers on ships before they are unloaded and allocated to the yard, and the number of export containers before they are brought and stored in the storage yard area in the container yard.

Kim, Ryu and Kim [27] focus on the storage allocation problem with export containers and proposed a simulated annealing based methodology. In the study, the authors derive stacking strategies for export containers while their weights are not known. With their strategies, they aim to reduce the number of rehandlings compared to the traditional same-weight-group stacking strategy. Kang, Ryu and Kim [28] then study the problem with the import containers using the same approach and assumptions in [27]. For import containers, including their dynamic system in departure times, Bazzazi, Safaei and Javadian [29] formulate a mathematical model based on dynamic programming, and then develop a genetic algorithm to solve an extended allocation problem. The solutions found in their study are nearly five percent to the optimum.

2.4.2 Remarshaling

The idea of making arrangements within a block or bay in order to minimize the rehandlings occurred during the retrieval processes is also a major work field. Kim [30] presents a methodology having a dynamic characteristic with this objective where he only assumes a single bay with no extra arrival container. In the same study, assumptions of no other container arrive into that bay during pick-ups and each relocated containers remains in the same bay are considered. Due to the complexity of the problem, he proposes several regression equations and useful tables to estimate the number of rehandlings easily. Kim and Bae [31] propose a methodology in order to provide a better, desirable layout from the current layout of the block that will result in reduced turn-around times for vessels. The transformation between layouts is done by remarshaling moves, which are performed in fewer numbers and in shortest travel distances and defined as clearing moves in [30]. At that point they divide the problem into three levels and solve them by using, respectively, dynamic programming, transportation problem and traveling salesman problem.

On the other hand, Kang et al. [32] propose an intra-block remarshaling with the same purpose where the arrangement is done within the same block in a way that containers to be claimed earlier are placed on top of the other containers. They tried to minimize the time of remarshaling and interference between cranes which are used during remarshaling while avoiding rehandling moves. A simulated annealing algorithm is used for the problem to generate an efficient crane scheduling in a reasonable time for remarshaling.

2.4.3 Retrieval

Kim and Hong [33] address finding pick-up sequences for the containers from a specific bay and locations for relocated containers. They study only one bay and assume that the precedence relations of pickups among blocks are known, relocations can only occur during a pickup, and relocated containers remain in the same bay. They propose a branch and bound algorithm to find the optimal solutions and then propose a heuristic rule for the decision within the solutions. They conclude that the heuristic rule is exceeding the optimal results not more than ten percent and also working less than two seconds in larger cases, where branch and bound algorithm finds the optimal results within one to twenty minutes. Aydin [34] uses a similar branch and bound algorithm for the problem proposed in [33] with a single bay and with only the retrieval operation. He is able to solve about 91% of the problem instances generated. He also proposes several heuristic algorithms and reports an average optimality gap of 6% and 10% for two heuristics that performed the best, respectively.

Lee and Lee [35] propose a three-phase optimization heuristic for a crane to retrieve all the containers with the aim of minimizing the total number of container movements. Their assumption on the initial layout and the retrieving order of containers is similar with that of [33]. In their heuristic, first they generate a simple and feasible movement sequence for the containers. Second, by movement reduction phase, they lower the number of movements in that feasible sequence. Finally, they reduce the total working time by modifying the sequence in the time reduction phase. They argue that examples which are given in [33] are smaller than the usual number of containers in real life container terminals. Therefore they generate new instances in which a total of between 70 and 720 containers can be stored in a block and the blocks are approximately 75% full. Their heuristic algorithm is able to solve even the instances of more than 700 containers, while the final movement numbers are close to their lower bounds. Moreover, Lee and Lee [35] compare their results with that of [33] for only one-bay yard and conclude that their heuristic resulted in fewer movements. Additionally, they generate “upside-down” instances to prove that the number of movements is much higher than that of the randomly generated instances. Unluyurt and

Aydın [36] only compare the results of the randomly generated single bay instances, in total for ten instances, and they do not find the optimal solutions by using their proposed branch and bound algorithm for these instances. They conclude that the heuristic provided by Lee and Lee [35] outperforms the heuristic algorithms of [36] for these instances.

Caserta, Voβ and Sniedovich [37] work with a "blocks relocation problem" while considering same assumptions of [33]. They point out that the layout of a bay is influenced by the arrival of containers into the storage yard area. They propose a dynamic programming algorithm and the corridor method, which enable to solve the large problem instances, but not guarantee that the optimal solution is reached. They conclude that with their solution methodology, fewer numbers of relocations are observed compared to [33].

Finally, a new extension of "lock relocation problem", which is including the weights of the containers, is given in [38]. Hussein and Petering [38] suggest "the global retrieval heuristic" which is embedded in a genetic algorithm based optimization method. They point out, based on the computational results, the importance of relocating heavy containers and number of stacks/bays, in number of relocating and fuel consumption.

Chapter 3

PROBLEM DEFINITION

In this chapter, we will describe the problem considered in this thesis and establish its importance within the container terminal operations. Most activities of container terminals take place in container storage yard areas, where containers are stored temporarily after they are discharged from vessels or before they are loaded onto vessels. The storage yard area of a container terminal consists of multiple blocks. Each block has several bays, and each bay is made up of several rows of container stacks. Generally a block of a container yard is divided up to 30 bays, where each bay has between 3-7 stacks and each stack contains 4-7 rows. While transfer cranes and trucks are used as container handling equipment to move containers in and out of blocks, yard cranes are used for stacking operations within the blocks and can move between bays.

Figure 12 presents a view of a single block, which is located in the storage area. In this block, each container is stored in slots those are defined by using bay, stack, and row numbers as (bay, stack, row). Moreover, while 20ft containers occupy a single bay, 40ft containers occupy two consecutive bays. Therefore, in most container terminals, containers of different sizes are not mixed in the same block because of the safety of containers and the inefficiency in the container handling.

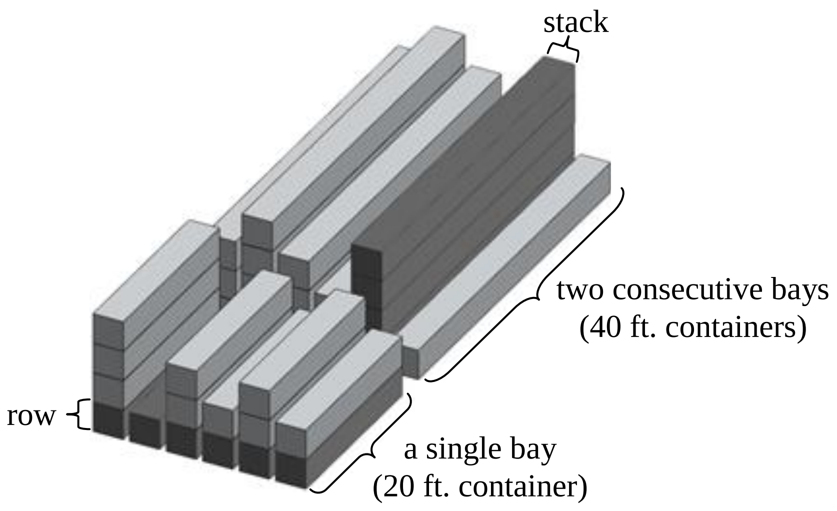


Figure 12: A representation of a block and its components.

Any container that is stored in the container terminal arrives to the side of the blocks by a truck and stacked at a block by a yard crane. Figure 13 gives the schematic front view of a bay with a yard crane while a truck arrives to the block which is loaded with a container and Figure 14 gives the side view of the yard crane. These figures include the idea of the monitoring system of these cranes in the block and the storage area. Sensors which are located at A, B, C and D monitor actions within the block and prevent any contact between containers and sensors which are located at E, F, and G that controls the movements between bays in the storage area. These sensors are used to avoid a contact between two cranes or a crane and vehicles.

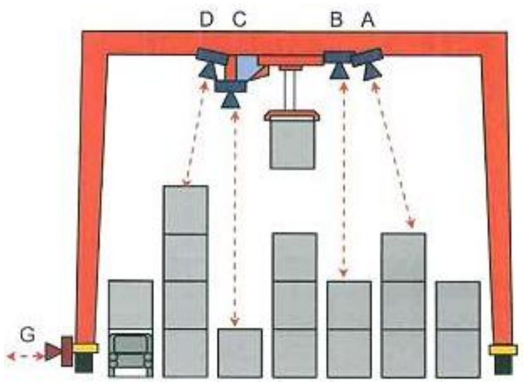


Figure 13: Front view of a bay and a yard crane.

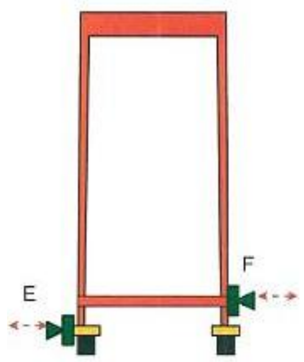


Figure 14: Side view of a yard crane.

Figure 15 is a representation of any possible movements of a yard crane within a block. The yard cranes are not only used for these allocation operations in a block or picking up operations from the block of a container. During the idle periods of the yard crane, it might be used for remarshaling operations (relocating the existing containers within the block) to have a more accurate system during the next allocation and pick-up operations.

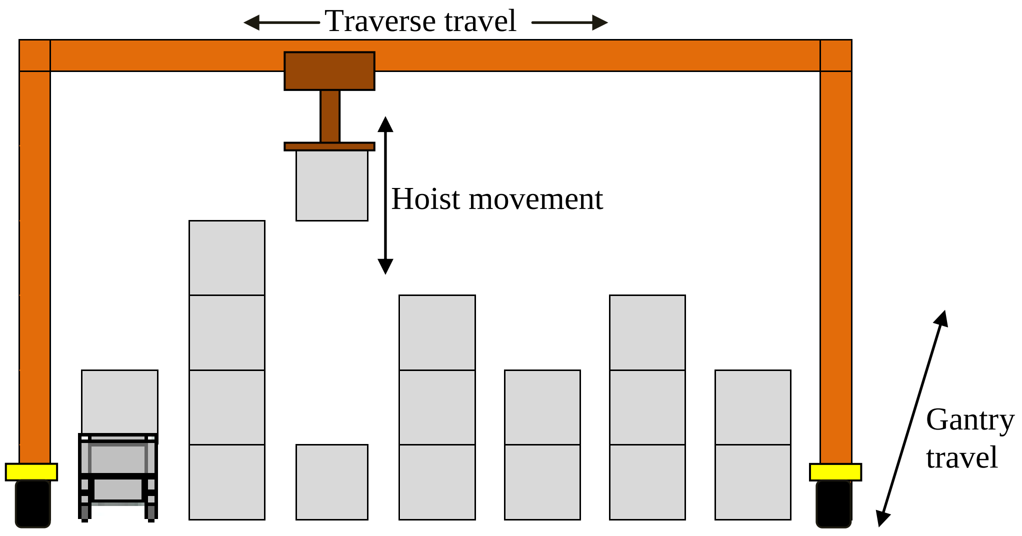


Figure 15: Movements of a yard crane within a block.

Each crane movement is controlled by a crane operator and for any operation firstly, gantry travel is performed to place the crane over the bay in which the operation will be performed. For allocation operation, crane operator performs a traverse travel to position the crane over the truck lane. Then crane is lowered by a hoist movement to reach the container, which is located on the truck, and then lowered up with the container. The container is lifted until it does not touch any other container while the traverse travel is performed. Whenever the lifted container reaches to the required height, the traverse travel occurs to position it on the selected stack and then it is lowered down and placed on that selected stack.

In retrieval operation the movements follow a similar pattern. After the crane is moved to the desired block by a gantry travel, the crane is positioned over the stack in which the claimed container is stored. At that point, in order to access the claimed container, there should not be any other container on it. If there are any containers on the claimed container, some relocation movements should be performed to remove these containers. These relocations cause rehandlings and they are also performed by the crane. Whenever a claimed container is accessible by the crane, the crane is lowered down to pick it up by a hoist movement. The picked up container is then lifted to a required height, moved over the truck lane by a traverse travel and finally loaded on the truck which is waiting in the truck lane.

Yard crane movements are elements of location assignment level, which is at the low level in the decision hierarchy [6], and these movement decisions are made more frequently. In busy container terminals this decision level becomes a holdup for other operations as well. Decreasing the movements of the yard crane, which are usually occurred due to the rehandling of containers, improves the overall system performance in the container terminals. Therefore this decision level is an important working area for researchers.

In most researches, the gantry travel movement is not considered since movements between bays cost much higher than the other movements of the yard crane. But for practitioners in the container terminals, the gantry travel is another significant movement. Thus, during the decisions on yard crane movements, gantry level travel should not be excluded and storage areas should be focused on dependent bays, or on blocks.

In the light of these facts, the related problem that we study in this thesis can be summarized as follows: In a block, in which the initial configuration is known, we are interested in finding the exact locations of the arrived or relocated containers while minimizing the number of rehandlings. We work with the blocks in which only 20 ft. export containers are stored, meaning that any container arrived recently into the block or an already stored container in the block has a known position in the retrieval sequence and occupies only one slot in the block. In this thesis, we consider rehandlings which are occurred during each possible operation of allocation, remarshaling and retrieval. Moreover we apply several different heuristic algorithms under different container terminal policies, which are based on the location allocation decisions and named as scenarios.

Figure 16 illustrates the representation of a block and the first bay is magnified to show how the initial configuration is seen in the block. The block is made of five bays, which are filled with containers in up to six stacks and four rows, as in Figure 13 and 15, and the truck lane is located on the left side. The numbers seen on the enlarged containers represent the position of containers in their retrieval sequence of the container in the block and these ranks have to be updated with the arrival of a new container or after the retrieval operations. In Figure 16, the first stack includes containers those are numbered as {1, 14, 22, and 6}. Hence, the container that is numbered as {1} is going to be retrieved at first, therefore Containers {6}, {22} and 1{4} will be relocated in order to make Container {1} accessible by the yard crane.

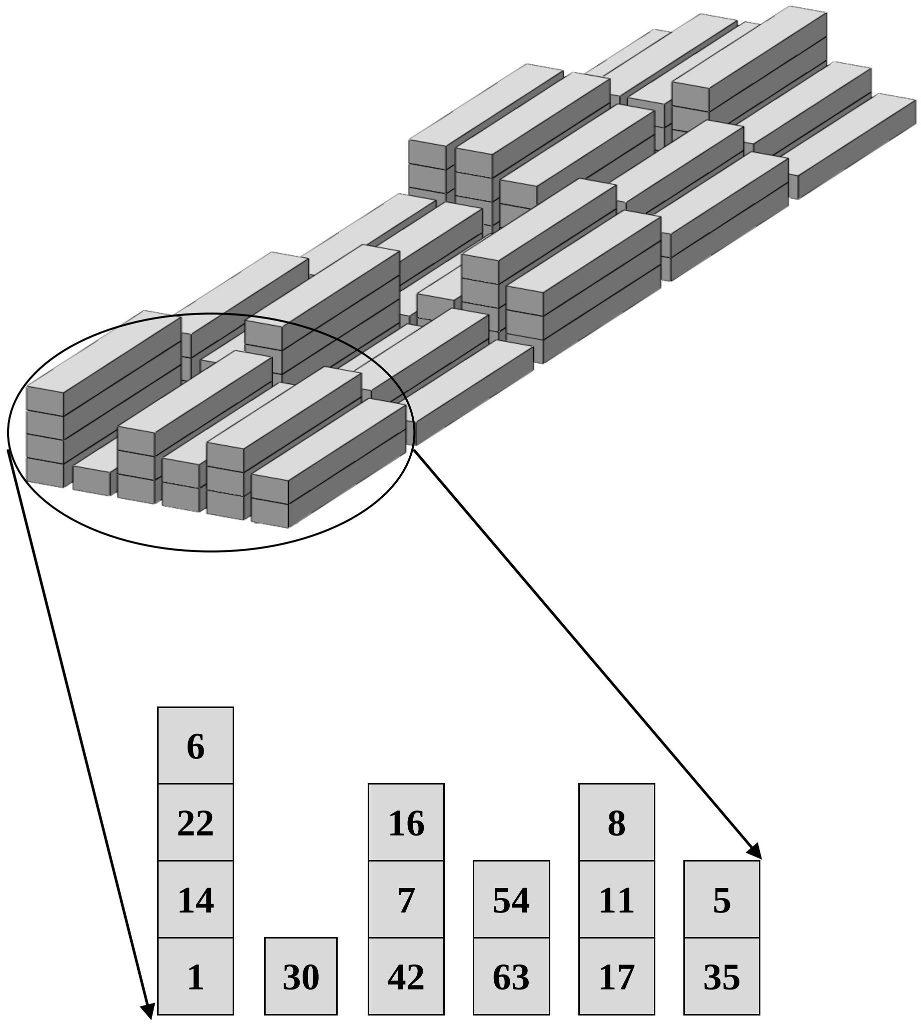


Figure 16: An illustration of a block, with a display of rank numbers.

In Figure 17, the idea of updating rank numbers of containers within a block, before allocation of a container arrived recently into the block or after the retrieval of a claimed container from the block is given. In Figure 17, it is assumed that, in a specified time period, only one container arrives at the block during the allocation operation, and similarly only one container is claimed in the pick-up operation with no remarshaling. In Figure 17 (a), the initial configuration of a bay is given. The updated rank numbers of this configuration with the allocation of a container arrived recently, Container {3}, to the block, are seen in Figure 17 (b). Figure 17 (c) represents the same bay, after the retrieval of Container {1} and this time period ends. The rank numbers for the remaining containers in that bay is then updated (Figure 17 (d)).

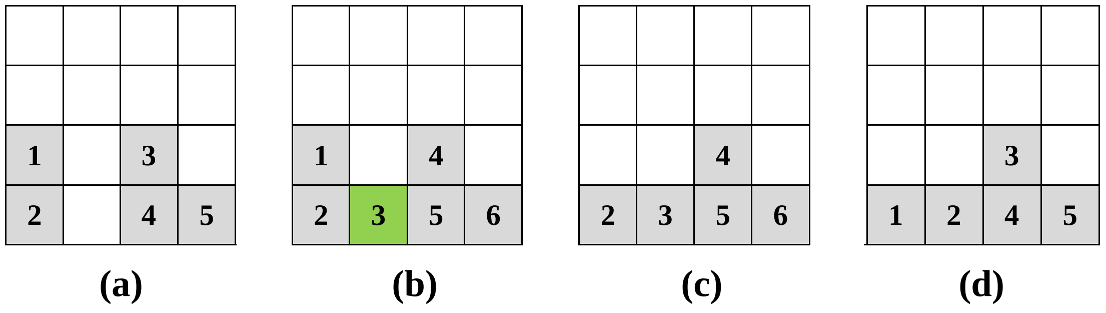


Figure 17: The idea of updating rank numbers of the containers.

In a system like this, we define the objective function for a time period k as in Equation (3.1).

$$Cost_k = r * NumberofRehandlings \tag{3.1}$$

where,

$Cost_k$: Total cost in period k

r: Cost of each rehandling, which is taken as 0.8$ [39].

$NumberofRehandlings$: Total number of rehandling in period k.

Equation (3.2) expresses the details of the main causes of the rehandling that is given in Equation (3.1).

$$\begin{aligned} Cost_k = {} & r * NumberofAllocationRehandlings \\ & + r * NumberofRemarshalingRehandlings \\ & + r * NumberofRetrievalRehandlings \end{aligned} \tag{3.2}$$

where,

$NumberofAllocationRehandlings$: Total number of rehandling occurred during the allocation operation in period k

$NumberofRemarshalingRehandlings$: Total number of rehandling occurred during the remarshaling operation in period k

$NumberofRetrievalRehandlings$: Total number of rehandling occurred during the pick-up operation in period k

Total number of rehandling occurred during allocation operation includes the number of allocated containers into the block in that period. Additionally total number of rehandling happened during the pick-up operation contains the number of retrieved container in that period. Since the rehandling cost during any operation is considered as equal, we will be using Equation (3.1) as the general objective function and the objective functions of each scenario will be generated from it.

3.1 Scenarios

In Chapter 1, we remarked the physical capacities of several international and national container terminals. As seen from their detailed descriptions, these capacity constraints affect every decision in these container terminals such as the number of blocks in the storage yard area or the material handling equipment which are used during stacking operations in these blocks and the retrieval operations policies.

In the location allocation level, there are two main different policy variations. The first variation depends on the rule of the retrieval operation, which is argued under two scenarios: “Sequence Based” and “Group Based”. On the other hand, the storage yard capacity is the second dependent factor depending on the storage yard area capacity. Extra stacks are added in order to increase the capacity of the system only during rehandlings.

Figure 18 demonstrates a schematic view of a bay with an extra stack, which is located next to it. The extra stack policy is combined with the first two scenarios and these are referred to as “Extra Stack, Sequence Based” and “Extra Stack, Group Based”, respectively.

