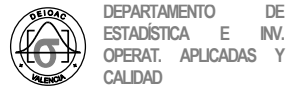
PhD thesis

Heuristic Algorithms for Real Crane Scheduling Problems in Container Yards

Hongtao Wang

Universitat Politècnica de València (UPV)



Heuristic Algorithms for Real Crane Scheduling Problems in Container Yards

Ph.D. thesis

Hongtao Wang

Supervisors:
Prof. Dr. Rubén Ruiz García
Prof. Dr. María Fulgencia Villa Juliá
Prof. Dr. Eva Vallada Regalado

March, 2025

Departamento de Estadística e Investigación Operativa Aplicadas y Calidad
Grupo de Sistemas de Optimización Aplicada
Universitat Politècnica de València
Camino de Vera s/n, 46021, València, Spain

"I believe that one day, we can ride the long wind to break through the waves, hang up the cloud sail high, and march forward bravely to the deep, deep sea."

– Bai Li , *Three Poems on Difficult Journeys* (705-762).
The Chinese writing above and the painting below were approximately created in the 18th century by anonymous authors.

Resumen

El transporte de contenedores es un factor clave en el comercio moderno, representando el segmento de más rápido crecimiento en el transporte marítimo, lo que ejerce una inmensa presión sobre los terminales portuarios, especialmente en las grúas de patio. Esta tesis aborda un problema realista de programación de grúas de patio (YCSP) que incluye asignaciones dinámicas de puntos de entrada/salida (I/O) durante el proceso de optimización. Los puntos I/O actúan como buffers entre los modos de transporte dentro de la terminal, y su disponibilidad afecta la eficiencia en los procesos de almacenamiento y extracción. El problema es altamente realista pero complejo de resolver, lo que limita la eficiencia de carga en el terminal.

Para mejorar la eficiencia de carga, esta tesis primero introduce una heurística simple, denominada Regla de Tiempo de Finalización Simulado (SETR), que secuencia contenedores y asigna puntos I/O. SETR supera las heurísticas basadas en reglas existentes, aunque sigue enfrentando desafíos en instancias más grandes. Además, recurrimos a una serie de metaheurísticas, incluidos los Procedimientos de Greedy Randomized Adaptive Search Procedure (GRASP) y el Iterated Greedy (IG). Se proponen dos enfoques GRASP junto con mejoras en las fases de construcción de soluciones y búsqueda local, específicamente adaptadas a este YCSP. Para una mejora adicional, se diseñan tres métodos IG con una intensificación de nuevos operadores IG de destrucción, reconstrucción y búsqueda local. Todos los métodos GRASP e IG propuestos se calibraron y evaluaron cuidadosamente a través de ex-

perimentos computacionales exhaustivos. Los resultados indican que ambos métodos GRASP pueden resolver de manera efectiva y eficiente instancias grandes con mejoras significativas en comparación con los enfoques más avanzados. Los métodos IG propuestos incluso superan a todos los algoritmos GRASP, donde pequeños cambios en las características del algoritmo tienen un impacto profundo en el rendimiento final.

En resumen, esta tesis se centra en mejorar la eficiencia operativa de las grúas de patio al considerar operaciones de carga realistas durante la programación de grúas, lo que acerca el estudio científico a la práctica en el mundo real. Diseñamos, calibramos y evaluamos heurísticas y metaheurísticas efectivas para optimizar las operaciones de entrega, con un enfoque práctico en las asignaciones dinámicas de los puntos I/O. Esto no solo mejora la eficiencia de los terminales portuarios, sino que también promueve un futuro más verde y sostenible.

Resum

El transport de contenidors és un factor clau en el comerç modern, representant un dels segments de més ràpid creixement en el transport marítim, la qual cosa exerceix una gran pressió sobre els terminals portuaris, especialment en les grues de pati. Aquesta tesi aborda un problema realista de programació de grues de pati (YCSP) que inclou assignacions dinàmiques dels punts d'entrada/ixida (I/O) durant el procés d'optimització. Els punts d'I/O actuen com a búfers entre els modes de transport dins de la terminal, i la seua disponibilitat afecta l'eficiència dels processos d'emmagatzematge i extracció. El problema és molt realista, però complex de resoldre, la qual cosa limita l'eficiència de la càrrega en el terminal.