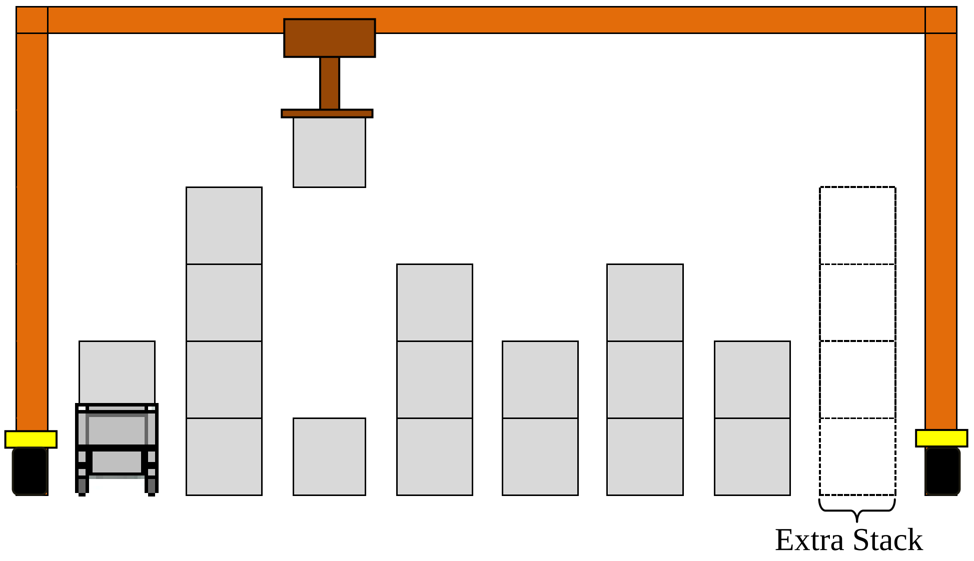


Figure 18: A view of a bay, which has an extra stack.

3.1.1 Sequence Based Scenario (Scenario 1)

In this scenario, during the pick-up operation, a specific retrieval sequence is strictly followed as a policy. In this respect, some of the claimed containers might not be accessible by yard cranes since the containers on top of them could not be relocated within the block because of the capacity of the block. For these cases, whenever a container, which is positioned next in the retrieval sequence, fails to be retrieved, the remaining containers in the retrieval sequence are also considered to be failed.

Figure 19 displays an example for this scenario on a bay, in which the retrieval operation is started for three containers. Since the policy of this scenario is forcing to follow the retrieval sequence strictly, Container {1} will be retrieved at first. Since Container {1} is accessible by yard crane, Figure 19 (a), it is picked up without any extra relocation and Container {2} is listed next in the retrieval sequence. As seen in Figure 19 (b), Containers {14}, {9}, and {8} should be relocated in order to make Container {2} accessible by the yard crane. Unfortunately, the capacity of the bay, which is two slots, is not sufficient for these relocations. Therefore Container {2} is marked as failed to be retrieved. In Figure 19 (c), the final container in this period's retrieval sequence is seen but even the capacity of the bay is enough for making Container {3} accessible by the yard crane, with the relocation of Container {15}, it is considered as failed to be retrieved since Container {2} has failed to be retrieved.

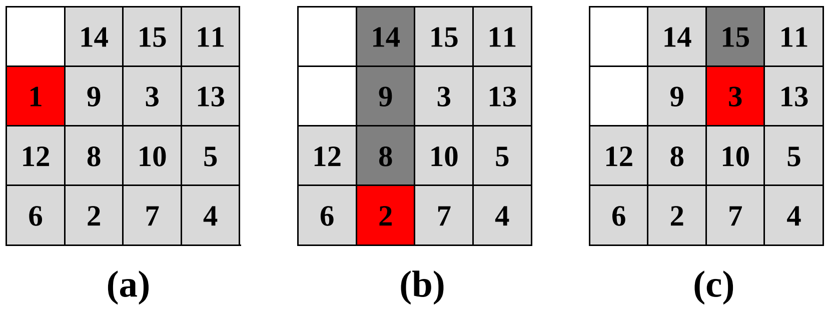


Figure 19: Retrieval under Scenario 1.

In this respect, Equation (3.3) represents the objective function for this scenario, which is a modification of the general objective function, Equation (3.1), by adding a penalty cost for those containers that failed to be retrieved:

$$Cost_k = r * NumberofRehandlings + p * NumberofFailed \tag{3.3}$$

where,

p: Cost of each container that failed to be retrieved,

$NumberofFailed$: Total number of containers which are considered failed to be retrieved in period k.

The penalty cost, p, depends on the contracts between the container terminals and the vessel or transportation companies. In our computational experiments p is taken as 160$ in order to see the effect of failed containers within the system.

3.1.2 Group Based Scenario (Scenario 2)

In this scenario, the claimed containers in a period's pick-up operation are considered as a group. This consideration formats the pick-up operation policy of the container terminals; such as whenever a container fails to be picked-up, it is skipped and considered later within the same period. At the end of the period, there might be still some claimed containers which are failed to be retrieved.

Figure 20 explains the initial iteration of Scenario 2 with an example, in which three containers are asked to be retrieved in the focused period. Container $\{1\}$ is not accessible by the yard crane and the capacity of the bay is not enough (Figure 20 (a)). In this scenario, policy skips to the next container in the retrieval, which is Container $\{2\}$ (Figure 20 (b)). After relocation of Container $\{18\}$, Container $\{2\}$ is picked-up, and next container in the retrieval sequence, which is Container $\{3\}$ is tried to be retrieved. Since Container $\{3\}$ cannot be accessible due to capacity constraints (Figure 20 (c)), the first round on the retrieval sequence is completed with one successful container retrieval, Container $\{2\}$, and two failed container retrieval trials, Containers $\{1\}$, and $\{3\}$. Therefore, a second trial for these failed containers is performed as in Figure 21.

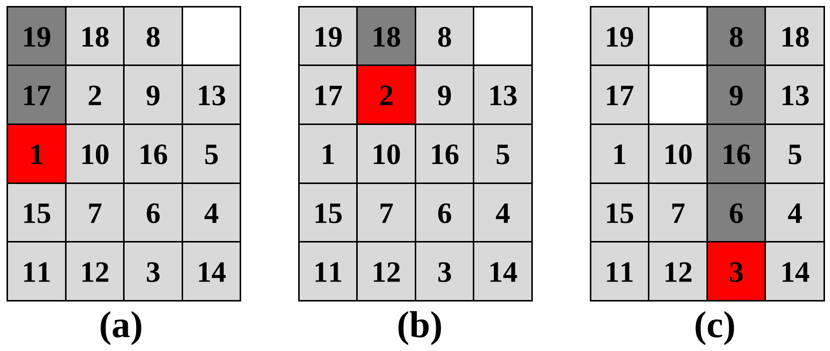


Figure 20: First trial of the retrieval sequence under Scenario 2 in period k.

In the second round, the containers which failed to be retrieved are considered in order. In Figure 21 (a), Container {1} now becomes accessible by relocation of Containers {19} and {17}. Finally Container {3} is tried to be removed again (Figure 21 (b)), but due to the capacity constraint of the bay, it will remain as failed to be retrieved. At the end, while Container {1} and Container {2} are successfully retrieved, Container {3} is failed to be retrieved.

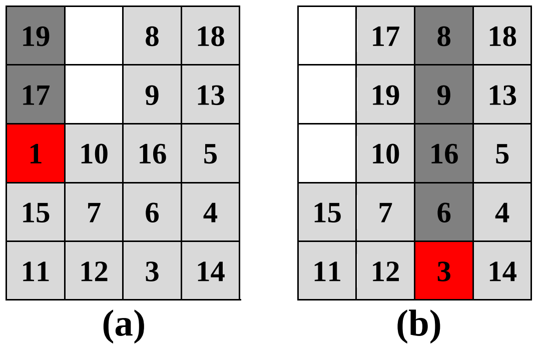


Figure 21: Second trial of the retrieval sequence under Scenario 2 in period k.

Despite the containers to be retrieved in a period are considered as a group, there might be still some containers which will fail to be retrieved. In this respect, the objective function of Scenario 2 will be the same with Scenario 1's objective function, that is, Equation (3.3).

3.1.3 Extra Stack, Sequence Based Scenario (Scenario 3)

In some container terminals, depending on the policy of storage yard, there are extra stacks for each bay, as displayed in Figure 18. The combination of this policy with the sequence based retrieval policy is discussed as Scenario 3. These extra stacks are used during stacking operations in each period. Moreover, at the end of each period, if there is any container stored in the extra stack, they are relocated back into the block. Therefore, depending on the capacity of the bay, any relocation is initially performed within the bay, if possible.

Figure 22 represents an application of Scenario 3, with the same bay configuration given in Figure 19. With this scenario, Containers {2} and {3}, which were failed to be retrieved under Scenario 1, are successfully retrieved. Container {1} is directly picked up due to its position in the bay (Figure 22 (a)). In Figure 22 (b), it is seen that Container {2} is only accessible by relocating containers those are located on top of it. In this scenario, to relocate the containers, initially the available capacity of the bay is used, and then, if needed, extra stack is used. After Container {2} is successfully retrieved, (Figure 22 (c)), Container {3} is considered as the final container in the retrieval operation in this period. Container {15} is relocated within the bay in order to make Container {3} available. After the retrieval of Container {3}, the bay looks like as in Figure 22 (d). Since the current period ends with the removal of Container {3}, the extra stack must be emptied, so

Container {8} should be relocated back in the bay. Therefore, with the help of extra stack, each container, which is listed in the retrieval sequence in a period, is successfully picked-up from the bay.

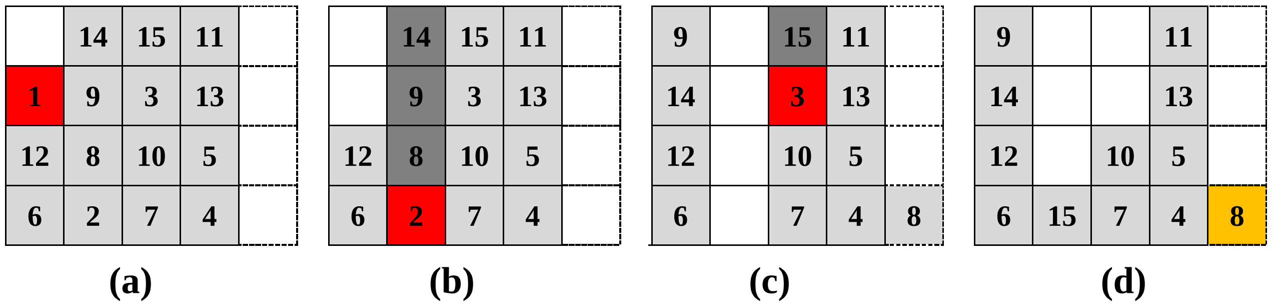


Figure 22: An example for the application of Scenario 3.

The objective function of Scenario 3 which is given in Equation (3.4) is formulated by modifying the general objective function (Equation (3.1)). The modification is caused by the additional cost, F, which is due to the extra stack and this fixed cost depends on the land cost of each container terminal. On the other hand, since each claimed container will be successfully retrieved, the $NumberofFailed$ variable is going to be equal to zero, so it is not considered in the objective function of Scenario 3 (Equation (3.4)).

$$Cost_k = r * NumberofRehandlings + F \tag{3.4}$$

where,

F: Fixed cost of extra stack.

3.1.4 Extra Stack, Group Based Scenario (Scenario 4)

Scenario 4 is the extra stack added version of Scenario 2. This scenario applies group based retrieval policy within the container terminals, which have extra stacks next to each bay in the blocks. Hence, like Scenario 3, all of the claimed containers become accessible by the yard crane, which makes $NumberofFailed$ variable equal to zero during the pick-up operation. Moreover, similar to Scenario 3, extra stacks add a fixed cost to the objective function, and as a result Scenario 4 also uses Equation (3.4) as its objective function.

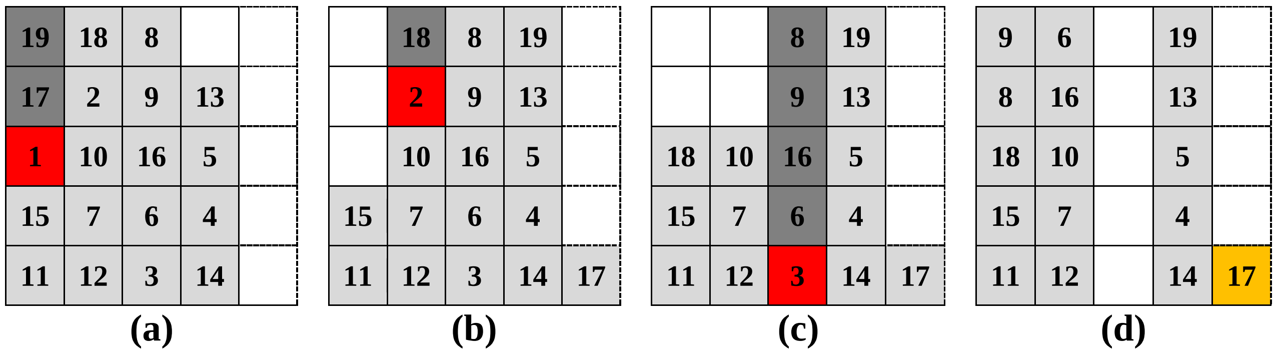


Figure 23: Representation of a retrieval operation in a period by Scenario 4.

Figure 23 is used to represent the application of Scenario 4 on the same example that is used for Scenario 2. In this scenario, each claimed container could be retrieved in the first round of the retrieval sequence. Container {1} is accessible by both using the capacity of the bay and the extra stack (Figure 23(b)). Later, Container {2} and {3} are retrieved by relocating containers which are stored over them within the bay. Figure 23 (d) displays the final bay configuration after the removal of all claimed containers. Since the pick-up operation is ended for that period, Container {17} is the only container that will be relocated back into the block before the next period.

Chapter 4

HEURISTIC ALGORITHMS

In this chapter, we present several heuristic algorithms to solve the problem described in Chapter 3. The heuristic algorithms are categorized under two topics, which are allocation-retrieval relocations heuristic algorithms and remarshaling relocations heuristic algorithms. Allocation-retrieval relocations heuristic algorithms are used during the allocation and pick-up operations, while remarshaling relocations heuristic algorithms are used during the remarshaling operations.

The main idea behind each algorithm depends on the feasible position selection criteria. In allocation-retrieval relocations heuristic algorithms, algorithms are divided into two main groups. The first group uses the differences between the rank numbers of the containers for the feasible position selection (High Rank, Min Rank, High Rank Modified, Min Rank Modified, Smart Heuristic Algorithms, and Tabu Search Algorithm), while the second group selects positions randomly (Random, and Hybrid Heuristic Algorithms) to see the effect of the first group. On the other hand, remarshaling relocations heuristic algorithms are only based on the differences between container rank numbers.

The following notations will be used for explaining the proposed algorithms.

b	: Bay number.
s	: Stack number in the selected bay.
$\min\_rank(b, s)$	: The minimum ranked container that is placed at stack s of bay b.
$ava\_slot\ (b, s)$	: The number of available slots in stack s of bay b.
big_M	: A big number, which is equal to 10,000.
$container\ on\ top\ (b, s)$	: The first container to be accessed at stack s of bay b.

4.1 Allocation-Retrieval Relocation Heuristic Algorithms (Primary)

In this section, the heuristic algorithms proposed for both the allocation operations and pick-up operations will be presented. A period in a block starts with the arrival of new containers into that block and the update of the rank numbers of the containers already stored in this block. The position for each arriving container will be the output of the allocation-retrieval relocation heuristic algorithm. If any container is needed to be rehandled during these allocations, their position within the same block will be determined by the same allocation-retrieval relocation heuristic algorithm.

The following algorithms are straightforward and can be easily applied in practice. In describing the algorithms, a container waiting to be positioned in the block, either an arriving or a rehandled one, is defined as the *waiting container*.

4.1.1 High Rank

This algorithm starts with by finding stacks in the block which have a *min_rank* that is equal to zero. These stacks' *min_rank* are then equaled to *big_M* for the future usage within the algorithm. In the next step, algorithm arranges an *available stack list*, which includes the stacks of the focused block which have at least one available slot.

The stack with the highest *min_rank* is selected among the *available stack lists*. If there is more than one possible stack to be selected then the one with the minimum *b* is selected. If they are in the same bay then the one with the minimum *s* is selected. The main idea in this algorithm is to select the empty stacks first as the feasible slot and then locate the *waiting containers* on the higher ranked containers to make the lower ranked containers easily accessible by the yard crane. The pseudo code for the algorithm is given in Appendix C for allocation operation and in Appendix J for pick-up operation.

Figure 24 illustrates how the algorithm iterates during the allocation operation. Figure 24 (a) displays the initial configuration; while Figure 24 (b) displays the updated configuration after Container {4} and Container {5} have arrived to the block. At that point, each stack is listed in the *available stack list* since the *ava_slots* are (3, 1, 3 and 4) with the corresponding *min_ranks* (6, 1, 7, and 10,000). Stack (4) has the highest min-rank, because it stores no container, and it is selected to locate Container {4} (Figure 24 (c)). After the allocation of Container {4}, feasible slots for Container {5} are searched within the updated *available stack list*. Despite the list contains the same stacks, their corresponding *min_ranks* are updated as (6, 1, 7, and 4). At this time, Stack (3) is selected for Container {5} to be allocated. Figure 24 (d) demonstrates the final configuration after the allocation operation ends. The effect of this operation into the objective function is two rehandlings, which is equal to the number of containers arrived recently, since their allocation did not cause any additional rehandling.

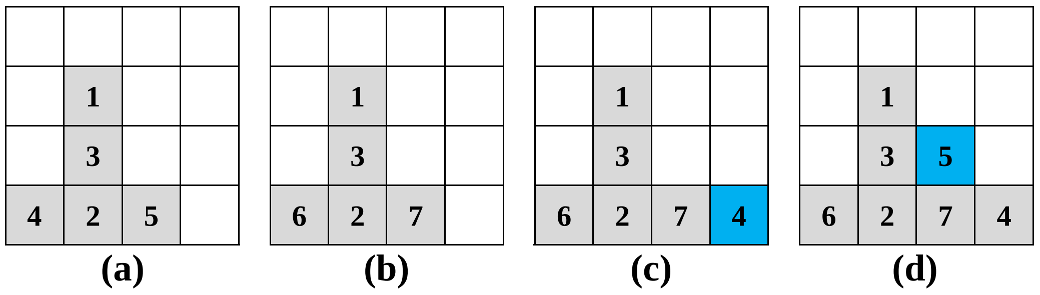


Figure 24: The High Rank Heuristic Algorithm during allocation operation.

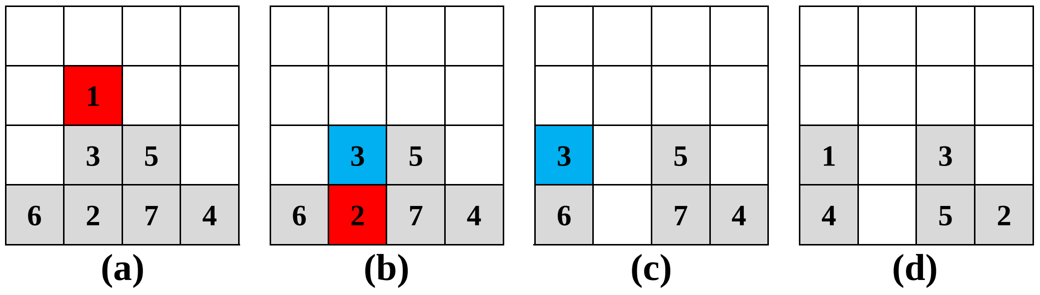


Figure 25: The High Rank Heuristic Algorithm during pick-up operation.

Additionally Figure 25 displays the same algorithm application during the pick-up operation. Starting with the initial configuration (Figure 25 (a)) and two containers in the *pick-up list* in that period, yard crane is able to remove Container {1}. In order to make Container {2} accessible, Container {3} needs to be rehandled (Figure 25 (b)). The High Rank Heuristic Algorithm lists each stack in the *available stack list* with *min_ranks* (6, 2, 5, and 4), respectively. Hence, Container {3} is relocated to the available slot in Stack (1) and Container {2} is picked up by the yard crane (Figure 25 (c)). After reaching the end of the *pick-up list* for this period, the ranks of the remaining containers are updated as in Figure 25 (d). Total effect of this operation to the objective function is three in total, which indicates two rehandlings from the claimed containers and one rehandling for relocating a container to reach a claimed container.

4.1.2 Min Rank

This algorithm starts with by forming the *available stack lists* and then divides this list into two sets depending on the relation between their *min_ranks* and the rank number of the *waiting container*. The first set is named as the *higher stack list*, since it includes the containers having a higher *min_rank* than the *waiting container*'s rank number and the second set is called as the *lower stack list*, since it includes the ones having a lower *min_rank* than the *waiting container*'s rank number.

After dividing available stacks into two sets, algorithm is performed in two steps. In the first step, algorithm searches the minimum *min_ranked* stack within the *higher stack list*. If it exists, the corresponding stack is selected and the second step of the algorithm is skipped. On the other hand, if the first step fails to find a stack within the higher stacks, then the second step is used. This second step basically applies the High Rank Heuristic Algorithm within the *lower stack list*, which initially equals empty stack's *min_ranks* to *big_M* and then chooses the highest *min_ranked* stack within the *lower stack list*.

This algorithm tries to locate the near rank numbered containers on top of each other, to eliminate future relocations. Appendix D gives the steps of the algorithm for the allocation operations while Appendix K presents the pseudo code for the pick-up operation. The following examples demonstrate the steps of the algorithm during allocation (Figure 26) and pick-up operations (Figure 27).

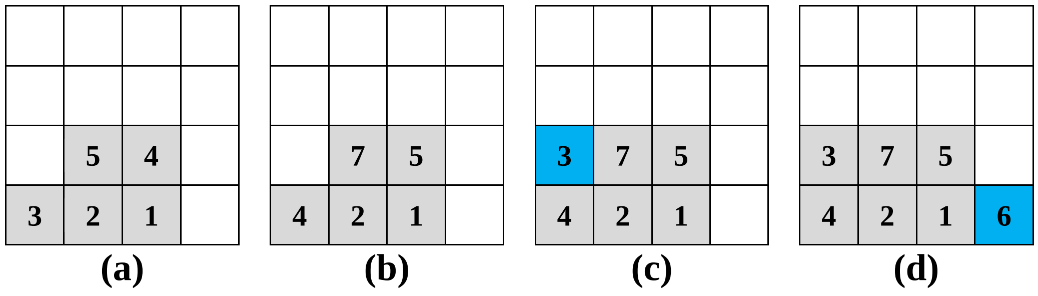


Figure 26: The Min Rank Heuristic Algorithm during allocation operation.

Initial configuration of a bay is given (Figure 26 (a)) with the assumption of incoming containers are ranked as {3}, and {6} during that period's allocation operation. Rank numbers of already stored containers in the bay are updated before the allocation operation starts (Figure 26 (b)). With the update of the rank numbers, all stacks are positioned in the *available stack list*, since each of them has at least one *ava_slot*, and their corresponding *min_ranks* are (4, 2, 1, and 0). Later, for allocation of Container {30}, Stack (1) is listed in the *higher stack list*, and the remaining stacks in the *available stack list* are positioned in the *lower stack list*. Later, a search within the *higher stack list* is completed and since Stack (1) is the only container in that list, it is selected and Container {3} is located at this stack (Figure 26 (c)). For allocation of Container {6}, the *available stack list* is updated but it remains the same, since each stack still has empty slots. However, their *min_ranks* are updated as (3, 2, 1, and 0), which are lower than Container {6}. Therefore, for the next allocation each stack is listed in the *lower stack list*, which means that none of the stacks is listed in *higher stack list*. Hence, algorithm skips the first step that is the search within the *higher stack list*, and updates the *min_rank* numbers of the *lower stack list*, where the updated *min_ranks* become (3, 2, 1, and 10,000). Stack (4), which has the highest *min_rank* in the *lower stack list*, is then selected for the allocation of Container {6}. Figure 26 (d) represents the final configuration of the bay after this allocation operation is completed. The total effect of the algorithm to the objective function is two rehandlings, which are only caused by the allocation of two containers arrived recently.

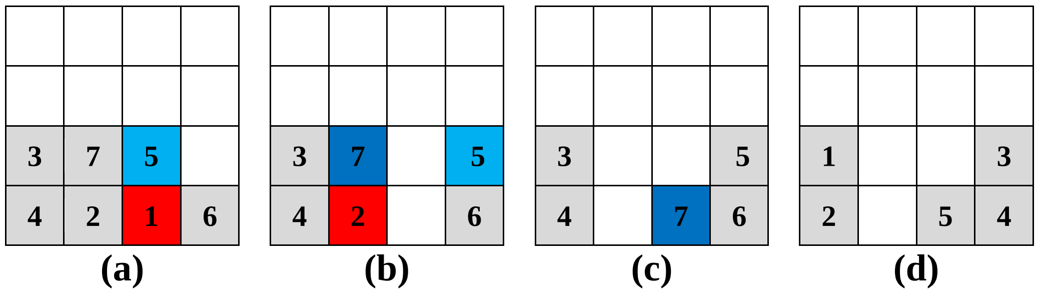


Figure 27: Pick-up operation performing by the Min Rank Heuristic Algorithm.

Figure 27 is an example for the application of the Min Rank Heuristic Algorithm during the pick-up operation. In this period, we assume that only two containers are claimed. Figure 27 (a) shows the initial configuration before the pick-up period, in which claimed containers are not accessible by the yard crane. Container {5} is the first rehandled container, for which each stack is listed in the *available stack list*. These stacks' *min_ranks* are (3, 2, 1, and 6), hence only Stack (4) is listed in the *higher stack lists* and others are included in the *lower stack list*. Since only one stack is listed in the *higher stack list,* it is selected as Container {5}'s new stack. With the pick-up of Container {1}, Container {7} has to be rehandled to reach Container {2} (Figure 27 (b)). New arranged *available stack list* also includes all stacks, where *min_ranks* are (3, 2, 0, and 5), which means that all of the available stacks are listed in the *lower stack list*. Therefore, updated *min_ranks* are (3, 2, 10,000, and 5) for these stacks and Stack (3) is selected for the rehandled Container {7} (Figure 27 (c)). The updated rank numbers for the remaining containers in the bay is given in Figure 27 (d). By using the Min Rank Heuristic Algorithm during the pick-up operation, four rehandlings are added into the objective function, where two of these rehandlings are caused by the retrieval of the claimed containers and the other two rehandlings are performed during making these claimed containers accessible by the yard crane.

4.1.3 High Rank Modified

A modified version of the High Rank Heuristic Algorithm is applied in this algorithm. Stack selection for a *waiting container* is performed by the same rules; which are defined in the High Rank Heuristic Algorithm. However an extra step is applied in this algorithm if the *container on top* of the selected stack has a lower rank number than the *waiting container*. This step is added in order to eliminate the future rehandling, which will occur because of locating a higher ranked container over a lower ranked one.

Application of the additional step requires an extra empty slot, rather than the slot that is reserved for the *waiting container*. This extra slot is used to relocate the *container on top* temporarily. If the High Rank Modified Heuristic Algorithm requires an extra slot like this and there exists such a slot then the *container on top* is removed until the *waiting container* is located into the block. The *waiting container* is placed into the slot, which is emptied by the relocation of the *container on top*, and then the *container on top* is moved over the *waiting container*. If there is no extra slot, the *waiting containers* are directly stored over the *container on top*. The pseudo code for allocation and pick-up operations are given in the Appendix E and Appendix L, respectively.

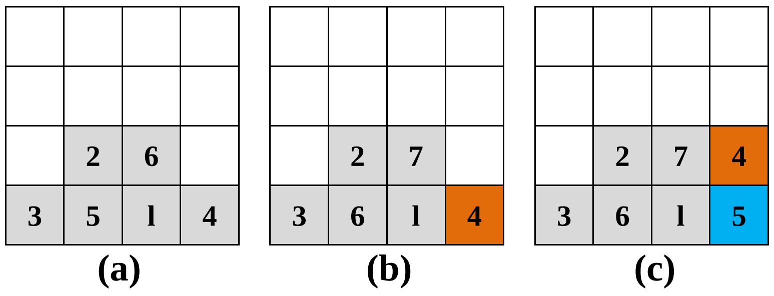


Figure 28: The High Rank Modified Heuristic Algorithm during allocation operation.

Figure 28 demonstrates the application of the High Rank Modified Heuristic Algorithm during the allocation operation. With the given initial configuration (Figure 28 (a)), Container {5} is assumed to be the only container that is arrived recently. In the updated configuration (Figure 28 (b)), each stack is listed in the *available stack list* with the *min_ranks* as (3, 2, 1, and 4) and Stack (4) is selected for locating Container {5} based on the High Rank Heuristic Algorithm. Therefore the rank number of the *container on top* of this stack, which is Container {4}, is compared with the container arrived recently. Since Container {4} has a lower rank than Container {5} and there is an extra slot for temporary relocation, Container {4} is removed. Container {5} is located at that slot and Container {4} is relocated over it, which is seen in Figure 28 (c). The effect of the allocation to the objective function is three rehandlings, where two rehandlings are occurred by relocating the *container on top* and one by allocating the container arrived recently.

The pick-up operation is performed similarly for only one container in this period (Figure 29 (a)). In order to reach the claimed container by the yard crane, Container {7} has to be relocated and becomes the *waiting container*. Since every stack is included in available stacks and their *min_ranks* are listed as (3, 2, 1, and 4). Stack (4) is selected for the *waiting container*, since it has the highest *min_rank*. In this stack, the *container on top*, which is Container {4} has a lower rank than Container {7} and there is an extra slot; thus Container {7} is located in Container {4}'s slot and Container {4} is located over it. Next, accessible Container {1} is retrieved and the system is updated (Figure 29 (b) and (c)). During this operation, three rehandlings are added into the objective function. While one of these rehandlings is caused by the retrieval of the claimed container and the remaining ones are due to the relocation of the *container on top* to eliminate its future rehandling.

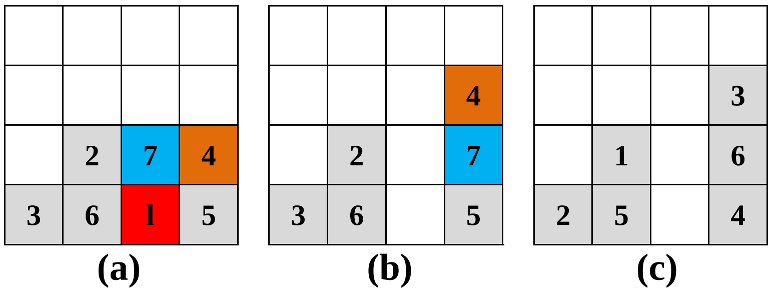


Figure 29: Pick-up operation with the High Rank Modified Heuristic Algorithm.

4.1.4 Min Rank Modified

This algorithm is the modification of the Min Rank Heuristic Algorithm with the addition of the extra step which is given in the High Rank Modified Algorithm. The stack for any *waiting container* is selected by the similar rules as in the Min Rank Heuristic Algorithm. Moreover, there is an extra step to compare the rank numbers of the *container on top*, of the selected stack, and *waiting container*. Appendix F includes a pseudo code for allocation operation of this algorithm and Appendix M for pick-up operations. In Figure 30 and 31, examples are given for the allocation and pick-up operations.

In the focused period, application of the Min Rank Modified Heuristic Algorithm during the allocation operation is given in Figure 30. With the given initial configuration in Figure 30 (a), Container {6} is assumed to be the container arrived recently into the bay. According to this arriving container, the rank numbers of the configuration are updated before the allocation (Figure 30 (b)). For the allocation, the *available stack list* includes all stacks of the bay and these stacks' *min_ranks* are (3, 2, 1, and 5). Since the *higher stack list* is empty, the highest *min_ranked* stack of the *lower stack list*, which is Stack (4), is selected. Meanwhile, the *container on top*'s rank is checked, which is lower than the rank of the container arrived recently. Under this condition the extra step is applied, by which slot of Container {5} is temporarily relocated and this slot is occupied by Container {6} and Container {5} is stored over it. Through this operation, three rehandlings are occurred, where one rehandling is happened during the allocation of Container {6} and the other two rehandlings are occurred with the relocation of the *container on top*, Container {5}.

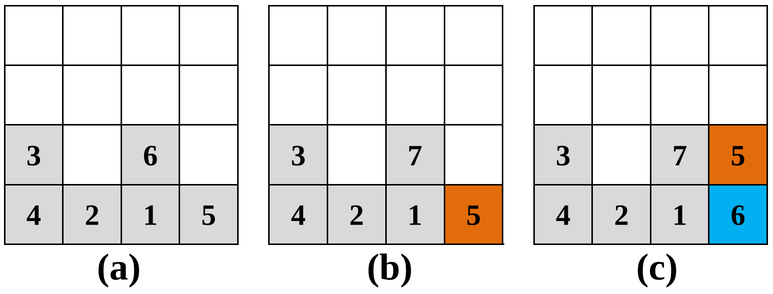


Figure 30: The Min Rank Modified Heuristic Algorithm during allocation operation.

For the given configuration in Figure 31 (a), Container {7} has to be rehandled, while all of the stacks are listed in the *available stack list*. *Min_ranks* for these stacks are (3, 2, 1, and 5), in which there is no stack that can be listed in the *higher stack list*. Therefore, Stack

{4}, which has the highest rank number within the *available stack list*, is selected. For the selected stack, Container {7} has a higher rank number than the *container on top*, Container {5}, which results in the relocation of the *container on top* temporarily. At that point, Container {7} is located at the *container on top*'s slot and the *container on top* is relocated over it (Figure 31 (b)). After these relocations, the claimed container is removed from the bay. The final version of the system is given in Figure 31 (c) after updating the remaining containers' rank numbers. This period's pick-up operation also increases the rehandling numbers of the objective function by three rehandlings. Two of these rehandlings are caused by the relocation of the *container on top*, while the remaining one rehandling is due to the retrieval of the claimed container.

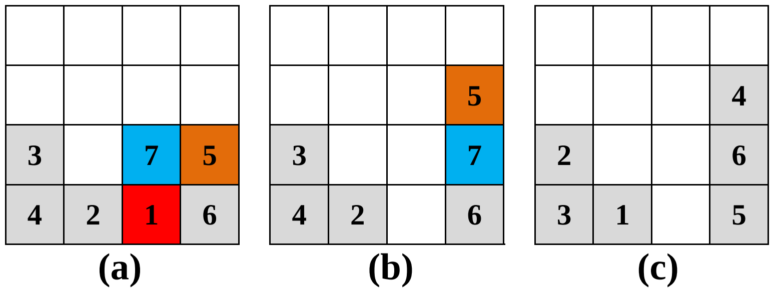


Figure 31: The Min Rank Modified Heuristic Algorithm during pick-up operation.

4.1.5 Random

Different than the heuristic algorithms defined up to this section, this algorithm selects the slot randomly to place *waiting containers* and containers arrived recently. Since the selection is done randomly, a stack with no empty slots might also be selected by the Random Heuristic Algorithm. Whenever a stack with no empty slot is selected, the *container on top* of this stack is relocated permanently by using the Min Rank Heuristic

Algorithm. Then if there are containers stored at or over the selected slot, the Random Heuristic Algorithm first relocates them temporarily. After these relocations, the *waiting container* is located into the randomly selected slot. With locating the *waiting container*, algorithm returns the temporarily relocated containers back to the selected stack. If there is no container that is stored in the selected slot the *waiting container* is located on the top of that stack. Allocation and pick-up operations are both explained in the following two figures, Figure 32 and Figure 33), and also in the pseudo codes, which are represented in the Appendix G and Appendix N, respectively.

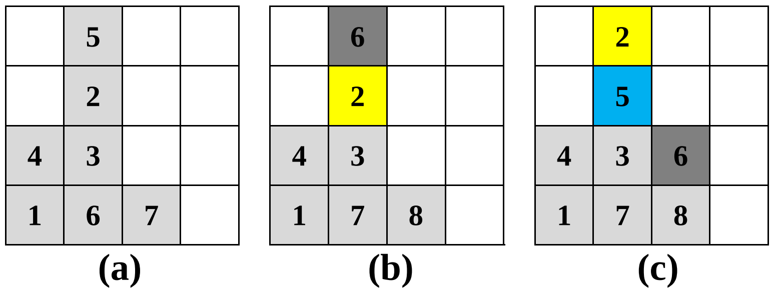


Figure 32: The Random Heuristic Algorithm application during allocation operation.

Container {5} is the container arrived recently into the initial configuration given in Figure 32 (a). The updated configuration of the bay is given in Figure 32 (b), and the randomly selected position for the *waiting container* is Stack (2)-Row (2). Since the *ava_slots* is zero in the selected stack, the *container on top* of Stack (2), that is Container {6}, is relocated by the Min Rank Heuristic Algorithm. Later, Container {2} is temporarily relocated, Container {5} is stored in the selected position and Container {2} is returned to the selected stack (Figure 32 (c)). Total rehandling added to the objective function is four rehandlings. While one of these rehandlings is caused by the allocation of the container arrived recently, the reason of the other three rehandlings is the relocation of Container {2} and Container {6}.

In the pick-up operation, in which only one container is claimed, Container {4} has to be relocated to reach the claimed container. The randomly selected position for it is Stack (3)-Row (3). Since the selected stack has at least one empty slot but the selected position is not empty, the container that is stored in the selected slot, that is Container {6}, is temporarily relocated and Container {4} is located in this slot and later Container {6} is moved back into the selected stack (Figure 33 (b)). Figure 33 (c) represents the updated configuration of the bay after the claimed container is retrieved. This relocation operation adds four rehandlings into the general objective function; only one of these rehandlings is occurred as the retrieval of the claimed container and the other three rehandlings are caused by the relocation of the container that was stored in the randomly selected position and relocation of Container {4}, which was stored over Container {1}.

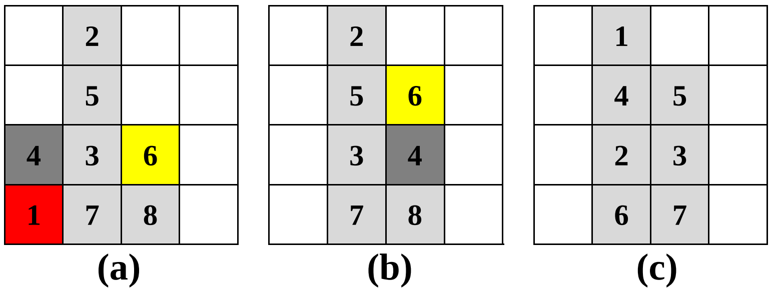


Figure 33: Pick-up operation under the Random Heuristic Algorithm.

4.1.6 Hybrid

In this algorithm, the ideas of the previous heuristics are merged. The algorithm selects the slots randomly for only containers arrived recently. Other than that all of the slot selections are completed by using the Min Rank Heuristic Algorithm.

As a result, the allocation operation differs from the previously proposed heuristic algorithms, for which an example is given in Figure 34. On the contrary of the allocation operation, the pick-up operation is exactly like the pick-up operation of the Min Rank Heuristic Algorithm, which is explained in Figure 27. The pseudo codes for the Hybrid Heuristic Algorithm are given in Appendix H for the allocation operation and in Appendix O for the pick-up operation.

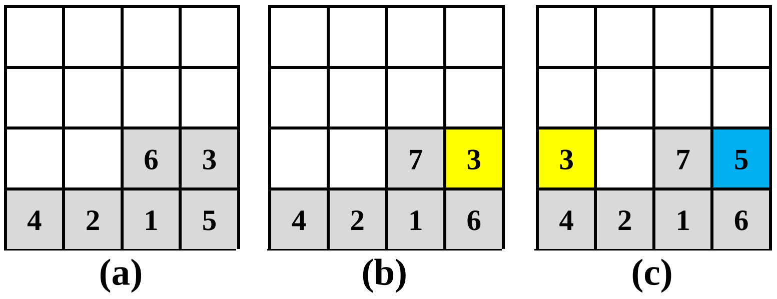


Figure 34: Application of the Hybrid Heuristic Algorithm during allocation operation.

In Figure 34, the application of the Hybrid Heuristic Algorithm during the allocation operation is given. The initial bay configuration of the focused period is given in Figure 34 (a), while the container arrived recently is Container {5}. The randomly selected position for Container {5} that is Stack (4)-Row (3), and the updated configuration is given in Figure 34 (b). As in the Random Heuristic Algorithm, if the selected slot is not empty, the containers stored at that slot or over it should be relocated but this time they are moved by using the Min Rank Heuristic Algorithm. Hence, Container {3} that is in the selected position is relocated into Stack (1), by the Min Rank Heuristic Algorithm and Container {5} is located into its position (Figure 34 (c)). In this operation two rehandlings are occurred, where one of these rehandlings is reasoned by allocation of the container arrived recently and the other rehandling is reasoned for the relocation of the container that is stored previously in the randomly selected stack.

4.2 Allocation-Retrieval Relocation Heuristic Algorithm (Improved)

In this section, we proposed an additional allocation-retrieval relocation heuristic algorithm, the Smart Heuristic Algorithm. This algorithm is described under a new title, since it does not suggest any different stack selection algorithm than the Min Rank Heuristic Algorithm. Differently, this new algorithm adds some steps to be applied within the focused period.

In the Smart Heuristic Algorithm, smart relocations during the allocation operation are added into the algorithm. This algorithm takes the Min Rank Heuristic Algorithm's stack selection rule as a basis, modifies and adds new rules into it. This modification is performed during allocation operations by adding smart relocations before relocating any *waiting container* while the removal operation remains exactly the same as in the Min Rank Heuristic Algorithm. The main reason for taking the Min Rank Heuristic Algorithm as a base is the outperforming results of the Min Rank Heuristic Algorithm for which the detailed results are given in Chapter 5.