Per a millorar l'eficiència de la càrrega, aquesta tesi primer introdueix una heurística senzilla, denominada Regla de Temps de Finalització Simulat (SETR), que seqüencia contenidors i assigna punts d'I/O. SETR supera les heurístiques basades en regles existents, encara que s'enfronta a desafiaments en instàncies més grans. A més, es proposa una sèrie de metaheurístiques, incloent-hi Procediments de Greedy Randomized Adaptive Search Procedure (GRASP) i Iterated Greedy (IG). Es proposen dues aproximacions GRASP juntament amb millores en les fases de construcció de solucions i de cerca local, específicament adaptades per a aquest YCSP. Per a una millora addicional, es dissenyen tres mètodes IG amb una intensificació de nous operadors IG de destrucció, reconstrucció i cerca local. Tots els mètodes GRASP i IG proposats són calibrats i avaluats acuradament mitjançant experiments

computacionals exhaustius. Els resultats indiquen que ambdós mètodes GRASP poden resoldre de manera efectiva i eficient instàncies grans, amb millores significatives respecte als enfocaments més avançats. Els mètodes IG proposats superen fins i tot a tots els algoritmes GRASP, on xicotets canvis en les característiques dels algoritmes tenen un impacte profund en el rendiment final.

En resum, aquesta tesi se centra en millorar l'eficiència operativa de les grues de pati tenint en compte operacions de càrrega realistes durant la programació de grues, la qual cosa acosta l'estudi científic a la pràctica del món real. Hem dissenyat, calibrat i avaluat heurístiques i metaheurístiques efectives per a optimitzar les operacions de lliurament, amb un enfocament pràctic en les assignacions dinàmiques dels punts I/O. Això no només millora l'eficiència dels terminals portuaris, sinó que també promou un futur més verd i sostenible.

Abstract

Containerized transport is a major factor in modern trade leading the fastest-growing segment of maritime transport, which exerts immense pressure on port terminals, particularly on yard cranes. This thesis addresses a realistic yard crane scheduling problem (YCSP) that includes dynamic assignments of input/output (I/O) points during the optimization process. I/O points serve as buffers between transportation modes within the terminal, and their availability affects efficiencies in storing and retrieval processes. The problem is highly realistic yet complex to solve, which constricts the loading efficiency in the terminal.

To improve the loading efficiency, the thesis first introduces a simple heuristic, referred to as the Simulated Ending Time Rule (SETR), which sequences containers and assigns I/O points. SETR outperforms existing rule-based heuristics, while it still encounters challenges in larger instances. Furthermore, we turn to a series of metaheuristics including Greedy Randomized Adaptive Search Procedures (GRASP) and Iterated Greedy (IG). Two GRASP approaches are proposed along with improvements to the solution construction and local search phases that are specifically tailored for this YCSP. For further improvement, three IG methods are designed with novel IG operators of destruction, reconstruction, and local search. All the proposed GRASP and IG methods are carefully calibrated and evaluated throughout compre-

hensive computational experiments. The results indicate that both the GRASP and IG methods can effectively and efficiently solve large instances with significant improvements against the state-of-the-art approaches. The proposed IG methods even outperform all GRASP algorithms, where small changes in algorithm features have a profound impact on the final performance.

In summary, this thesis focuses on improving the operational efficiency of the yard crane by considering realistic loading operations during the crane scheduling, which brings the scientific study closer to real-world practice. We design, calibrate, and evaluate effective heuristics and metaheuristics to optimize delivery operations, with a practical focus on dynamic I/O assignments. This not only improves the efficiency of terminal ports but also promotes a greener and more sustainable future.

Acknowledgements

It is commonly believed that the completion of a Ph.D. defense marks the end of the journey as a student. I never thought about how my student life would end and how it would be fantastic to be a real Ph.D. and an independent researcher. The process of writing the thesis itself may be uneventful, but as I organize and review the past materials, every moment recalled, whether ordinary, disheartening, or exhilarating, is worth remembering.

Looking back on this journey, I would like to express my deepest gratitude to my Ph.D. supervisor, Rubén Ruiz García, whose unwavering support, guidance, and encouragement have been instrumental in the completion of this dissertation. Rubén has been a constant source of inspiration, providing insightful advice and valuable feedback at every stage of this journey. I am also profoundly grateful to my co-supervisors, Eva Vallada Regalado and María Fulgencia Villa Julià, for their invaluable input and continuous support. Without their expertise and dedication, this work would not have been possible. Their knowledge and perspective have greatly enriched this research, and I am fortunate to have had the opportunity to work under their guidance.