Position selection for any *waiting container* is decided under two steps. The first step is performed by listing stacks under the *available stack list*, and then the *higher stack list* as in the Min Rank Heuristic Algorithm. In the second step, any available stack in the *higher stack list* is searched. If any stack is successfully selected, each other available stack's *container on top* is investigated. If any *container on top*'s rank is higher than the *min_rank* of its stack in which it is stored, then it is listed in the *check list.* In the next step, the arranged *check list* is searched in order to find containers which have lower rank numbers than the *min_rank* of the selected stack. Also these containers should have higher rank numbers than the rank number of the *waiting container*. If such containers are found, they are sorted in descending order in the *possible inter-relocation list*. If the selected stack has more than one available slot, the first container of the sorted *possible inter-relocation list* is relocated into the selected stack. Later a new *check list* and *possible inter-relocation list* are formed and if possible a similar relocation is done under the same conditions. These relocation processes are implemented until either the *possible inter-relocation list* is empty or the selected stack has only one empty slot. Then the *waiting container* is stored in the selected stack.

On the other hand, if the algorithm is unable to select a stack in the first step, a stack, which is defined to be a *control stack*, has to be modified in order to be selected. Then the *waiting container* is assigned into that stack. In order to decide on which stack will be the *control stack*, the stacks are listed in descending order based on their empty slots. By forming a list like this, those stacks storing less number of containers are listed as first stacks to be modified. In this list, which is named as the *control list*, whenever two or more stacks have equal number of empty slots, the one with the highest *min_rank* is modified first within this group. The first stack in the *control list* is selected if its *min_ranked* container is stored at the bottom row of its stack or any container that are stored under *min_ranked* container has a higher rank number than the *waiting container*. The main reason behind this selection is finding an available slot for the *waiting container* by making few relocations and causing few future relocations. Any container in the *control stack* can be relocated within the block by causing no additional rehandling in future. This can be achieved by relocating these containers into stacks, which have higher *min_ranks* than the relocated containers. This algorithm is described in a pseudo code and is given in Appendix I and in Figure 35.

Allocation operation under the Smart Heuristic Algorithms is illustrated in Figure 35. For the initial configuration of a bay given in Figure 35 (a), a container arrives into the bay, which has a rank number of 6. First of all, the rank numbers are updated (Figure 35 (b)), and stack selection is performed by following the previously defined steps. Stack (4) is selected based on the rules defined in the Min Rank Heuristic Algorithm. Later Containers {4} and {7} are included in the *check list* but only Container {7} is included in the *possible inter-relocation list*. Having more than one empty slot in the selected stack enables the algorithm to relocate Container {7} into the selected stack. After this relocation, the *possible inter-relocation list* becomes empty indicating the end of the relocation process. Then the allocation of Container {6} into Stack (4) is completed (Figure 35 (c)). At that point, the arrival of Container {9} starts and the first step fails to find a stack within the block to allocate it since no stack is listed in the *higher stack list*. Hence the second step is applied, which sorts the stacks as Stack (1, 3, 2, and 4) under *control list* and Stack (1) is selected as the *control stack*. With successful relocation of Container {2}, which was stored in the *control list*, the *control stack* becomes the selected stack and Container {9} is allocated into it (Figure 35 (d)). In the same figure, the relocation of Container {2} is shown. For the stack selection for Container {2} available stacks which are different than the stack that was storing Container {2} are searched, and they have the following *min_rank* numbers (1, 3, and 6). Later the stacks having higher *min_rank* numbers are investigated, which are Stack (3) and Stack (4) with the corresponding *min_ranks* 3 and 6.

Finally, the one with the lowest *min_rank*, which is Stack (3), is selected and the relocation is completed. With the completion of the allocation, four rehandlings are added into the general objective function. Two of these rehandlings are added by allocation of Container {6} and Container {9} and the remaining two rehandlings are added by the additional smart relocations occurred during these allocation operations.

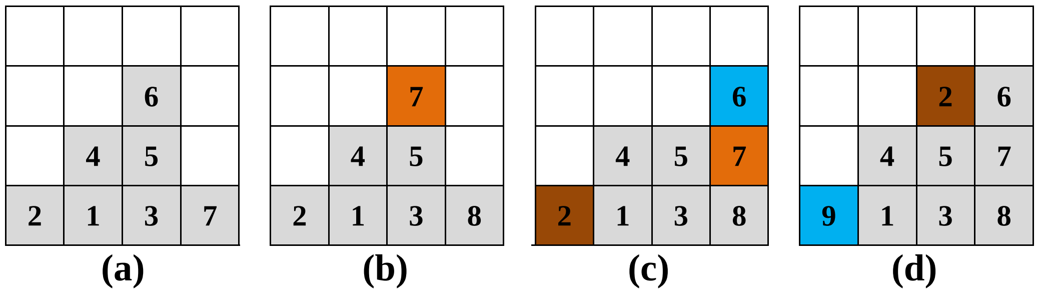


Figure 35: The Smart Heuristic Algorithm during allocation of Containers {2} and {9}.

4.3 Tabu Search Algorithm

In this algorithm, the basic tabu search algorithm is applied for the arrangement of the initial configuration in a period, in which the Smart Heuristic Algorithm is applied. The Tabu Search Algorithm uses a neighborhood search procedure to move iteratively from the initial configuration of a block to another configuration until the stopping criterion is satisfied. During the search, explored configurations are not searched recurrently through the use of memory structures of the algorithm, which stores explored configurations.

The Tabu Search Algorithm, at first, calculates the initial layout configuration's objective function value, which is named as *initial_cost*. Meanwhile, during the application of this algorithm, each configuration’s objective function is calculated by the Smart Heuristic Algorithm. Then neighborhoods are defined as configurations, which are resulted by all possible relocations. These relocations are resulted by relocating each container that is located on top of a stack into each other stack in the block. At this point, there are two possible cases where a neighborhood is not created. The first case, Case 1, occurs whenever a stack contains no container. The second case, Case 2, arises whenever a container has to be relocated into a stack with no empty slot. For example, a bay with four stacks has 12 possible neighborhoods, which is a permutation of the number of stacks in groups of two.

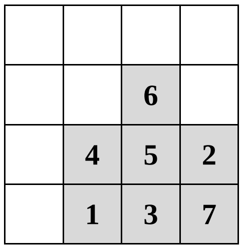


Figure 36: Neighborhood creation failure, Case 1.

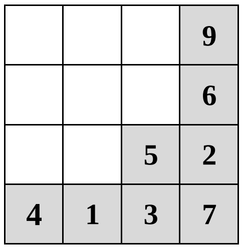


Figure 37: Neighborhood creation failure, Case 2.

Figure 36 displays an example of Case 1. In bay of four stacks, three of the stacks have containers and at least one empty slot, while the remaining stack contains no containers. In this configuration, there are nine neighborhoods, where three possible neighbors are eliminated since no containers can be relocated from Stack (1) to others. In Figure 37, an example of Case 2 is given, in which again three of the stacks contain containers with at least one empty slot but the last stack has no available slots. Therefore there are again nine neighborhoods, but this time the reason is that any container that has to be relocated from Stack (1), (2), and (3) are unable to located into Stack (4).

After deciding on the neighborhoods, the Tabu Search Algorithm calculates the objective function values of each successfully generated neighborhood. Later the neighborhood search starts within these objective function values, by finding the one with the minimum objective function value, which is named as *min_neighborhood*. The algorithm performs its steps with this selection depending on the possible scenarios. These scenarios are depending on the objective function values of the recently selected neighborhood and the initial configuration. In the first scenario, algorithm faces with a *min_ neighborhood* which has a cost lower than or equal to *initial_cost*. The Tabu Search Algorithm checks whether this move is in the tabu list or not. If it is not in the tabu list, algorithm assigns the *min_neighborhood*'s cost as the new *initial_cost*, places this neighborhood into tabu list with defined tabu tenure, which is initialized as the number of stacks in the block. On the other hand, if the move is in the tabu list, the aspiration criterion is checked, which states that if the objective function value of *min_neighborhood* is lower than *initial_cost* then, accept this value as the new *initial_cost*. With the acceptance of this move, the tabu tenure of this neighborhood is set because of the memory structure of the Tabu Search Algorithm. Whenever the aspiration criterion is not valid, the next minimum neighborhood is considered as the new *min_neighborhood*, and new neighborhood is analyzed under the possible scenarios. The second scenario is the termination step of the Tabu Search Algorithm and this happens when the *min_neighborhood* is higher than the *initial_cost*. The termination of the algorithm means that the configuration of the *initial_cost*, at that level, is taken as the initial configuration of the block and the operations of the period performed on it. Any relocation occurred while achieving this configuration is also added into the general objective function.

4.4 Remarshaling Relocation Heuristic Algorithms

In a period, during the idle times of the yard cranes these remarshaling relocation heuristics are applied. The idea of these heuristics is to reorganize the containers which are stored in the block to make the *min_ranked* containers accessible. This idea is established by relocating the containers which have higher rank numbers and are currently stored over the lower rank numbered containers, into other stack to prevent future relocations. All of the following remarshaling algorithms are based on the differences between rank numbers of the containers and processed either at the beginning of a period, before any allocation operation, or between allocation and pick-up operations within the same period.

4.4.1 Remarshaling High

In this remarshaling heuristic algorithm, each stack's *container on top* is checked to see whether its rank number is higher than the corresponding stack's *min_rank* or not. When it is higher, the *container on top* is relocated by the High Rank Modified Heuristic Algorithm, if it does not cause any more future rehandling.

The remarshaling algorithm is applied into each stack depending on their stack and bay numbers. Lower ranked stacks and bays have to be arranged at first since they are assumed to be closer to the truck lane. Moreover, each stack's *container on top* is only visited once during this heuristic algorithm. Figure 38 and the pseudo code in Appendix P, are given to describe the algorithm in detail.

In Figure 38 (a), an initial configuration is given during the idle time of the yard crane, before either an allocation or removal operation. Therefore stacks, starting from small rank numbers, are explored to see whether there is a remarshaling movement or not.

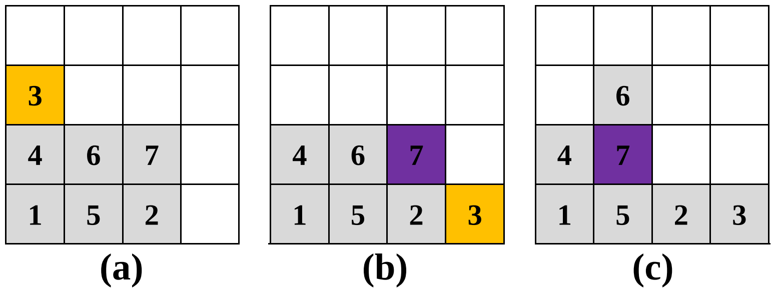


Figure 38: An example for the Remarshaling High Heuristic Algorithm.

At first, Stack (1) is searched and Container {3} is selected since its rank number is higher than the stack's *min_rank*, which is 1 (Figure 38 (a)). Later, Container {3} is located into the empty stack, which is selected by the High Rank Modified Heuristic Algorithm. Figure 38 (b) displays the relocation of Container {3} into Stack (4). Later the next stack's *container on top*, that is Container {6}, which also has the higher rank number than its corresponding stack's *min_rank*, is considered. But since there is not any stack that has a higher *min_rank* number than Stack (2), Container {6} is not relocated into another stack. After the investigation of Stack (2) is completed, Stack (3) is searched and Container {7} is selected because of its rank number, which is higher than the *min_rank* of the corresponding stack. Stack (2) is selected for this container's relocation by the High Rank Modified Heuristic Algorithm, but the *container on top* at the selected stack, that is Container {6}, has a smaller rank number than Container {7}. In a position like this, the algorithm stores them in the reserve order in order to eliminate future rehandling of Container {7}, which will occur while making Container {6} accessible. With these relocations, the arranged configuration results in as in Figure 38 (c) and then the next operation, either allocation or pick-up, is applied. This remarshaling operation adds four rehandlings into the objective function value for the period it is applied. One of these rehandlings occurs for relocating Container {3} and the remaining three rehandlings occur during the relocation of Container {7}, where one rehandling is incurred while placing Container {7} into Stack (2) and the other two rehandlings are caused by removing Container {6} from its current slot and positioning it back into the selected stack.

4.4.2 Remarshaling Min

This heuristic follows the same rules with the Remarshaling High Heuristic Algorithm about which containers on top are going to be remarshaled and in which order containers on top of the stacks are searched. The only difference is the relocating of the *container on top*, if necessary. In this algorithm, the remarshaling operations are performed by using the Min Rank Modified Heuristic Algorithm. An example for the application of this remarshaling heuristic algorithm is given in Figure 39 and the pseudo code of the heuristic algorithm is presented in Appendix Q.

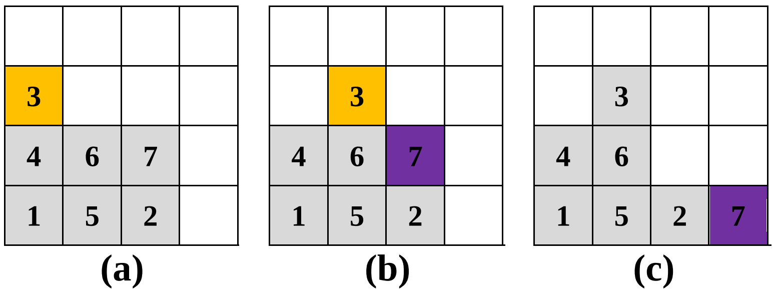


Figure 39: Illustration of the Remarshaling Min Heuristic Algorithm.

Figure 39 (a) illustrates the same configuration that is given in Figure 38 (a), and Figure 39 (b) displays the remarshaling of Container {3}. Stack (2) is selected as the new stack for

Container {3} by the Min Rank Modified Heuristic Algorithm and it is relocated there. Container {3} now becomes the *container on top* of Stack (2), which means that Container {6} is not included in the algorithm and since Container {3} is already relocated this stack is skipped during this remarshaling operation. Later, Stack (3) is investigated and Container {7} is relocated into the empty stack, which is Stack (4). Container {7} is removed from Stack (3) since it has a higher rank number than Container {2}, which is *min_rank* of Stack (3), and there is a stack for it to be stored in without causing any future rehandling. Since each stack is scanned for a single time during the remarshaling operation for this period, the algorithm ends. As a result, the Remarshaling Min Heuristic Algorithm adds only two rehandlings into the objective function value, which are caused by the relocation of Container {3} and Container {7}.

4.4.3 Remarshaling Several

This remarshaling algorithm is a modification of the Remarshaling Min Heuristic Algorithm. In this algorithm only the search of the *container on top* differs from the Remarshaling Min Heuristic Algorithm. The search starts similar, in which the lower numbered stacks and bays are searched for once, but then it continues to search the same stacks in the same order. The search loop is continued until there is no possible relocation during the remarshaling operation of this period.

Figure 40 demonstrates the application of this algorithm on the same example given for the Remarshaling Min Heuristic Algorithm in order to see the difference in the search procedure. Moreover, the Remarshaling Several Heuristic Algorithm is given in Appendix R.

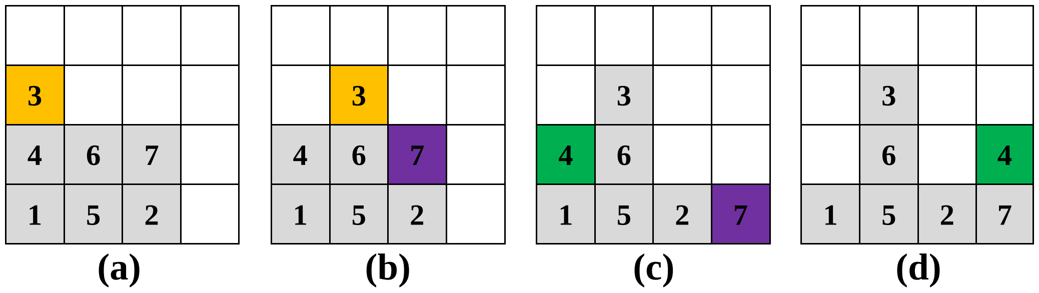


Figure 40: Application of the Remarshaling Several Heuristic Algorithm.

Since the layout given in Figure 40 is same with the layout given in Figure 39 for the Remarshaling Min Heuristic Algorithm , configurations which are given in Figure 40(a), (b) and (c) are exactly the same with Figure 39 (a), (b) and (c). These steps are performed during the first search of the stacks by the Remarshaling Several Heuristic Algorithm as they are completed in the Remarshaling Min Heuristic Algorithm. Later, a second search is performed in the Remarshaling Several Heuristic Algorithm which has the layout as in Figure 40 (c). In this search, Container {4} is specified as the one to be relocated, and the algorithm selects Stack (4) for this relocation. With this final relocation, there is no *container on top* in the block that needs to be relocated, which means that the remarshaling operation is completed (Figure 40 (d)). Subsequently, the number of rehandling added to the objective function value by the Remarshaling Several Heuristic Algorithm is three, where first two rehandlings are reasoned by the relocations of Container {3} and Container {7} as in the Remarshaling Min Heuristic Algorithm which are occurred in the first search and the final rehandling is caused by the additional rehandling of Container {4} that is done during the second search.

4.4.4 Remarshaling Smart

Different than previously defined three remarshaling algorithms, this algorithm forms a *remarshaling list*. This list includes all the containers on top which have higher rank numbers than the *min_rank* of the stack they are stored in. Later these containers which are listed in the *remarshaling list* are sorted in descending order. Each container that is stored in the *sorted remarshaling list* is a candidate for remarshaling, and is referred to as the *remarshaling container* during the remarshaling operation. Through the remarshaling operation of each *remarshaling container*, the stack selection is completed by forming initially the *available stack list* and then the *higher stack list.* If there is any stack in the *higher stack list*, the one which stores the minimum *min_ranked* container is selected. On the other hand, if there is not a higher stack, an empty stack is searched. At that level, if such an empty stack exists then it is selected. In the case of finding more than one empty stack, the one with the smaller bay number is selected. If they are in the same bay then the one with the smaller stack number is selected. Throughout the stack selection for the *remarshaling container*, there are two possible scenarios: a stack is successfully selected for the relocation of the *remarshaling container*, so it is relocated into the selected stack or the stack selection is failed, hence the *remarshaling container* remains in its own position. Regardless of the output of the stack selection, each *remarshaling container* is removed from the *remarshaling list* after stack selection is performed as described above.

The Remarshaling Smart Heuristic Algorithm is performed until the *remarshaling list* is emptied. The example given in Figure 41 explains the application of the Remarshaling Smart Heuristic Algorithm. Besides, the pseudo code defining this remarshaling algorithm is given in Appendix S.

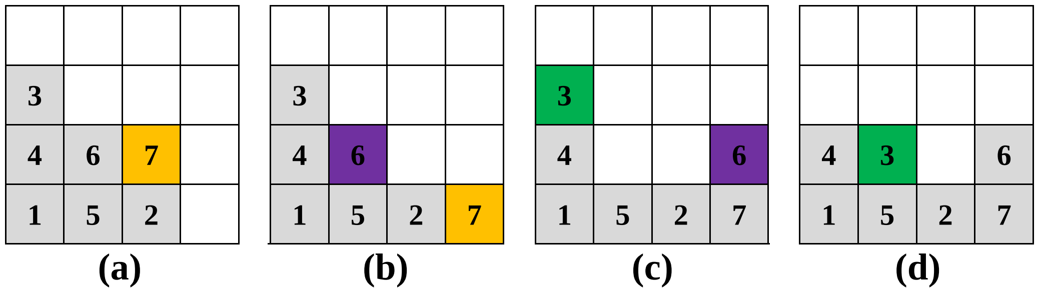


Figure 41: A representation of the Remarshaling Smart Heuristic Algorithm application.

To show the difference of the Remarshaling Smart Heuristic Algorithm, the same initial configuration that is given for the previously defined remarshaling heuristic algorithms is used (Figure 41 (a)). First, the *remarshaling list* is achieved by including Containers {3}, {6}, and {7}. Later this list is sorted and becomes Containers {7}, {6}, and {3}, where Container {7} becomes the first *remarshaling container*. Meanwhile, each stack in the bay is listed in the *available stack list*, but none of them is listed in the *higher stack list*. Therefore, an empty stack is searched and Stack (4) is found, which is then selected for the relocation. Container {7} is relocated into Stack (4) and removed from the *remarshaling list*. With the removal of this container from the *remarshaling list*, Container {6} becomes the new *remarshaling container* (Figure 41 (b)). The *available stack list* is updated for the stack selection for Container {6}'s relocation, which includes all stacks. Stack (4) is then listed in the *higher stack list* and since it is the only stack in this list, it is selected for the relocation of Container {6}. After relocating Container {6} into Stack (4), it is removed from the *remarshaling list* and the remaining container, Container {3}, is named as the final *remarshaling container* (Figure 41 (c)). For Container {3} while all of the stacks are included in the *available stack list*, only Stack (2) and Stack (4) are listed in the *higher stack list*, since their *min_ranks* are higher than Container{3}. The stacks in the *higher* *stack list* have the min-ranks of 5 and 6, respectively; therefore because of having the minimum *min_rank*, Stack (2) is selected. Container {3} is relocated into the selected stack and removed from the *remarshaling list* (Figure 41 (d)). The *remarshaling list* is emptied and the remarshaling operation ends. This operation added one rehandling for each relocated containers, which means that in total three rehandlings are added into the general objective function value by the Remarshaling Smart Heuristic Algorithm.

4.4.5 Remarshaling Smart_2

Remarshaling Smart_2 Heuristic Algorithm is a modified version of the Remarshaling Smart Heuristic Algorithm. The only difference between these heuristic algorithms is that the *remarshaling list* is updated in the Remarshaling Smart_2 Heuristic Algorithm. The Remarshaling Smart_2 Heuristic Algorithm forms the *remarshaling list* to decide which *container on top* will be relocated first within the bay, as it is formed in the Remarshaling Smart Heuristic Algorithm. Moreover, this *remarshaling list* is updated after any container is removed from this list. During the update process, any container that is removed from the *remarshaling list* is not listed in this list again. The pseudo code for this heuristic algorithm is given in Appendix T and Figure 42 demonstrates the application of the Remarshaling Smart_2 Heuristic Algorithm.

In the given initial configuration (Figure 42 (a)), Containers {4} and {6} are listed in the *remarshaling list*. After sorting this list in the descending order, Container {6} is selected as the *remarshaling container*. Despite each stack is listed in the *available stack list*, none of them is listed in the *higher stack list*. Therefore, the only empty stack, that is Stack (4), is selected for the relocation; Container {6} is moved into this stack and removed from the *remarshaling list* (Figure 42 (b)). Following the removal of a container from the *remarshaling list*, this list is updated. In the updated list, Container {4} and Container {5} are included, and after sorting these containers, Container {5} is selected as the *remarshaling container*. Therefore, the new *available stack list* and the new *higher stack list* are formed. Stack (4) is now becomes the only stack in the *higher stack list* and is selected for the relocation of Container {5}, which is removed from the *remarshaling list* (Figure 42 (c)). At that point, the updated the *remarshaling list* includes only Container {4}, for which again all stacks are in the *available stack list* but only Stack (4) is in the *higher stack list*. Figure 42 (d) represents the relocation of the remaining *remarshaling container*, which is Container {4}, into the selected stack that is Stack (4). The effect of the Remarshaling Smart_2 Heuristic Algorithm for the general objective function is three additional rehandlings due to the relocation of the *remarshaling containers*.

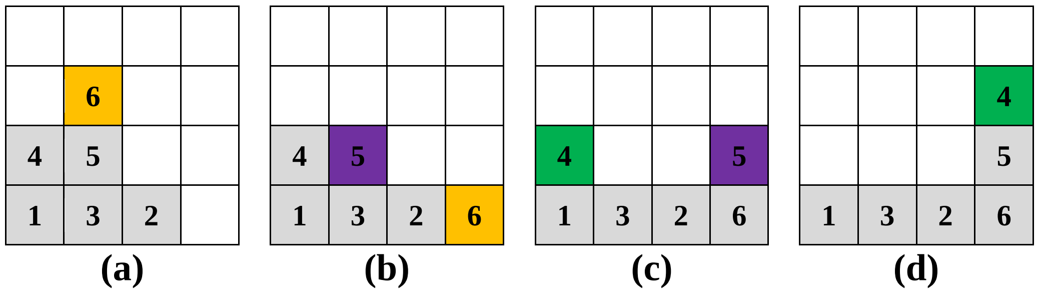


Figure 42: An illustration of the Remarshaling Smart_2 Heuristic Algorithm.

Chapter 5

RESULTS

In this chapter, we will first describe the computational experiments designed to test the performance of the heuristic algorithms presented in Chapter 4. Then we will present and discuss the results of these computational experiments. For our computational experiments, each primary heuristic algorithm is arranged in a way such that several versions of the algorithms are formed. The formulations of these versions are defined in Section 5.1. Later, all of these versions were run for each scenario, which are previously defined in detail in Chapter 3. For these runs, several initial configurations are generated, based on some attributes, which are explained during Data Generation in Section 5.2.

For each scenario, first of all versions of each primary heuristic algorithm are compared with each other. After finding the best performing version of each primary heuristic algorithm, their performances under each scenario are examined. Later, the Smart Heuristic Algorithms' versions are compared under each scenario, and then a comparison under each scenario is performed between the best performing Smart Heuristic Algorithm version and the best performing primary heuristic algorithm version. Next, the best-performed algorithm of this comparison and the corresponding Tabu Search Algorithm are compared. Finally, the best performing algorithm (primary, Smart or Tabu) and the Aydin Heuristic

Algorithm, which is based on difference heuristic algorithm and stack selection rule, for which the main idea is given by Aydin [34], are compared under each scenario. The detailed comparisons and the discussions of the results are given in Section 5.3.

The following notations are used while defining the algorithms.

H	: High Rank Heuristic Algorithm,
M	: Min Rank Heuristic Algorithm,
H_M	: High Rank Modified Heuristic Algorithm,
M_M	: Min Rank Modified Heuristic Algorithm,
R	: Random Heuristic Algorithm,
Hy	: Hybrid Heuristic Algorithm
S	: Smart Heuristic Algorithm,
T	: Tabu Search Algorithm
A	: Aydin Heuristic Algorithm
Re_H	: Remarshaling High Heuristic Algorithm,
Re_M	: Remarshaling Min Heuristic Algorithm,
Re_Se	: Remarshaling Several Heuristic Algorithm,
Re_Sm	: Remarshaling Smart Heuristic Algorithm,
Re_Sm2	: Remarshaling Smart_2 Heuristic Algorithm,

5.1 Versions of Heuristic Algorithms

Each of the primary allocation-retrieval relocation heuristic algorithms is modified by including each remarshaling heuristic algorithms. All five types of remarshaling heuristic algorithms are used either before any containers arrive recently, which is the allocation operation, or before any container is claimed, which is the retrieval operation. Moreover, a version is given in which there is not any remarshaling heuristic algorithm. In total, eleven different versions are proposed for each heuristic algorithm. The following list clarifies the version numbers:

(Algorithm name) – 0	: No remarshaling heuristic algorithm,
(Algorithm name) – 1	: Re_H is used before allocation operation,
(Algorithm name) – 2	: Re_H is used before retrieval operation,
(Algorithm name) – 3	: Re_M is used before allocation operation,
(Algorithm name) – 4	: Re_M is used before retrieval operation,
(Algorithm name) – 5	: Re_Se is used before allocation operation,
(Algorithm name) – 6	: Re_Se is used before retrieval operation,
(Algorithm name) – 7	: Re_Sm is used before allocation operation,
(Algorithm name) – 8	: Re_Sm is used before retrieval operation,
(Algorithm name) – 9	: Re_Sm2 is used before allocation operation,
(Algorithm name) – 10	: Re_Sm2 is used before retrieval operation.

5.2 Data Generation

In the computational experiments, the data set generated by Aydin [34] forms the basis of our data. We added to this data set the arrival rates for the containers arriving recently and removal rates for the claimed containers. Moreover, an additional level for the initial layout density was used to reflect better the characteristics of our problem. Therefore the data sets for our computational experiments have the following factors and the levels:

Stored containers: balanced or unbalanced (containers in stacks are near in number or not),

Number of rows in a stack: 4, 5, 6, or 7,

Number of stacks in a bay: 3, 4, 5, 6 or 7,

Initial layout density: 55%, 60%, 65%, 70%, 75% and 80%,

Arrival Rate: 30%, 50%, and 70% (depending on the available slots in the block),

Removal Rate: 30%, 50%, and 70% (depending on total number of containers, which are stored in the block),

A total of 40 instances were generated for each combination. Hence a total of 86,400 instances were run by using previously defined 11 different heuristic algorithm versions for each scenario and their objective function value is calculated. Then, the same instances were also run for the Smart Heuristic Algorithm, the Tabu Search Algorithm and the Aydin Heuristic Algorithm for each scenario and the results were compared.

5.3 Comparison of Results

Objective function of the first two scenarios, which have no extra stack to help possible rehandlings, include two main variables. These variables form the bases of the changes in the objective function values of the heuristic algorithm versions. The first variable is the number of rehandlings and the second one is the number of containers failed to be retrieved during pick up operations. On the other hand, for the remaining two scenarios, all of the claimed containers are successfully retrieved in each case and cost for the extra stack is fixed, therefore only the number of rehandlings occurred in a given period affects the objective function value of the algorithm version.

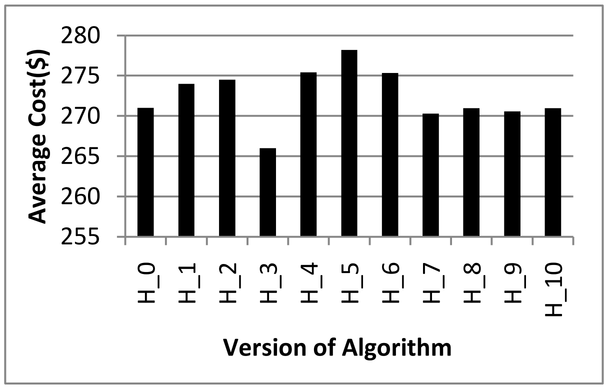
In order to make comparisons between versions of the heuristic algorithms, the average of the objective function values for each instances run for the heuristic algorithm versions are evaluated. This objective function values are accepted as the objective function value for that algorithm version in each scenario and compared.

5.3.1 Comparison of Heuristic Algorithms

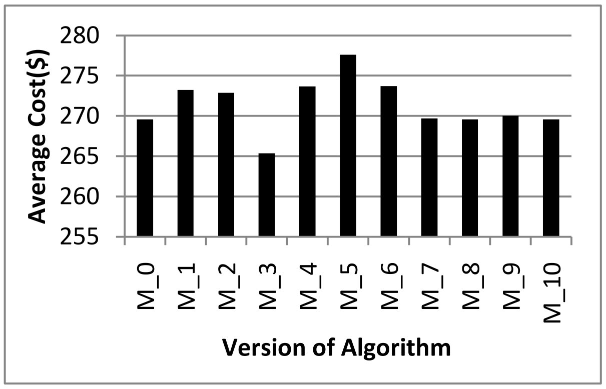
In this section, we present the results of our computational experiments designed for the heuristic algorithms by plotting the objective function value of each algorithm version in a graph. We first give the results of our comparisons for different versions of each heuristic algorithm to identify the best performing one for that heuristic algorithm under each scenario. Then we present the comparison of the best performing heuristic algorithms.

SCENARIO 1

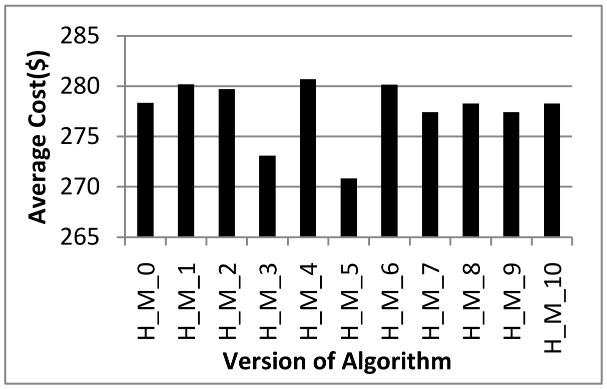
High Rank Heuristic Algorithm



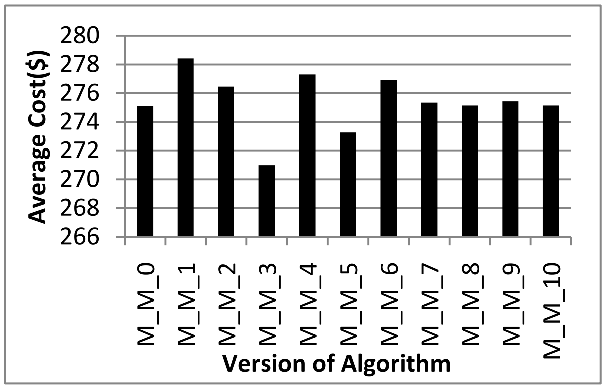
Min Rank Heuristic Algorithm



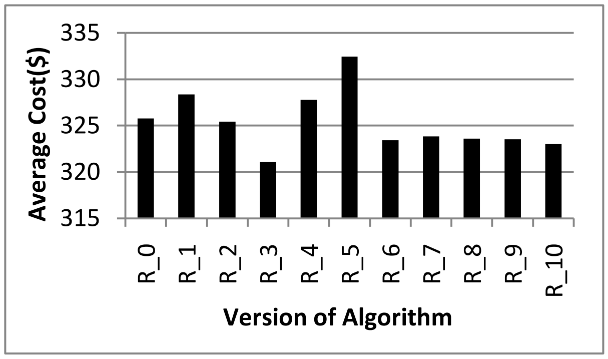
High Rank Modified Heuristic Algorithm



Min Rank Modified Heuristic Algorithm



Random Heuristic Algorithm



Hybrid Heuristic Algorithm

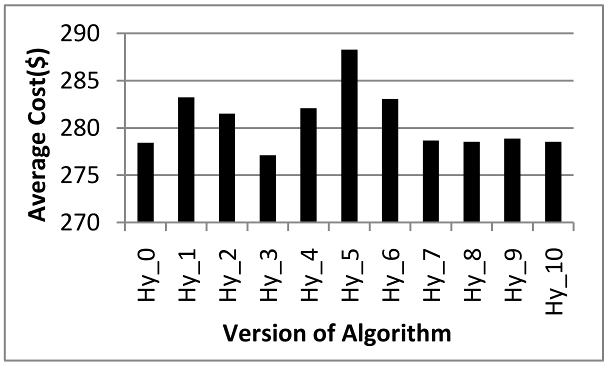


Figure 43: Detailed results for algorithm versions of six initial heuristic.

Figure 43 presents the graphs of the average objective function values of primary allocation-retrieval relocation heuristic algorithms for Scenario 1. It is clearly seen that version 3 dominates the other versions in all six heuristics except the High Rank Modified Heuristic Algorithm, which results better in version 5. The main reason for having better objective functions when used remarshaling is the lowered number of failed containers, since the cost of these failed containers is relatively higher than the remarshaling operation.

Figure 44 depicts the results for Scenario 2. By the graphs it is seen that overall objective functions are significantly lower than Scenario 1. The main reason for that is the decrease in the number of containers failed to be picked-up because of group-based retrieval is allowed, which indicates that the remarshaling operations have lost their effect on the objective function. In this scenario, Min Rank, Min Rank Modified and Hybrid Heuristic Algorithms' initial versions (versions 0), which have no remarshaling operation, resulted in the lowest costs within their versions. On the other hand, High Rank, High Rank Modified and Random Heuristic Algorithms attained the minimum objective function value when the remarshaling operation is used because of the allocation and retrieval rules of the heuristic algorithms. The best performing versions are H_10, H_M_9 and R_10 respectively for these heuristic algorithms, since the efficiency of the layout in a bay is improved by adding smart remarshaling operations within the heuristic algorithms' basic rules.

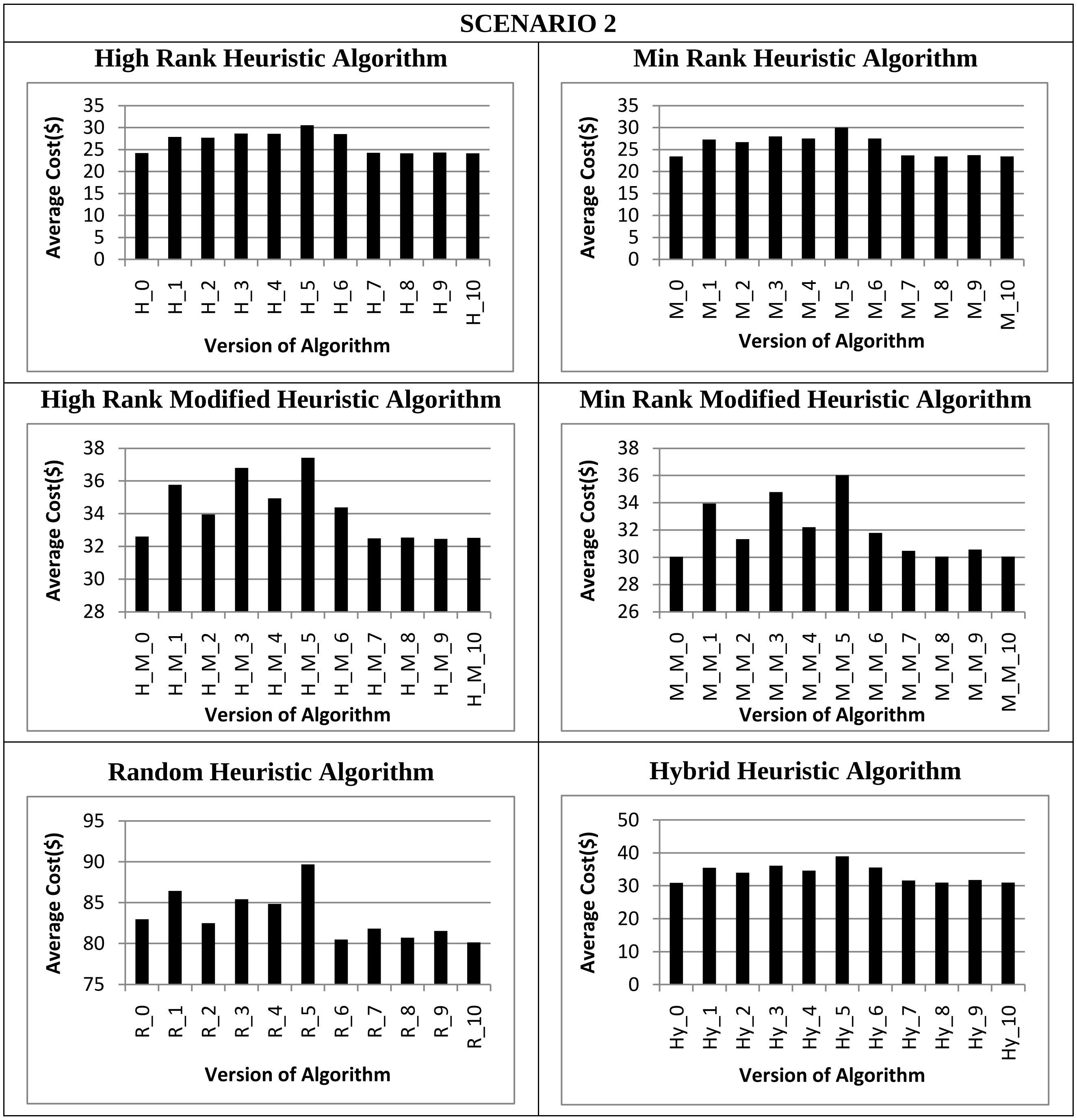


Figure 44: Results of Scenario 2 for primary allocation-retrieval relocation heuristic algorithms.

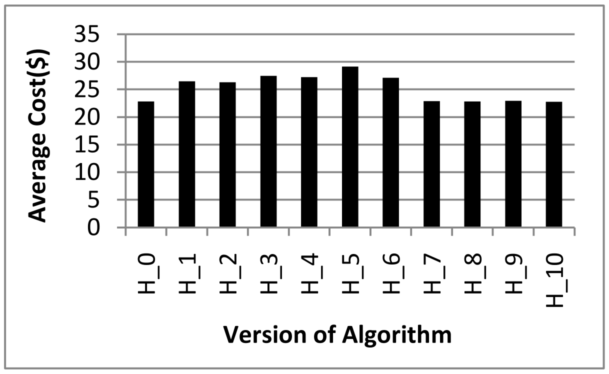
In Figures 45 and 46 results for Scenario 3 and Scenario 4 are given, respectively. In the graphs fix cost of adding an extra stack is not reflected since with the use of these extra stacks, it is ensured that each container is successfully picked up. Hence the penalty cost in the objective function does not have any effect, and only the number of rehandlings in each algorithm version has an influence on the objective function. As a result, the general objective function values are lower than that of Scenario 1 and Scenario 2 only in terms of the cost of rehandlings.

The results for the versions of primary allocation-retrieval relocation heuristic algorithms under Scenario 3 are given in detail in Figure 45. The trend in the results is same as Scenario 2 for each algorithm. Only the results for the Random Heuristic Algorithm show some differences, which is based on the random position selection rule of the heuristic algorithm.