Special thanks go to professor Federico Perea Rojas-Marcos at Sevilla University for welcoming me during my academic stay in September of 2023. The time spent there was immensely rewarding, offering

new perspectives and insights that have significantly contributed to this dissertation. I especially cherish the new friends there and those Friday lunch gatherings, where we discussed academics, life and even gossip.

I would like to extend my heartfelt thanks to my colleagues and friends who have been by my side throughout these years. Alba, Joan, Sergio, Giulia, Celia, Pedro, Maider, and Mike, your camaraderie, encouragement, and readiness to assist with even the smallest matters in life and work have become some of the most cherished memories during my doctoral journey. I will always remember the time when you sang "Cumpleaños Feliz" to me on my first birthday celebrated in Spain. The same thanks also go to the Chinese friends I met here, Jiajun, Minjie, Yongkun, Tianliang, Yuda and so on. I am so lucky to have found a group of people to play badminton with, organize parties, and even travel to other European countries.

Participation in several conferences during my Ph.D. has been a remarkable experience. I am grateful for the feedback and discussions with experts and peers during the conferences including the "XL Congreso Nacional de Estadística e Investigación Operativa (SEIO 2023)" held from 7th to 10th of November, 2023, in Elche, Spain; "DSA International Summer Conference" from 6th to 7th of June, 2024, in Valencia, Spain; "33rd European Conference on Operational Research (EURO 2024)" from 30th of June to the 3rd of July, 2024. These conferences have broadened my horizons to the academic community, which have helped refine my ideas and provided inspiration for my research.

To my family, I owe an immense debt of gratitude. My grandpa, mom, dad, sister, and girlfriend, your love, patience, and belief in me have been the foundation of my whole life. Thank you for your endless support and for always standing by my side through both moments of triumph and times of struggle. Having you in my life is the greatest

blessing I have ever known.

Lastly, I would like to thank all those who have participated, directly or indirectly, in the completion of this dissertation. All the support, recognition, and even doubts and rejections have become the driving force on my path forward, and I am truly grateful.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AGVs	Automated Guided Vehicles
ANOVA	Analysis Of Variance
AP	Assignment Problem
ARPD	Average Relative Percentage Deviation
ASCs	Automated Stacking Cranes
ATC	Automatic Container Terminal
B&B	Branch and Bound
BAP	Berth Allocation Problem
BCE	Before the Common Era
DOE	Design Of Experiments
DWTs	Dead Weight Tons
E	Equal Instances
GA	Genetic Algorithm
GRASP	Greedy Randomized Adaptive Search Procedure
HSD	Honest Significant Difference
I/O	Input/Output Points
IG	Iterated Greedy
LSM	Local Search Methods
MTPR	Modified Time Parameter Rule
NCR	Nearest Container Rule
NE	Non-Equal Instances
PFSP	Permutation Flowshop Scheduling Problem
PSK	Problem Specific Knowledge
PSO	Particle Swarm Optimization
PSP	Post-Stacking Problems
QCAP	Quay Crane Assignment Problem
QCs	Quay Cranes
QCSP	Quay Crane Scheduling Problems

RMG	Rail-Mounted Gantry
RPD	Relative Percentage Deviation
RTG	Rubber Tire Gantry
SCs	Straddle Carriers
SDGs	17 Sustainable Development Goals
SETR	Simulated Ending Time Rule
TEU	20-Foot Equivalent Unit
TPR	Time Parameter Rule
TSP	Traveling Salesman Problem
UNCTAD	United Nations Conference on Trade and Development
UNDESA	United Nations Department of Economic and Social Affairs
VND	Variable Neighborhood Descent
VRP	Vehicle Routing Problems
YCs	Yard Cranes
YCSP	Yard Crane Scheduling Problem

CHAPTER 1

Introduction, motivation and objectives

Although humans have known shipbuilding for economic activities thousands of years ago, the history of motorized vessels predates us by just a few hundred years, and freight containers, just decades. In this section, we introduce readers first to the motivations of the research from a historical perspective to a sustainable development future in Section 1.1. Next, in Section 1.2, we focus on the container terminal with related equipment, layouts, and basic operations, following with the objectives of this research in Section 1.3 and the outline of the thesis in Section 1.4.

1.1 Motivation of the research

This section first introduces a brief history of the maritime trade, originating from the wooden ship trade age. Then we give an overview and basic knowledge of the current intentional maritime trade. In maritime trade, one of the most important categories, containerized

trade, is discussed in the next section, and the motivation of this study which is to build a sustainable future, is addressed in the last.

1.1.1 A wooden ship trade age

Since the beginning of civilization, mankind has never stopped exploring the unknown beyond. Thus came sailing. The first attempt for humans to build boats sailing to the sea is still uncertain. However, it is recorded that as early as 5,000 years ago, when the first urban civilization formed in China, Egypt, India, and Mesopotamia, people were already accomplished seafarers and seafaring facilitated travels across the globe (Stein, 2017). Although in their era, the wooden boat, easily degraded by worms and even water, was the only choice for crossing big rivers or even the sea, gradually became an important tool to promote maritime trade.

The purpose of crossing the sea in ancient times is not well known, maybe for the reason of trade, transport, warfare, or fishing. The earliest sailing recorded in the pictorial representation is from Egypt, dated to approximately 3rd century BCE[1] (Casson, 1995). In the same era, the Austronesian expansion also started in Taiwan island and eventually spread across the Pacific (Bellwood *et al.*, 2006).

In the Age of Navigation (1st BCE to 15th century CE[2]), sailing over the seas started to serve for trade. One clear commercial activity was recorded in the 1st BCE of Austronesian. They were already engaged in regular maritime trade with China, South Asia, and the Middle East for exchanging of cultivated crop plants, introducing Pacific coconuts, bananas, and sugarcane to the Indian subcontinent, some of which eventually reached Europe via overland Persian and Arab traders (Mahdi, 1999). The boat they used was named "Kunlunbo" in Chinese or "kolandiaphonta" known by the Greeks which owns 4 to 7 masts

[1]BCE: Before the Common Era
[2]CE: the Common Era

against the wind due to the usage of "Tanja" sails. In the 7th to 13th century CE, the Arab Empire expanded a wide trade network across parts of Asia, Africa, and Europe, reaching the border from Spain to China.

In the Age of Discovery (15th to 17th century CE), sailing activities were serviced for colonization and trade. The pioneer of Portuguese and later Spanish made long-distance maritime travels in search of alternative trade routes to "the East Indies," promoted by the trade of gold, silver, and spices, after which it turned to the British, French and Dutch from Europeans to cross the Atlantic. At a similar age began the Zheng He, a maritime operation ordered by the Ming dynasty. Fleets of 62 to 300 wooden ships (Finlay, 2008) were employed with porcelains, teas, silks, and spices, visiting over 30 countries across the Indian Ocean.

In recent centuries, wooden ships were phased out for sea trade after the first use of iron and steel in the fleet for military use in the 1820s. Moreover, benefiting from the development of engine technologies after the Industrial Revolution, commercial ships are now getting safer and more capable of commercial activities. The wooden ships had been linking the communication between different continents for thousands of years and witnessed significant changes in this world over decades. An era of international maritime trade all over the world was coming.

1.1.2 Contemporary international maritime trade

Nowadays, the world is closely connected by international maritime trade and has become very dependent on it. An interesting incident has shown this kind of deep dependence. In March 2021, the maritime ship Evergreen blocked the Suez Canal, leading to the temporary panic buying of toilet paper in Europe (Kay, 2021). To have a statistical review of the intentional maritime trade, since 1896, every after the United Nations Conference on Trade and Development (UNCTAD), the

Department of Division on Technology and Logistics, publishes a review of maritime transport (Shamika *et al.*, 2021) intending to foster the transparency of maritime markets and analyze relevant developments. This review analyzes statistical data on structural and cyclical changes affecting seaborne trade, ports, and shipping, as well as the major events in the maritime industry. Most of the review data are collected from the UNCTADStat (http://stats.unctad.org/maritime) by the UNCTAD, in which the annual or semi-annual statistical data from merchant fleets to world seaborne trade are open access. Based on the latest review (Shamika *et al.*, 2022), the international maritime trade flows in 2021 have bounced back to a total of 10.99 billion tons in volume after a small growth of 3.29% from 2020 (10.65 billion tons). Although, it is slightly lower than the pre-pandemic level of 2019, which keeps the historical record around 11.02 billion tons.