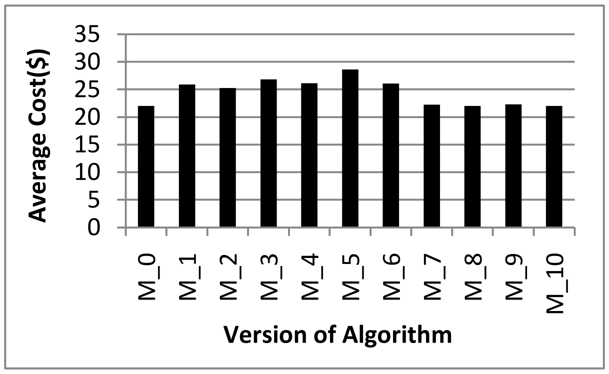
In Scenario 3, for the High Rank and the High Rank Modified Heuristic Algorithms, like in Scenario 2, remarshaling operations improve the objective function by lowering the number of rehandlings. Moreover, while in Scenario 2, the Random Heuristic Algorithm gives a better objective function value in version 10, in Scenario 3 version 6 dominates other Random Heuristic Algorithm versions. However, still each Random Heuristic Algorithm has relatively higher objective function value compared to any other algorithm versions. For the remaining heuristic algorithms, we observe that the initial version of the algorithms outperforms the other versions.

SCENARIO 3

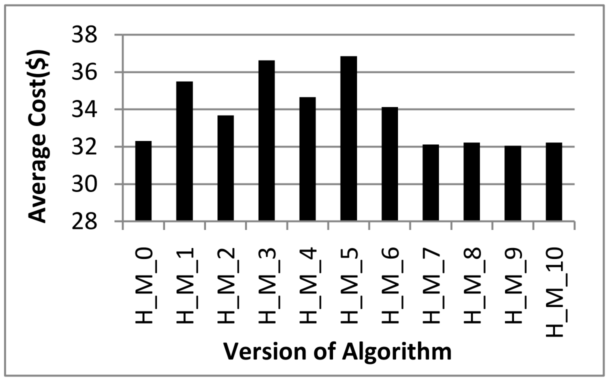
High Rank Heuristic Algorithm



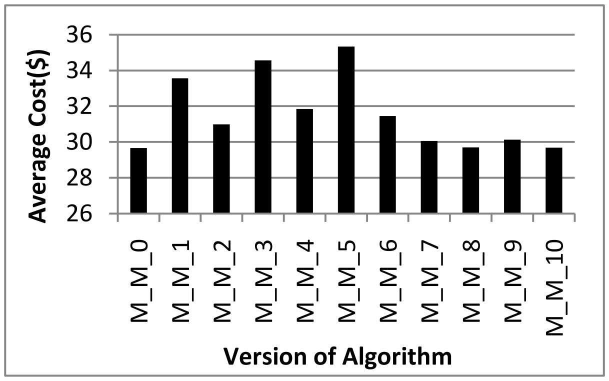
Min Rank Heuristic Algorithm



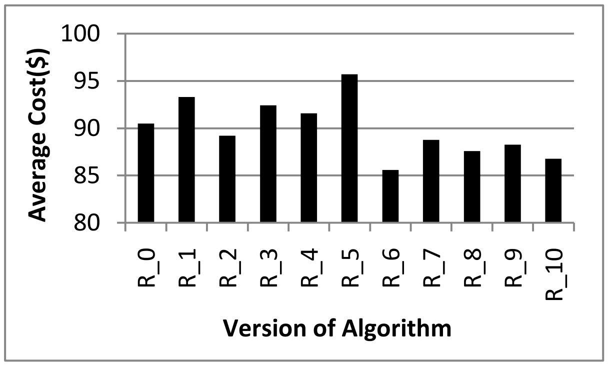
High Rank Modified Heuristic Algorithm



Min Rank Modified Heuristic Algorithm



Random Heuristic Algorithm



Hybrid Heuristic Algorithm

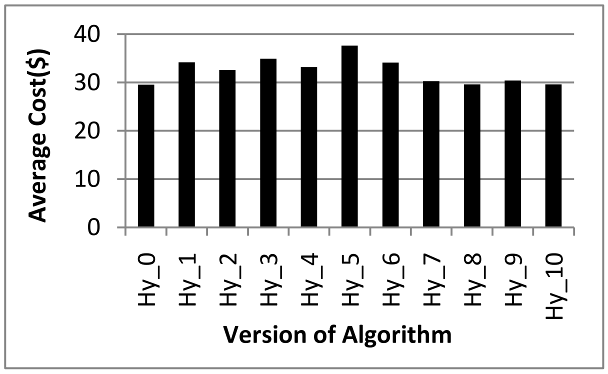
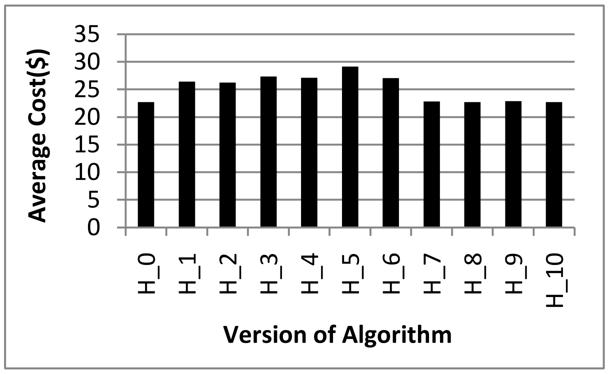


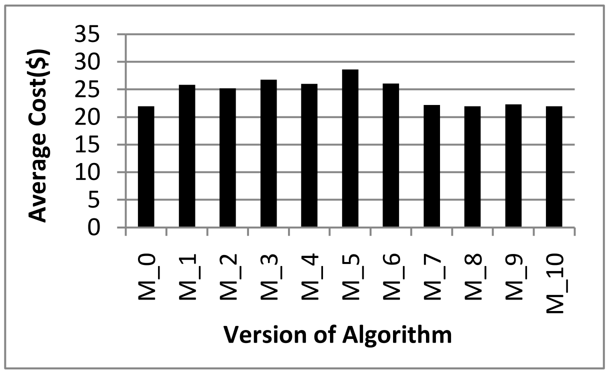
Figure 45: Scenario 3’s results for the primary allocation-retrieval relocation heuristic algorithm versions.

SCENARIO 4

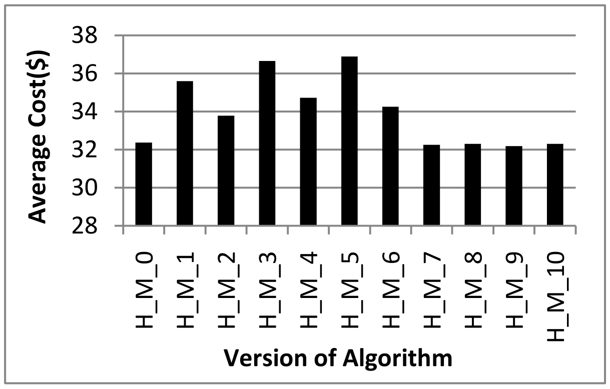
High Rank Heuristic Algorithm



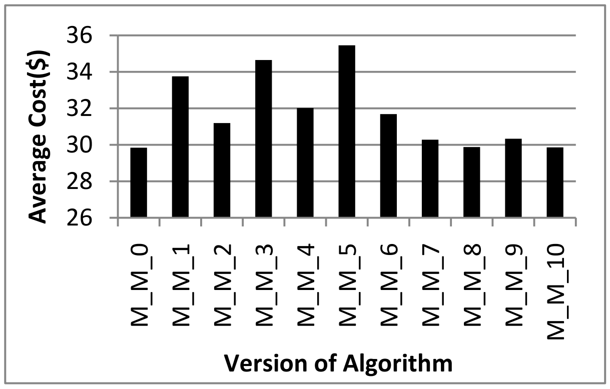
Min Rank Heuristic Algorithm



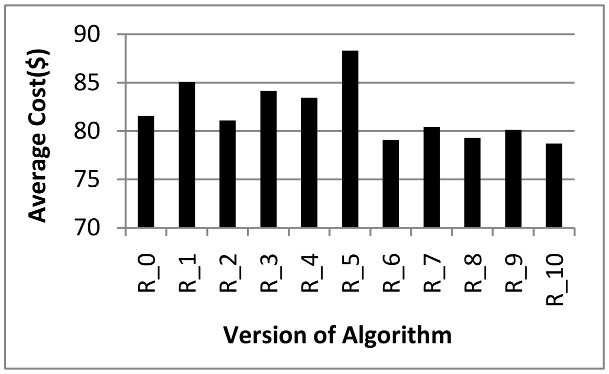
High Rank Modified Heuristic Algorithm



Min Rank Modified Heuristic Algorithm



Random Heuristic Algorithm



Hybrid Heuristic Algorithm

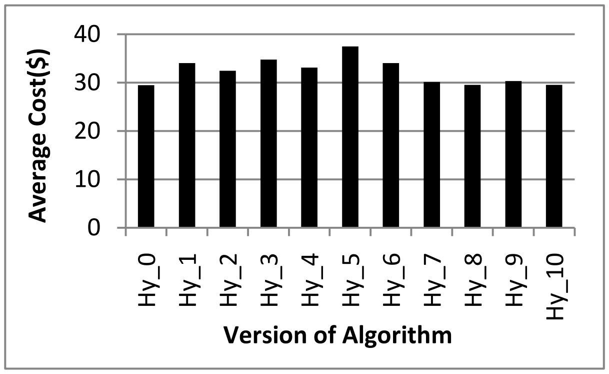


Figure 46: Primary allocation-retrieval relocation heuristic algorithm versions' results for Scenario 4.

Finally, Figure 46 provides the objective function values calculated for each version of the primary allocation-retrieval relocation heuristic algorithms under Scenario 4. The results follow the same behavior with Scenario 2, where the Min Rank, the Min Rank Modified and Hybrid Heuristic Algorithms result in lower objective function values in the initial version than the other versions. On the other hand, the High Rank, the High Rank Modified and Random Heuristic Algorithms results in lower costs when remarshaling operation is included, which is depending on the decreased number of rehandlings while using the remarshaling operation.

Moreover, average costs calculated by the objective function based only on the number of rehandlings are lower in Scenario 2 and Scenario 4 than Scenario 1 and 3. The main reason for this domination in the objective function values is two-fold: the number of containers, which are successfully rehandled, is increased and the total rehandling number is lower in the group based scenarios than the sequence based scenarios.

5.3.2 Comparison of Best Performing Algorithms

In Figure 47, the best performing algorithm versions under each scenario are compared. This comparison is performed in order to select the best performing primary allocation-retrieval relocation heuristic algorithm. So that this primary allocation-retrieval relocation heuristic algorithm can be enhanced to obtain the improved allocation-retrieval relocation heuristic algorithm that is the Smart Heuristic Algorithm.

In each scenario, the graphs show clearly that the Random Heuristic Algorithm gives the worst objective function values. On the other hand, the objective function values of the High Rank Heuristic Algorithm and the Min Rank Heuristic Algorithm are almost equal.

However, the Min Rank Heuristic Algorithm results in a better objective function value because its allocation and relocation decision rule results in better layouts for the following operations within the period. For Scenario 1 and Scenario 2, the Min Rank Heuristic Algorithm has lower rehandling numbers and the number of containers failed to be retrieved compared to the High Rank Heuristic Algorithm. On the other hand, in Scenario 3 and Scenario 4, each algorithm is able to remove each container when they are claimed, which means that there is no unsuccessful pick-up operation. Therefore, the only difference between the Min Rank Heuristic Algorithm and the High Rank Heuristic Algorithm is the rehandling number, which is lower in the Min Rank Heuristic Algorithm.

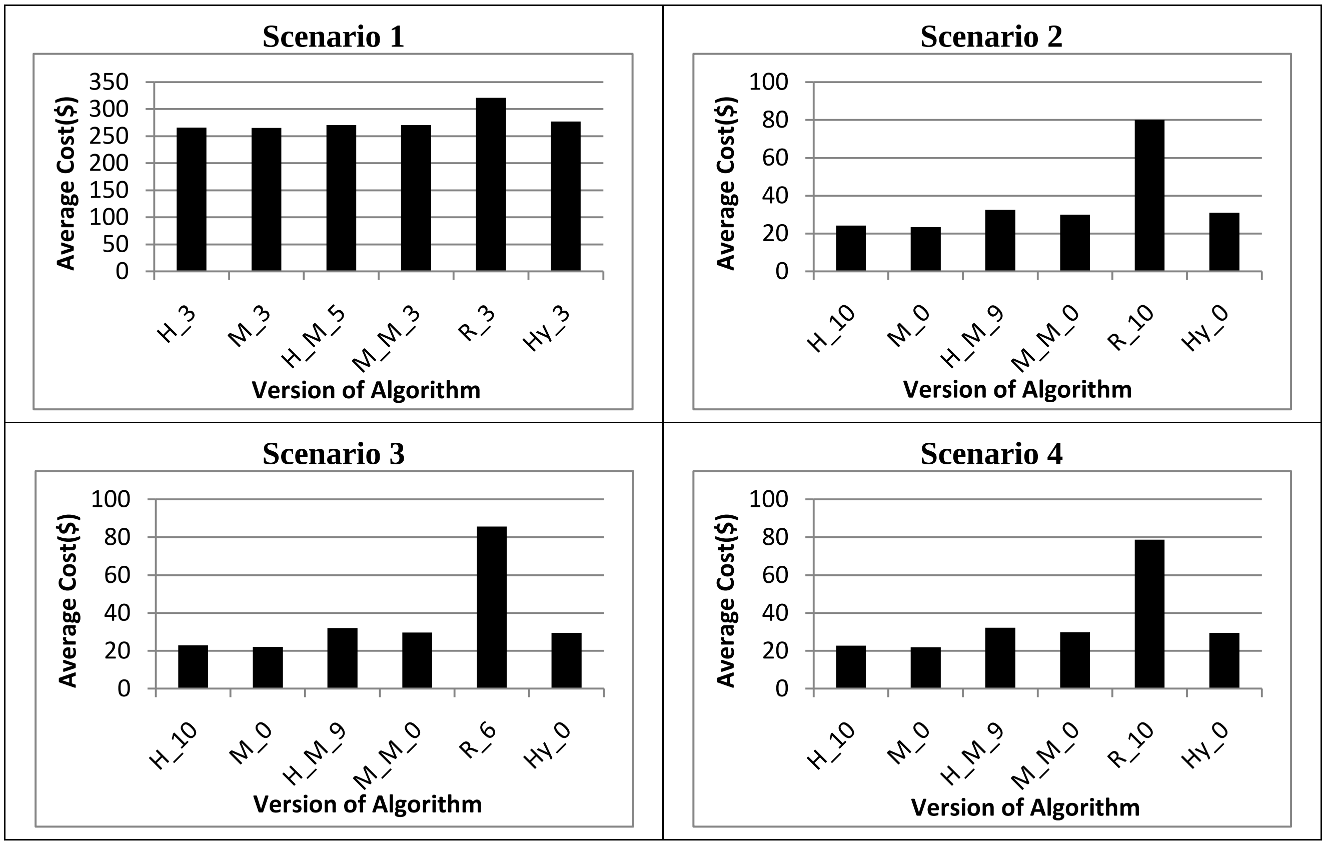


Figure 47: Comparison of each best performing algorithm version for each scenario.

5.3.3 Comparison of Improved Allocation-Retrieval Relocation Heuristic

The Smart Heuristic Algorithm is also run with all five remarshaling rules, which is resulted in again 11 versions. For each scenario, the performance of algorithm versions is compared in Figure 48. According to this figure, it is seen that remarshaling movements helped to improve the objective function values only for Scenario 1. On the other hand, Scenarios 2, 3, and 4 outperforms when no relocation operation is considered or only smart relocations are included, since the number of containers that are failed to be retrieved are either zero or less than that of Scenario 1. These results are parallel to the results of the Min Rank Heuristic Algorithm.

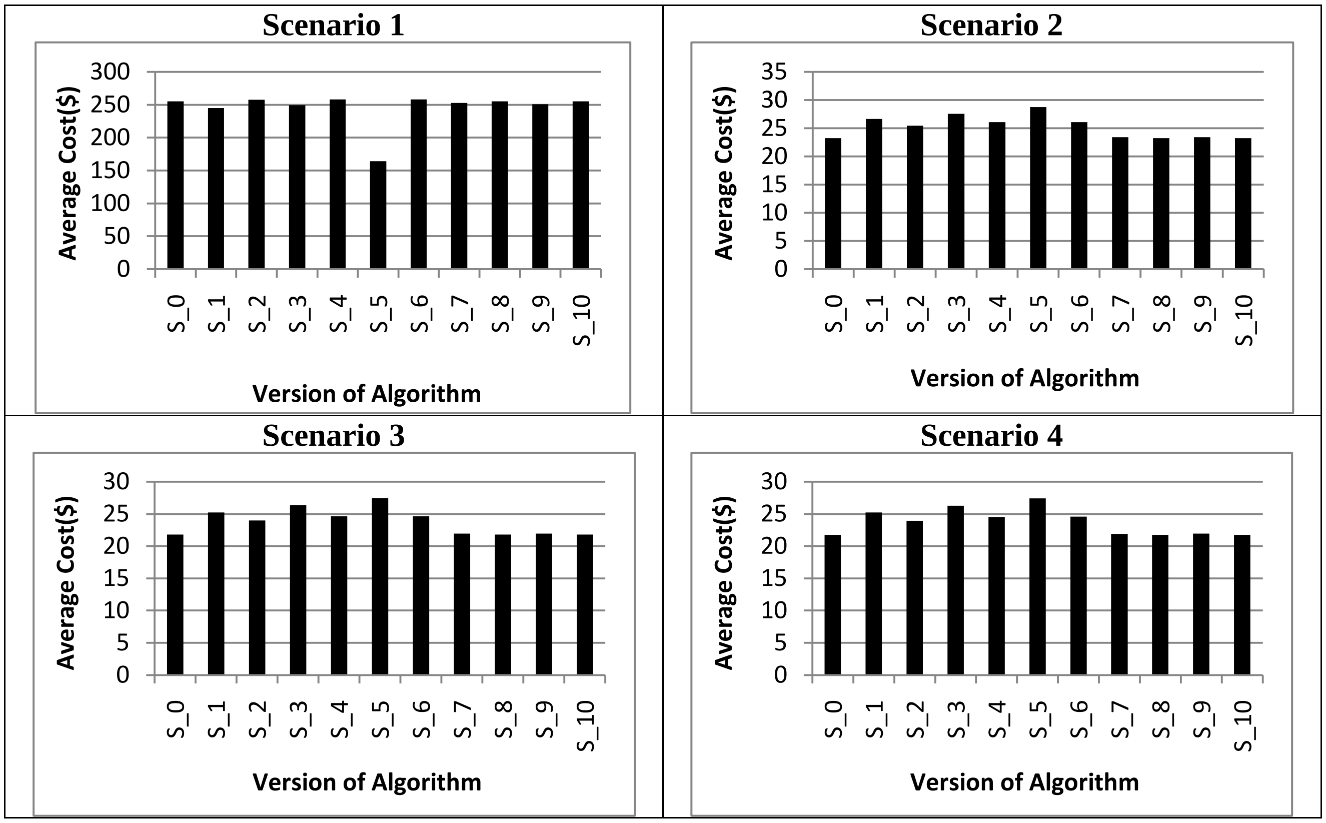


Figure 48: The Smart Heuristic Algorithm versions under each scenario.

Since the Min Rank Heuristic Algorithm is the outperforming algorithm within the primary allocation-retrieval relocation heuristic algorithms and the Smart Heuristic Algorithm is an upgrade of the Min Rank Heuristic Algorithm, the comparison of their results for each scenario is given in Figure 49.