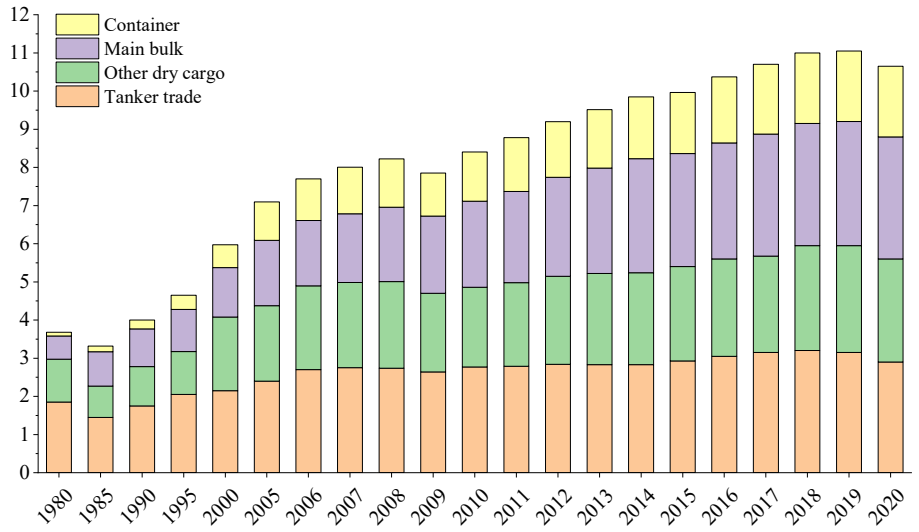


Figure 1.1: Volume of international seaborne trade from 1980 to 2020 in Millions of tons loaded. Data Source: the report of Review of Maritime Transport 2022 from the UNCTAD (Shamika *et al.*, 2021, 2022, 2023). Elaborated by the author.

In recent years, international maritime trade was also affected by

COVID-19 starting from the end of 2019. Its growth rate had already been weak from 2019 at 0.5%, but in 2020 it declined by 3.8% (10.65 billion tons). Similarly, the world container port traffic went down by 1.2 % to 815.6 million Twenty-feet Equivalent Units (TEUs) in 2021. The decline mainly resulted from the pandemic disrupting the world economy, cutting manufacturing activity and consumption with impacts on supply, demand, and logistics in 2020 (Shamika *et al.*, 2021). Even though the impact was not as dramatic as the initial crisis. In 2022, UNCTAD projects that the maritime trade will be able to grow moderately to 1.4% and will keep increasing in the short term. From 2023 to 2027, the growth is estimated to expand at an annual average of 2.1%, a slower rate than the previous three-decade average of 3.3%.

For many years the fastest-growing segment has been containerized trade. Figure 1.1 shows the volume of international maritime trade between 1980 and 2020 by cargo type in Millions of tons. Over the past decades, maritime transport increased by 289% in volume. Concerning the type of cargo transported, the container trade boosted about 18 times, making it the fastest-growing category. Even in more recent years (2010-2020), the number increased by 43%. The main bulk commodities, including mainly iron ore, grain, and coal, increased by 462% from 1980 and, in recent decades, by 42%. The data for the other dry cargo is 140% from 1980 and 29% from 2010. It should be noted that the main bulk includes iron ore, grain, coal, bauxite/alumina, and phosphate. Note that from 2006, "Main bulk" includes iron ore, grain, and coal only. Data relating to bauxite/alumina and phosphate are included under "Other dry cargo". The tanker trade includes crude oil, refined petroleum products, gas, and chemicals, keeping much stable and increasing 57 in the past forty years and only 5% in 2010 to 2020. The growth in 2022 is projected to be a tepid 1.2 %, before marginally picking up to 1.9 % in 2023 (Shamika *et al.*, 2023).

1.1.3 A containerized world

Containerizing is a kind of systematically standardized transportation. The transportation of containers is a great success in commercial trade nowadays. However, the use of containers started a few decades ago. In the 1950s, a steel box was invented for the entire delivery of cotton by truck driver Malcom McLean (Talley, 2000). Before his time, the delivery on terminals of the maritime trade could be regarded as a physically hard and labor-intensive job. This situation prompts the utilization of box containers for the great reduction of labor and convenience of entire transfer by machines. Unsurprisingly, these steel boxes with 2.591m height, 2.438m width, and two lengths of 20 equivalent feet (6.09m) and 40 equivalent feet (12.192m), as shown in Figure 1.2, quickly became popular and ubiquitous for the maritime trade of commercial goods. The standard 40 equivalent feet container is now one of the most commonly-used containers for the international transportation of ocean freight in which one container is able to offer 67.7 cubic meters of space and hold 30 tons of goods (Hai *et al.*, 2020). It is one of the greatest advances ever made by humankind to be capable of standardizing such huge goods in one load with several tenths of tonnes and to use it as a basis for the gradual realization of a global logistics system with ships, ports, routes, roads, transit stations, bridges, tunnels, and inter-modal transport (Parreño Torres, 2020).

Nowadays, containerized trade has been the increasingly important mode of transport for general commodities in maritime trade. In 2021, global container port throughput reached 857 million TEUs, with an increase of 6.8% (Shamika *et al.*, 2022). Although one year before, it faced a fall of 1.2% and 1.1% by 2019, respectively (Shamika *et al.*, 2022). The growth momentum of containerizing in maritime trade the had also been witnessed by container ships. In Figure 1.3, the global container fleet capacity has been expanded 28 times, up to 293 million

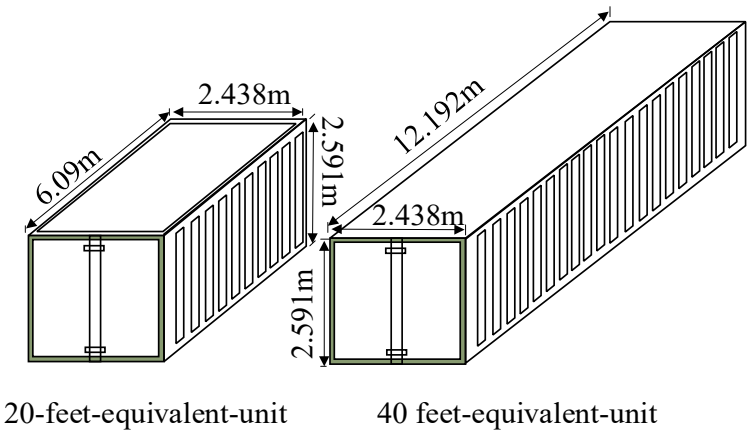


Figure 1.2: The standard container specifications for 20-feet-equivalent-unit and 40-feet-equivalent-unit. Elaborated by the author.

DWTs[3] in which more and more Neo-Panamax[4] or Post-Panamax[5] level container ships being built every year. In general, this kind of expansion has lasted for about 40 years, even during the recent pandemic crisis. What is more, the prosperity of containerized trade is also reflected in the proportions of container ships to registered merchant fleets. The proportion, which becomes an indicator of the developing trend of containerized trade, has reached up to 13.34% since 1980, and in recent decades, has increased by 771% from 1.53%.

1.1.4 Sustainable development goals for the future

The increasing maritime trade also puts pressure on sustainable development. The pressure turns to the global environment since energy consumption in the terminal port greatly influences the carbon footprint. Excessive energy consumption will influence not only the average

[3]DWTs: Dead Weight Tons

[4]Neo-Panamax: fleets can transit the expanded locks of the Panama Canal with up to a maximum 49m beam and 366m length overall with the capacity of 12,000 to 14,000 TEUs

[5]Post-Panamax: fleets with a capacity greater than 15,000 TEUs include some ships that are able to transit the expanded locks of the Panama Canal