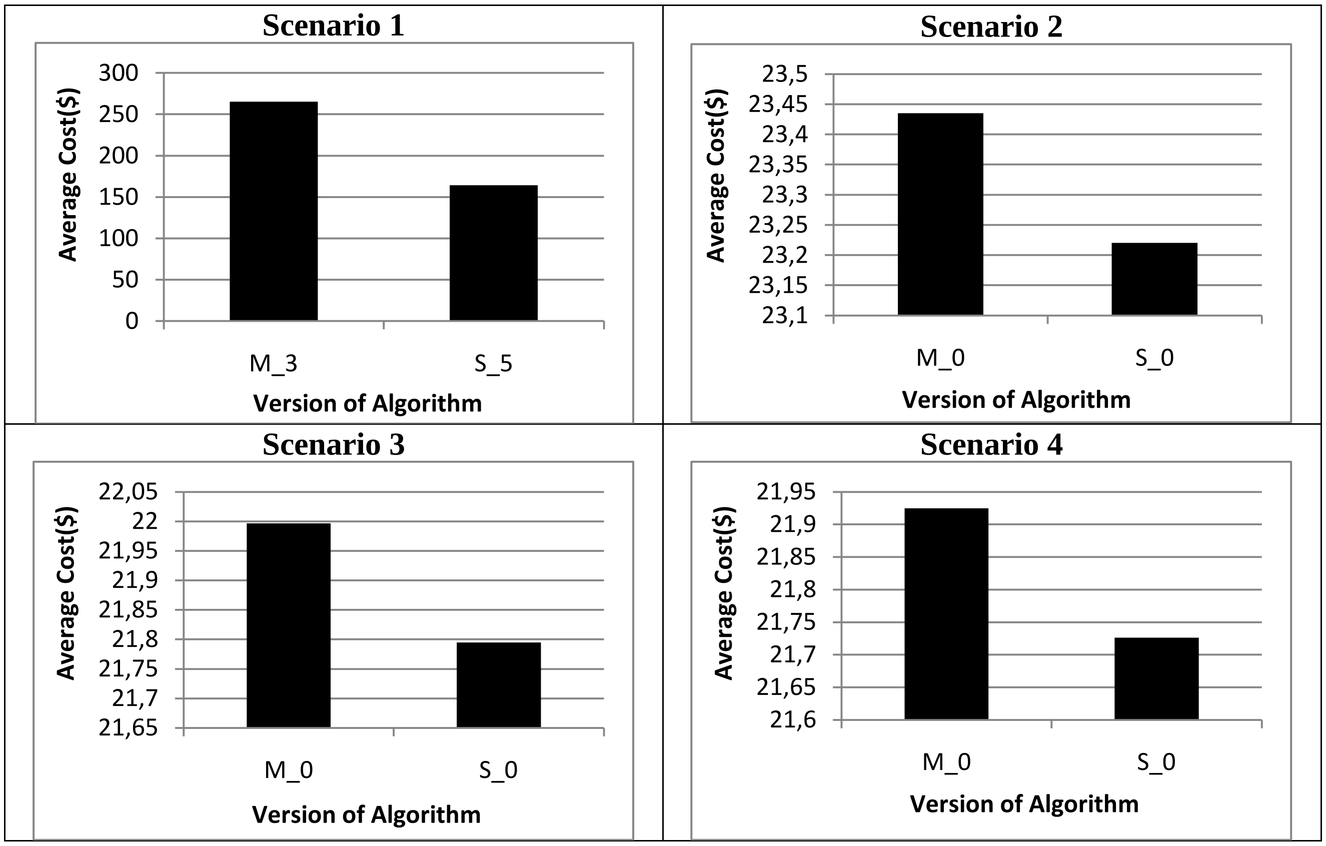


Figure 49: The best performing Min Rank and Smart Heuristic Algorithm are compared for each scenario.

For each scenario, the improvement movements, which are achieved by adding additional relocation operations by the Smart Heuristic Algorithm, show their effect in the objective function. The main reason is the increased efficiency of the layout caused by smart relocations applied at each level during the allocation operation in the Smart Heuristic Algorithm. Therefore, in Scenario 1 and Scenario 2 both the rehandling numbers and the number of unsuccessful pick up trials are decreased, which results in having better objective function values. On the other hand, for Scenarios 3 and 4, despite the number of containers failed to be retrieved remaining zero, for each heuristic algorithm the number of rehandlings is decreased because of the increased efficiency of the layout.

After demonstrating that the Min Rank Heuristic Algorithm is improved by adding the smart relocation movements as in the Smart Heuristic Algorithm, Figure 50 displays that the Smart Heuristic Algorithm is also improved by modifying the initial configuration which is done in the Tabu Search Algorithm. The average cost, which is calculated with the objective function value both for the Smart Heuristic Algorithm and the Tabu Search Algorithm are graphed under each scenario.

For each scenario, the best-resulted versions of the Smart Heuristic Algorithm are formulated with previously defined tabu rules (tabu tenure and aspiration criterion) in the Tabu Search Algorithm. The results show that there exist still some improvements in the configurations so the objective function values are successfully lowered in the Tabu Search Algorithm. A significant upgrade in the objective function value is seen in Scenario 1, which is achieved by lowering the number of rehandlings but their effect becomes negligible while the number of containers failed to be retrieved is lowered considerably. For other scenarios, the number of containers failed to be picked up does not change but the Tabu Search Algorithm effectively decreases the number of rehandlings occurs in these scenarios.

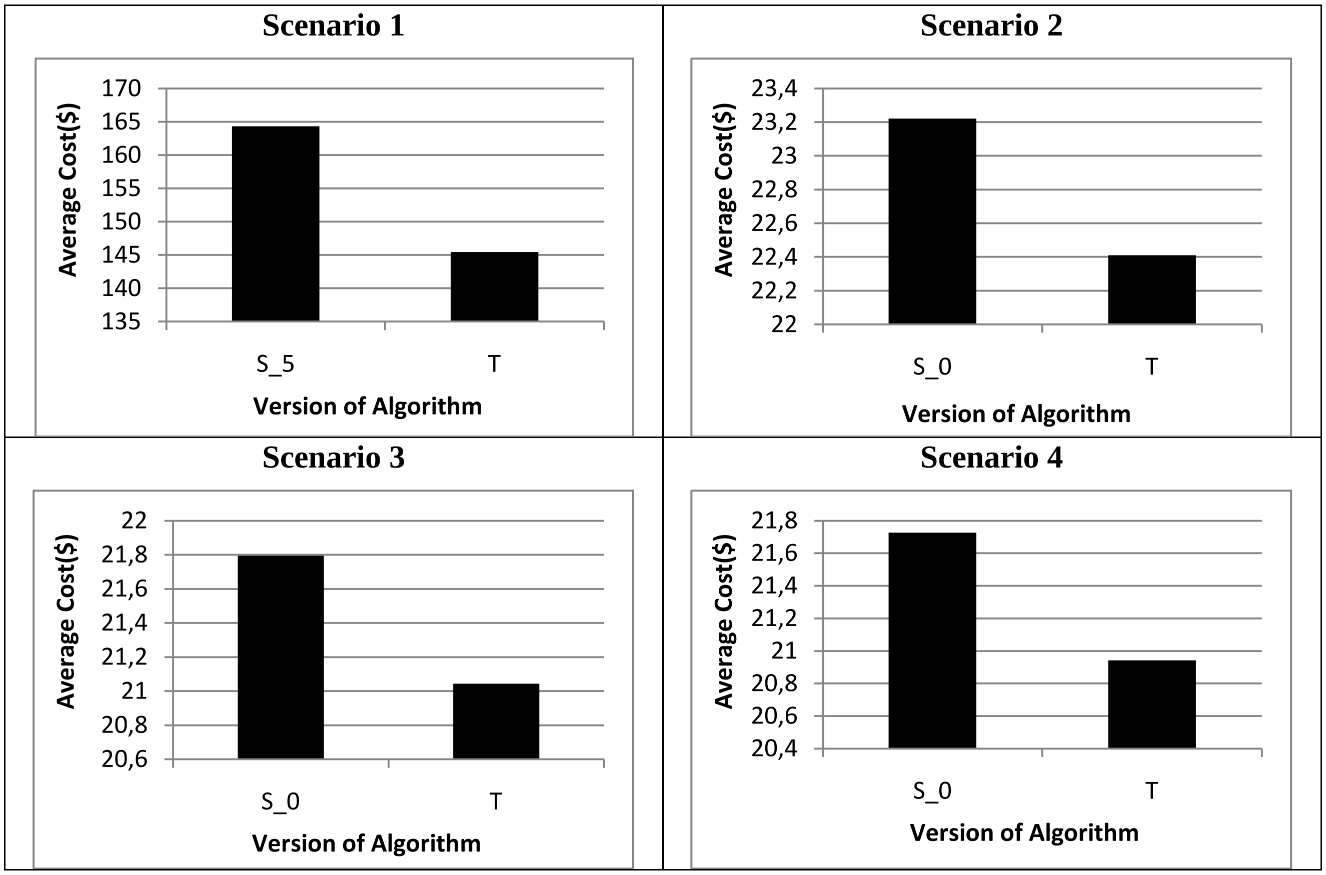


Figure 50: Improvements on the Smart Heuristic Algorithm by the Tabu Search Algorithm.

Finally, the Tabu Search Algorithm is compared with the results of the Aydin Heuristic Algorithm. We modified the difference rules defined in difference heuristic by Aydin [34] by including the allocation operations to obtain the Aydin Heuristic Algorithm. This change is successfully implemented into the algorithm by making the stack selection for the containers arrived recently by using the same difference rules defined in difference heuristic by Aydin [34]. Meanwhile, the retrieval operation is remained exactly the same in difference heuristic by Aydin [34]. Similarly, each version of this algorithm includes the same remarshaling operations with the other heuristic algorithms that we defined.

Figure 51 presents the objective function values of all versions of the Aydin Heuristic Algorithm under each scenario. Similar to other algorithms, due to the penalty cost of unhandled containers, the overall objective function value is high only in Scenario 1. Also only in this scenario, remarshaling operations improves the objective function value.

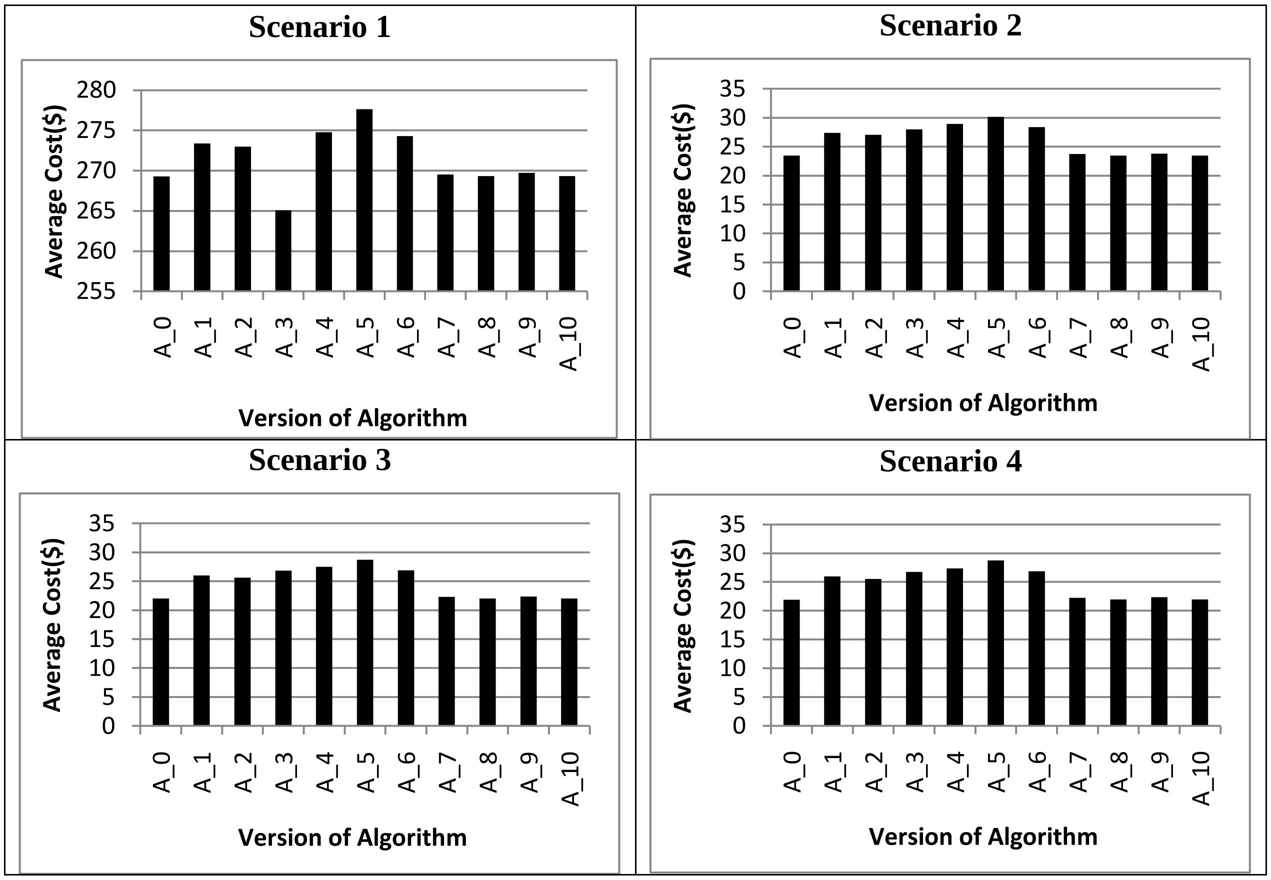


Figure 51: The Aydin Heuristic Algorithm versions, under each scenario.

However, it is obviously seen in Figure 52 that the Smart Heuristic Algorithm outperforms the Aydin Heuristic Algorithm in terms of the objective function under each scenario.

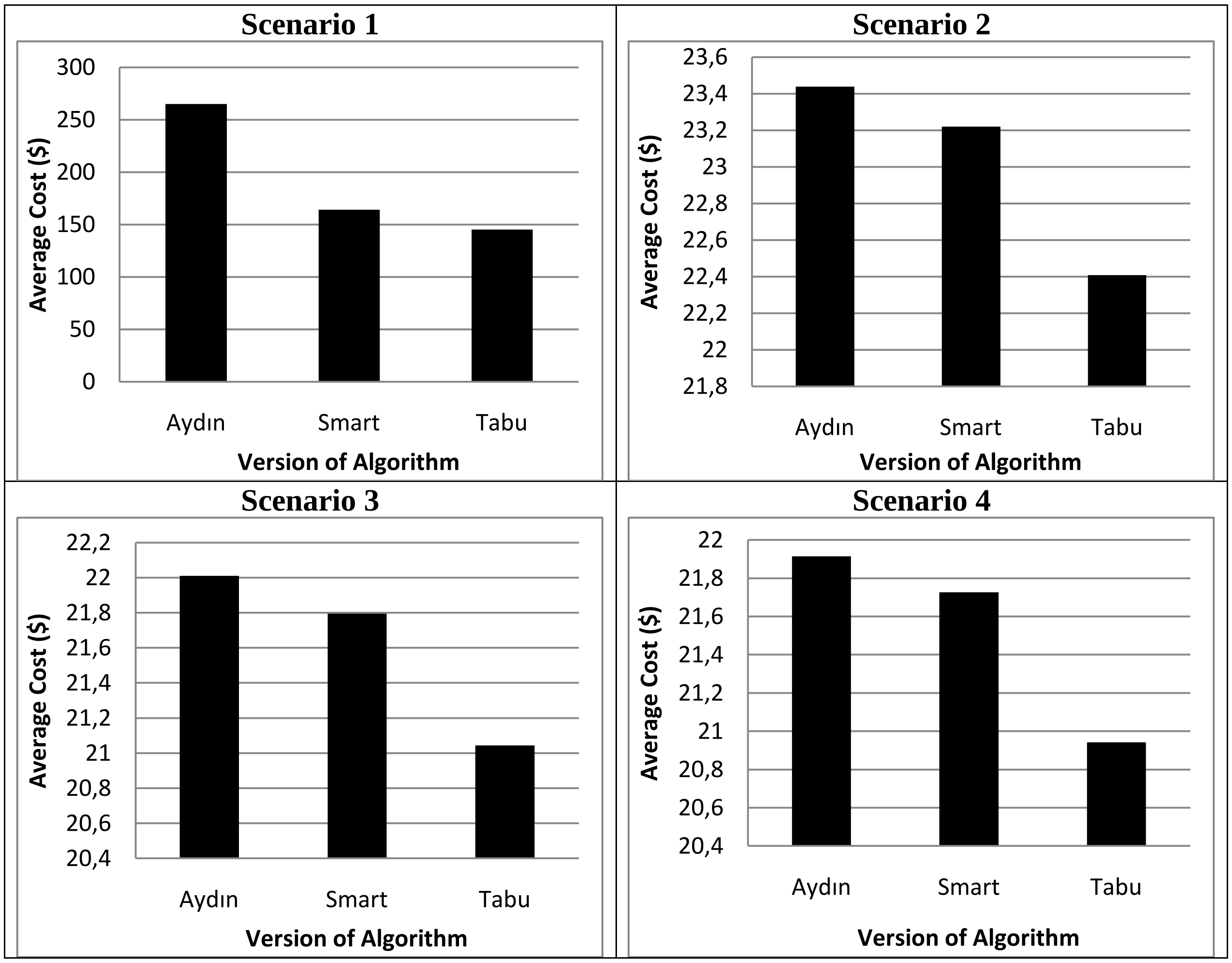


Figure 52: Comparison of the best performing results of the Aydin Heuristic Algorithm and the Tabu Search Algorithms.

Moreover, additional runs are performed by using some of the layouts given in [35]. These selected layouts are the ones which are suitable for our setting. Since these layouts store more containers than the layouts given by [34], depending on the initially defined arrival and removal rates, more containers are rehandled in each period. For each layout category, layouts with one bay and two bays are considered during these runs under each version of Smart and Aydın Heuristic Algorithm. Different than data sets of [34], since in each data set of [37], empty slot number is higher than the height of a stack, scenarios loss their effects in the objective functions. Therefore, the results of each algorithm version are identical to each other under each scenario.

Figure 53 gives the average cost of each algorithm version of these data sets for any scenario. In “random layouts”, the Smart Heuristic Algorithm performs better than the Aydın Heuristic Algorithm in each algorithm version. On the other hand, in the “upside-down layouts”, in half of the versions, Aydın Heuristic Algorithm performs better than the Smart Heuristic Algorithm, which is observed by the decreased objective function costs between 0.4% and 2.3%. For the other versions, the Smart Heuristic Algorithm achieves 0.1 – 14.9% better objective functions than Aydın Heuristic Algorithm.

As a result, we can conclude that the primary allocation-retrieval relocation heuristic algorithms, without any remarshaling operation, might work for any defined scenario. However, in some cases, we were able to improve them by adding remarshaling. Among these heuristics, we have seen that the Min Rank Heuristic Algorithm dominates, which is later improved by adding additional relocation movements during the allocation operations, given as the Smart Heuristic Algorithm. After we applied the Tabu Search Algorithm based on the best performing Smart Heuristic Algorithm version for each scenario, the results of these algorithm versions are improved.

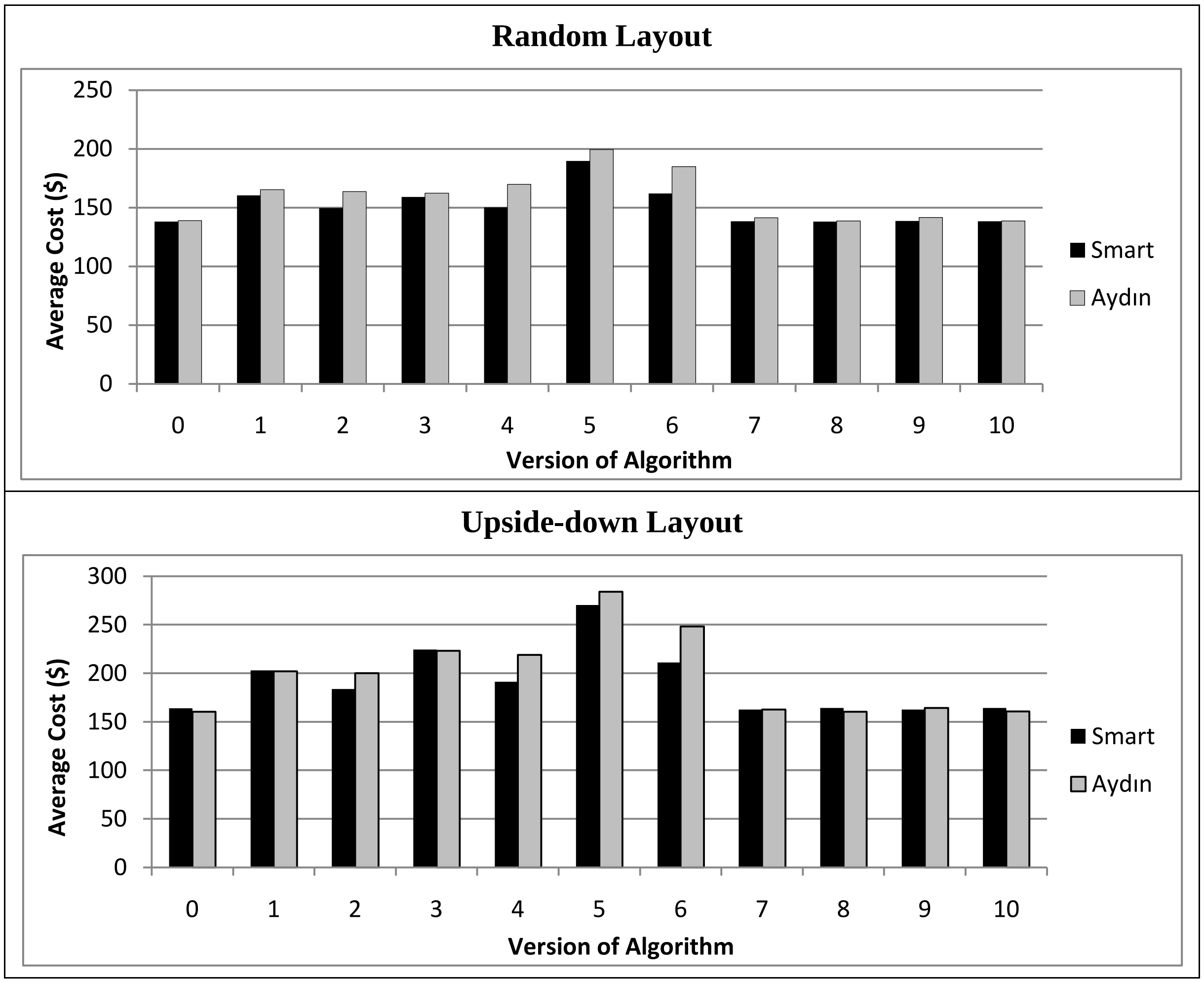


Figure 53: Comparison of Smart Heuristic Algorithm and Aydın Heuristic Algorithm under each version.