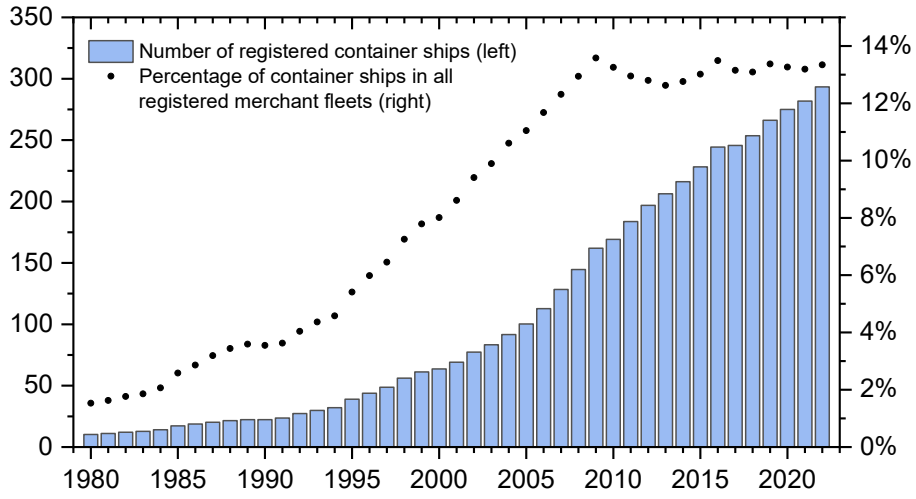


Figure 1.3: Annual registered container ships in millions of Dead-Weight-Tons (bars) and its proportion (points) in merchant ships from 1980 to 2022. Data Source: UNCTAD dataset (UNCTADStat, 2022).

temperature of the seas but also all the creatures on our planet, and the outcomes may continue into the future. To protect the global environment for now and the future, the United Nations Department of Economic and Social Affairs (UNDESA) provides substantive support and capacity-building for related issues, including water, energy, climate, oceans, urbanization, transport, science, and technology. In 2015, all United Nations member states adopted the 2030 Agenda (UNDESA, 2015) for sustainable development. At its heart are the 17 Sustainable Development Goals (SDGs), which stand the urgent call for action by all countries, developed and developing, in a global partnership. The SDGs aim to achieve the 17 goals in almost all areas of economics, including *no poverty, zero hunger, good health and well-being, quality education, gender equality, clean water and sanitation, affordable and clean energy, decent work and economic growth, industry, innovation and infrastructure, reduced inequality, sustainable cities and communities, responsible consumption and production, climate action, life below water, life on land, peace, justice, and strong institutions, partnerships*

for the goals (UNDESA, 2015).

Commercial trade activities with maritime transportation are highly correlated with the SDGs. The SDGs require more efficient delivery of maritime trade to save energy and reduce carbon emissions, especially in terminal ports. The total cost of energy consumption on handling equipment accounts for 15% to 25% of the total operational cost in a container terminal (Hernandez *et al.*, 2012). For instance, the total energy consumption for the container terminal of Tianjin port in 2013 was around 47 million kWh for handling 2.32 million TEUs (Twenty-feet Equivalent Units), almost equal to the power needed for a middle city (He *et al.*, 2015a). With regard to responsible energy consumption, it is possible to save energy by improving operational efficiency in terminals. One feasible way is to decrease the unproductive loading and unloading moves for heavy machinery, e.g., the yard crane, which is almost the most frequently-used handling equipment in a container yard, accounts for approximately 25 to 35% of the total energy consumption cost (He *et al.*, 2015a).

1.2 Container terminal

In this section, we provide the basic information about how the terminal basically operates. Firstly, the equipment like cranes and vehicles working in the terminal yard are addressed in Section 1.2.1. In Section 1.2.2, two basic container port configurations, Asian and European, are introduced. The operational research on the terminal ports is reviewed in Section 1.2.3.

1.2.1 Container terminal equipment

Plenty of equipment is employed nowadays at container terminals. Figure 1.4 details the main material handling equipment which includes quay cranes (QCs), yard cranes (YCs), Straddle carriers (SCs), Auto-

mated Guided Vehicles (AGVs), trucks, and trains. In the terminal, vehicles (AGVs and trucks) are used to transship containers between ships and QCs, QCs and YCs, YCs and the yard, etc. Usually, containers can be transshipped directly from one mode of transportation to another and vice versa. Alternatively, containers can also be stored in the yard for a period before being transferred to any mode.

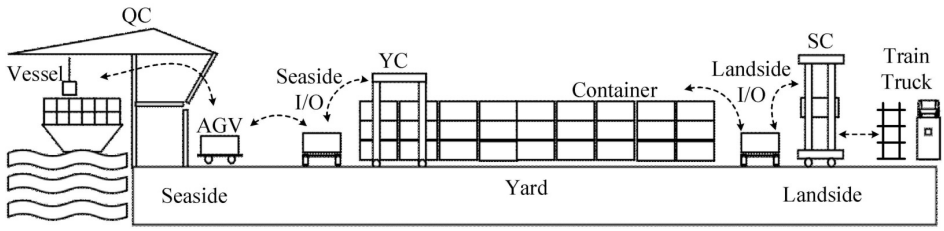


Figure 1.4: The equipment and container transferring process in a typical container terminal. Elaborated by the author.