Finally, our outperforming heuristic, that is the Tabu Heuristic Algorithm, was compared with the Aydin Heuristic Algorithm, which is based on the difference heuristic [34], and it was seen that the Tabu Heuristic Algorithm is the most successful heuristic algorithm for the defined problem by including all the improvements in its structure.

Besides, depending on the handling capacity of a yard crane, that is stated in [40] and [41], in a container terminal during a workday (24 hours), approximately 12 periods of operations are valid in primary allocation-retrieval relocation heuristic algorithms. The average numbers of periods are improved as 19 and 20 periods in a working day for Aydin Heuristic Algorithm and Smart Heuristic Algorithm, respectively. Finally, in the Tabu Search Algorithm the number of periods is around 22, in a working day. In the view of this number of periods, applying the proposed heuristic algorithms within a container terminal is viable since the algorithms take very little time to run for each time period due to the simple rules they include.

Chapter 6

CONCLUSION AND FUTURE WORK

In this thesis, we studied the storage allocation problem, which is encountered in container terminals. The distinguishing aspects of our study are the consideration of different scenarios for the container terminal setting, and the inclusion of the rehandlings, which occur during the allocation of containers arriving recently, the remarshaling operation and the retrieval of the claimed export containers. Due to these characteristics of the system, the aim of this study is to minimize the cost of the storage allocation operation by reducing the number of rehandlings. Since rehandlings may occur due to any operation within a block, several heuristic algorithms are suggested separately for the operations.

Six basic heuristic algorithms, an improved version of the most successful basic heuristic and a Tabu Search Algorithm were proposed for dealing with the rehandlings which occurs during the allocation and pick-up operations. Moreover, five remarshaling heuristic algorithms were included to be used during the idle time of the yard cranes.

The performance of the heuristic algorithms was tested with respect to the problem characteristics and several scenarios. A total of 86,400 instances were created and tested during the computational experiments. The results provided valuable insights regarding the scenarios created and heuristic algorithms suggested.

When the results are investigated, the importance of locating the *min_ranked* container, on top of stacks can be observed in any scenario. Therefore, the Min Rank Heuristic Algorithm is the dominating heuristic algorithm among the six basic heuristic algorithms.

In sequence based scenario, with no extra stack (Scenario 1), since there are claimed containers which are failed to be retrieved, the Min Rank Heuristic Algorithm using the remarshaling operations before the allocation operations decreased the number of containers that are failed to be retrieved. Different than the literature, focusing on not only the retrieval operation but also the allocation operation has improved the value of the objective function. In other scenarios, since containers are successfully retrieved when they are claimed, the Min Rank Heuristic Algorithm with no remarshaling operation is selected for lower costs in container terminals, since the number of rehandling is decreased.

In each scenario, adding smart relocations before any allocation or retrieval operation, as in the Smart Heuristic Algorithm, results in an efficient block. In this block, *min_ranked* containers are kept as accessible by relocating *containers on top* of stacks during each operation. Even though additional relocations are added, since the overall number of rehandling is decreased, the objective function is improved. Finally, with the Tabu Search Algorithm, additional improvements in the objective functions for each scenario are observed, since efficiency of the block is measured at each allocation or retrieval operation and then the slot selection is performed.

We plan to perform a detailed analysis on the cost elements to measure the sensitivity of the objective function as a future study. Furthermore, better remarshaling rules can be developed and heuristic algorithms can be modified to deal with unknown retrieval.

BIBLIOGRAPHY

[1] World Container Traffic - Drewry Annual Reports (2010).

[2] End Year Fleet Size - Container Leasing Market Analysis (2010).

[3] D. Steenken, S. Voβ, and R. Stahlbock, Container Terminal Operation and Operations Research – A Classification and Literature Review, OR Spectrum, 26 (2004), 3-49.

[4] R. Stahlbock and S. Voβ, Operations Research at Container Terminals: A Literature Update, OR Spectrum, 30 (2008), 1-52.

[5] From Hamburg Port Website, http://www.hafen-hamburg.de/en/content/container-port-throughput-global-comparison.

[6] C. Zhang, J. Liu, Y. Wan, K. G. Murty, and R.J. Linn, Storage Space Allocation in Container Terminals, Transportation Research Part B, 37 (2003), 883-903.

[7] F. A. Vis and R. Koster, Transshipment of Containers at a Container Terminal: An Overview, European Journal of Operational Research, 147(1) (2003), 1-16.

[8] I. Vacca, M. Bierlaire and M. Salani, Optimization at Container Terminals Status , Trends and Perspectives, Proceedings of the Swiss Transport Research Conference (STRC), (2007).

[9] K. G. Murty, J. Liu, Y. Wan, and R. Linn, A Decision Support System for Operations in a Container Terminal, Decision Support Systems, 39 (2005), 309-332.

[10] E. Kozan and P. Preston, Mathematical Modelling of Container Transfers and Storage Locations at Seaport Terminals, OR Spectrum, 28 (2006), 519-537.

[11] M. Bielli, A. Boulmakoul and M. Rida, Object Oriented Model for Container Terminal Distributed Simulation, European Journal of Operational Research, 175 (2006), 1731-1751.

[12] Y. K. H. Lau, L. K. Y. Chan and H. K. Wong, A Virtual Container Terminal Simulator for the Design of Terminal Operation, International Journal on Interactive Design and Manufacturing, 1(2) (2007), 107-113.
[13] E. Nishimura, A. Imai and S. Papadimitriou, Berth Allocation Planning in the Public Berth System by Genetic Algorithms, European Journal of Operational Research, 131 (2001), 282-292.
[14] A. Imai, E. Nishimura and S. Papadimitriou, Berthing Ships at a Multi-user Container Terminal with a Limited Quay Capacity, Transportation Research Part E, 44 (2008), 136-151.
[15] M. K. Bae, Y. M. Park and K. H. Kim, A Dynamic Berth Scheduling Method, International Conference on Intelligent Manufacturing and Logistics, (2007).
[16] S. R. Seyedalizadeh Ganji, A. Babazadeh and N. Arabshahi, Analysis of the Continuous Berth Allocation Problem in Container Ports Using a Genetic Algorithm, Journal of Marine Science and Technology, 15(2) (2010).
[17] D. Ambrosino, A. Sciomachen and E. Tanfani, A Decomposition Heuristics for the Container Ship Stowage Problem, Journal of Heuristics, 12 (2006), 211-233.
[18] A. Sciomachen and E. Tanfani, A 3d-bpp Approach for Optimising Stowage Plans and Terminal Productivity, European Journal of Operational Research, 183 (2007), 1433-1446.
[19] K. H. Kim and Y. M. Park, A Crane Scheduling Method for Port Container Terminals, European Journal of Operational Research, 156(3) (2004), 752-768.
[20] L. Moccia, J. F. Cordeau, M. Gaudioso and G. Laporte, A Branch-and-Cut Algorithm for the Quay Crane Scheduling Problem in a Container Terminal, Naval Research Logistics, 53 (2006), 45-59.
[21] M. Sammarra, J. F. Cordeau, G. Laporte and M. F. Monaco, A Tabu Search Heuristic for the Quay Crane Scheduling Problem, Journal of Scheduling, 10 (2007), 327-336.

[22] C. Bierwirth and F. Meisel, A Survey of Berth Allocation and Quay Crane Scheduling Problems in Container Terminals, European Journal of Operational Research, 202(3) (2009), 615-627.
[23] R. Dekker, P. Voogd and E. Asperen, 2007, Advanced Methods for Container Stacking, In: Container Terminals and Cargo Systems – Design, Operations Management, and Logistics Control Issues, K. H. Kim and H. O. Günther (editors), Springer-Verlag Berlin Heidelberg, (2007), 131-154.
[24] K. H. Kim and H. B. Kim, Segregating Space Allocation Models for Container Inventories in Port Container Terminals, International Journal of Production Economics, 59 (1999), 415-423.
[25] K. H. Kim, Y. M. Park and K. R. Ryu, Deriving Decision Rules to Locate Export Containers in Container Yards, European Journal of Operational Research, 124 (2000), 89-101.
[26] C. Zhang, J. Liu, Y. Wan, K. G. Murty and R. J. Linn, Storage Space Allocation in Container Terminals, Transportation Research Part B: Methodological, 37(10) (2003), 883-903.
[27] J. Kang, K. R. Ryu and K. H. Kim, Deriving Stacking Strategies for Export Containers with Uncertain Weight Information, Journal of Intelligent Manufacturing, 17 (2006a), 399-410.
[28] J. Kang, K. R. Ryu and K. H. Kim, Determination of Storage Locations for Incoming Containers of Uncertain Weight, In: Proc. IEA/AIE, M. Ali and R. Dapoigny (editors), Springer-Verlag Berlin Heidelberg, 4031 (2006b), 1159-1168.
[29] M. Bazzazi, N. Safaei and N. Javadian, A Genetic Algorithm to Solve the Storage Space Allocation Problem in a Container Terminal, Computers & Industrial Engineering, 56 (2009), 44-52.
[30] K. H. Kim, Evaluation of the Number of Rehandles in Container Yards, Computers & Industrial Engineering, 32(4) (1997), 701-711.

[31] K. H. Kim and J. W. Bae, Re-marshaling Export Containers in Port Container Terminals, Computers & Industrial Engineering, 35(3-4) (1998), 655-658.
[32] J. Kang, M. Oh, E. Y. Ahn, K. R. Ryu and K. H. Kim, Planning for Intra-Block Remarshalling in a Container Terminal, In: Proc. IEA/AIE, M. Ali and R. Dapoigny (editors), Springer-Verlag Berlin Heidelberg, (2006), 1211-1220.
[33] K. H. Kim and G. Hong, A Heuristic Rule for Relocating Blocks, Computers & Operations Research, 33 (2006), 940-954.
[34] C. Aydin, Improved Rehandling Strategies for Container Retrieval Process, M. Sc. Thesis, Sabanci University, (2006).
[35] Y. Lee and Y. Lee, A Heuristic for Retrieving Containers from a Yard, Computers & Operations Research, 37 (2010), 1139-1147.
[36] T. Unluyurt and C. Aydın, Improved Rehandling Strategies for Container Retrieval Process, Journal of Advanced Transportation, (2012) (in press, DOI: 10.1002/atr.1193).
[37] M. Caserta, S. Voβ and M. Sniedovich, Applying the Corridor Method to a Blocks Relocation Problem, OR Spectrum, 33 (2011), 915-929.
[38] M. Hussein and M. E. H. Petering, Genetic Algorithm-Based Simulation Optimization of Stacking Algorithms for Yard Cranes to Reduce Fuel Consumption at Seaport Container Transshipment Terminals, IEEE world Congress on Computational Intelligence, (2012).
[39] W. C. Huang and C. Y. Chu, A Selection Model for In-terminal Container Handling Systems, Journal of Marine Science and Technology, 12 (2004), 159-170.
[40] M. E. H. Petering and K. G. Murty, Effect of Block Length and Yard Crane Deployment Systems on Overall Performance at a Seaport Container Transshipment Terminal, Transportation Research Part E, 36 (2009), 1711-1725.
[41] M. E. H. Petering, Effect of Block Width and Storage Yard Layout on Marine Container Terminal Performance, Transportation Research Part E, 45 (2009), 591-610.

APPENDIX

A Feasible_stacks ()

```
Available stack list is null.
if (ava_slot(b, s) > 0)
Stack (b, s) listed in available stacks;
end
```

B Stack_selection_without_rehandle(CONTAINER NAME);

```
Available stack list is divided into two.
if(min_rank of the stack is higher than CONTAINER NAME)
        Stack is listed in Higher stacks
else if (There exists an empty)
        Stack is listed in Empty stacks
end
if(Number of Higher stacks>0)
        Lowest min_ranked stack is selected as selected_stack
else if (Number of Empty stacks>0)
        First stack in the Empty stack list is selected as selected_stack
end
```

C High Rank Heuristic Algorithm – Allocation Operation

```
while (There is a container to be allocated)
        Feasible_stacks ();
        if (There exists an empty stack in the available stack list)
                min_rank(empty stack) = big_M;
        end
        The highest min_rank(b, s) is selected among the available stack list as
selected_stack
        Allocate NEW in the selected stack
end
```

D Min Rank Heuristic Algorithm – Allocation Operation

```
while (There is a container to be allocated)
    Feasible_stacks ();
    Available stack list is divided into two.
    if(min_rank of the stack is higher than NEW)
        Stack is listed in Higher stacks
    else
        Stack is listed in Lower stacks
    end
    if(Number of Higher stacks>0)
        Lowest min_ranked stack is selected as selected_stack
    else if (There exists an empty)
        Empty stack is selected as selected_stack
    else
        Stack that has the highest min_rank is selected as selected_stack
    end
    Allocate NEW in the selected stack
end
```

E High Rank Modified Heuristic Algorithm – Allocation Operation

```
while (There is a container to be allocated)
    Feasible_stacks ();
    if (There exists an empty stack in the available stack list)
        min_rank(empty stack) = big_M;
    end
    The highest min_rank(b, s) is selected among the available stack list as
selected_stack
    if (Container on top in selected stack is lower than NEW)
        An extra slot is searched in the bay
        if (There exists such a slot)
            Relocate container on top into that empty slot temporarily
            Allocate NEW in the selected stack
            Relocate container on top back into selected stack
        else
            Allocate NEW in the selected stack
```

```
                end
        else
                Allocate NEW in the selected stack
        end
end
```

F Min Rank Modified Heuristic Algorithm – Allocation Operation

```
while (There is a container to be allocated)
        Feasible_stacks ();
        Available stack list is divided into two.
        if(min_rank of the stack is higher than NEW)
                Stack is listed in Higher stacks
        else
                Stack is listed in Lower stacks
        end
        if(Number of Higher stacks>0)
                Lowest min_ranked stack is selected as selected_stack
        else if (There exists an empty)
                Empty stack is selected as selected_stack
        else
                Highest min_ranked stackis selected as selected_stack
        end
        if (Container on top in selected stack is lower than NEW)
                An extra slot is searched in the bay
                if (There exists such a slot)
                        Relocate container on top into that empty slot temporarily
                        Allocate NEW in the selected stack
                        Relocate container on top back into selected stack
                else
                        Allocate NEW in the selected stack
                end
        else
                Allocate NEW in the selected stack
        end
end
```

G Random Heuristic Algorithm – Allocation Operation

```
while (There is a container to be allocated)
    Selected stack is chosen randomly
    A row in the selected stack is chosen randomly
    if(selected slot is empty)
        Allocate NEW in the selected stack
    else
        if (ava_slot(b, s)=0)
            Relocate container on top by Min Rank Algorithm
        end
        Relocate containers that are on the selected slot or above temporarily
        Allocate NEW in the selected slot
        Relocate temporarily removed containers in the selected stack
    end
end
```

H Hybrid Heuristic Algorithm – Allocation Operation

```
while (There is a container to be allocated)
    Selected stack is chosen randomly
    A row in the selected stack is chosen randomly
    if(selected slot is empty)
        Allocate NEW in the selected stack
    else
        Relocate containers that are on the selected slot or above by using Min
Rank
        Allocate NEW in the selected slot
    end
end
```

I Smart Heuristic Algorithm – Allocation Operation

```
while(There is a container to be allocated)
    Feasible_stacks ();
    Stack_selection_without_rehandle(New Container);
    if(A stack is selected)
        while (there is possible movements)
            for (Each stack in the bay)
                if (container on top (b, s) is not listed in Trial List)
                    if (Container on top(b, s) > min_rank(b, s))
                        Container on top(b, s) is listed in the
            Remarshaling list
                    end
                end
            end
            Containers in the Remarshaling list are sorted in descending
            order
            First container in the list is listed in Trial List
            if(First container's rank >min_rank of the selected stack &&
            First container's rank <NEW's rank&& Empty slots>1)
                Allocate First container in the selected stack
            end
        end
        Allocated NEW in selected stack
    else
        Stacks are sorted in ascending order based on their ava_slots , and
listed in Sorted List
        if(There are more than one stack with same empty slots)
            These stacks are sorted in descending order based on their
min_ranks
        end
        while (There is not a selected stack)
            Next stack in the Sorted List is selected
            if(If a stack is selected)
                if(min ranked container is stored at the bottom of the
selected stack || containers that are stored under min_ranked container>NEW)
                    Min_ranked container and containers stored
above it tried to be relocated into other stacks
```

```
                    if(Relocation is done)
                        NEW is located in that stack
                    end
                end
            else
                Min_ranked stacked is selected as Selected Stack
            end
        end
    end
end
```

J High Rank Heuristic Algorithm – Pick-up Operation

```
while (There is a container to be retrieved)
    if (Claimed container is accessible)
        Remove the claimed container
    else
        Containers which are stored over the claimed container are relocated
by High Rank
        Remove the claimed container
    end
end
```

K Min Rank Heuristic Algorithm – Pick-up Operation

```
while (There is a container to be retrieved)
    if (Claimed container is accessible)
        Remove the claimed container
    else
        Containers which are stored over the claimed container are relocated
by Min Rank
        Remove the claimed container
    end
end
```

L High Rank Modified Heuristic Algorithm – Pick-up Operation

```
while (There is a container to be retrieved)
    if (Claimed container is accessible)
        Remove the claimed container
    else
        Containers which are stored over the claimed container are relocated by High Rank Modified
        Remove the claimed container
    end
end
```

M Min Rank Modified Heuristic Algorithm – Pick-up Operation

```
while (There is a container to be retrieved)
    if (Claimed container is accessible)
        Remove the claimed container
    else
        Containers which are stored over the claimed container are relocated by Min Rank Modified
        Remove the claimed container
    end
end
```

N Random Heuristic Algorithm – Pick-up Operation

```
while (There is a container to be retrieved)
    if (Claimed container is accessible)
        Remove the claimed container
    else
        Containers which are stored over the claimed container are relocated by Random
        Remove the claimed container
    end
end
```

O Hybrid Heuristic Algorithm – Pick-up Operation

```
while (There is a container to be retrieved)
    if (Claimed container is accessible)
        Remove the claimed container
    else
        Containers which are stored over the claimed container are relocated by Min Rank Modified
        Remove the claimed container
    end
end
```

P Remarshaling High Heuristic Algorithm

```
Remarshaling High:
for (Each stack in the bay)
    if (Container on top(b, s) > min_rank(b, s))
        Container on top(b, s) is relocated by High Rank Modified Algorithm
    end
end
```

Q Remarshaling Min Heuristic Algorithm

```
for (Each stack in the bay)
    if (Container on top(b, s) > min_rank(b, s))
        Container on top(b, s) is relocated by Min Rank Modified Algorithm
    end
end
```

R Remarshaling Several Heuristic Algorithm

```
while (there is more possible movements)
    for (Each stack in the bay)
        if (Container on top(b, s) > min_rank(b, s))
            Container on top(b, s) is relocated by Min_Modified Algorithm
        end
    end
end
```

S Remarshaling Smart Heuristic Algorithm

```
for (Each stack in the bay)
    if (Container on top(b, s) > min_rank(b, s))
        Container on top(b, s) is listed in the Remarshaling list
    end
end
Containers in the Remarshaling list are sorted in descending order
for( Each container in the Remarshaling List)
    Available stack list is null.
    if (ava_slot(b, s) > 0)
        if((b, s) is different than container on top's (b, s))
            Stack (b, s) listed in available stacks;
        end
    end
    Stack_selection_without_rehandle(Container on top);
    if(A stack is selected)
        Allocate Container on top in the selected stack
    end
end
```

T Remarshaling Smart_2 Heuristic Algorithm

```
while (there is possible movements)
    for (Each stack in the bay)
        if (container on top (b, s) is not listed in Trial List)
            if (Container on top(b, s) > min_rank(b, s))
                Container on top(b, s) is listed in the Remarshaling list
            end
        end
    end
    Containers in the Remarshaling list are sorted in descending order
    First container in the list is listed in Trial List
    Available stack list is null.
    if (ava_slot(b, s) > 0)
        if((b, s) is different than container on top’s (b, s))
            Stack (b, s) listed in available stacks;
        end
    end
    Stack_selection_without_rehandle(Container on top);
    if(A stack is selected)
        Allocate Container on top in the selected stack
    end
end
```