The equipment located in different areas takes charge of various functions. In a classical container terminal, QCs stacking over the berth are movable along the quay. They are the more expensive equipment in the terminal. Generally, containers imported or exported overseas are transshipped by QCs working between the vessels and the seaside of the yard. To transfer containers after unloading from ships, vehicles are employed to deliver the container beneath QCs to the yard transfer area (Input/Output points in this thesis, see Section 1.2.2) at the seaside of each yard block (the detail of the yard block is given at Section 3.1). Trucks are usually used for container handling at manned terminals. For automated terminals, AGVs are preferred. At the landside, trains and trucks are usually adopted for container transportation from the outside to the landside of the yard block. From both sides, containers travel between the transfer area to the yard block by YCs working over the block. YCs need to pick up containers from the transfer area, in the seaside or the landside, to the yard block (inbound) and vice versa (outbound). These requests of delivering containers are defined in Section 3.2. Specifically, YCs also take charge of vehicle loading

and unloading operations in the transfer area (Input/Output points). Moreover, the delivery for retrieving one container from the yard to the transfer area can also be executed by the SCs in the seaside of the transfer area, which allows YCs to leave without waiting for vehicles for higher operational efficiency.

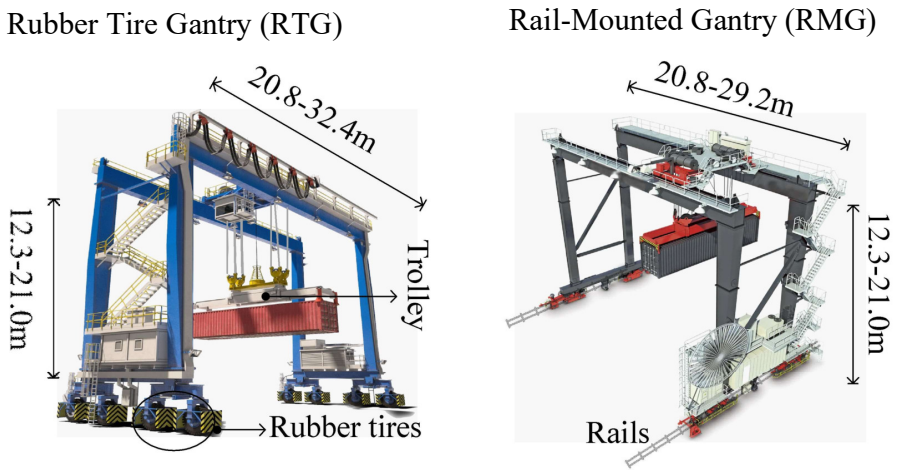


Figure 1.5: Main types of yard cranes in container terminals, Rubber Tire Gantry (RTG) and Rail-Mounted Gantry (RMG) with related span distances and lifting heights. Source: Liebherr (2023); Konecranes (2023). Elaborated by the author.

Compared to QCs, YCs are more often used for most of the loading/unloading operations since they are smaller and more flexible. Shown in Figure 1.5, there are two main types of YCs, Rubber Tire Gantry (RTG) cranes and Rail-Mounted Gantry (RMG) cranes. As the name suggests, RTG moves with tires, and RMG moves with rail-mounted wheels. Usually, these cranes are able to hoist a container to 12.3 to 21m (5 to 8 tiers of containers in Figure 1.2) with a span distance of more than 20m, for some big gantries, even 70m. Differently, the RMG is usually the automated yard crane where the loading process is executed automatically. Nevertheless, the service area is limited to a few adjacent blocks in the same rail row. In contrast, the RTG crane is more manual and requires drivers to operate it. But, it is easier to

move from one block to another.

Table 1.1: Speeds of hoisting, trolley and gantry for Rubber Tire Gantry (RTG) and Rail-Mounted Gantry (RMG). Source: Liebherr (2023); Konecranes (2023)

	Speed without load			Speed with load		
	Hoisting	Trolley	Gantry	Hoisting	Trolley	Gantry
RTG	0.93m/s	1.17m/s	2.16m/s	0.46m/s	1.17m/s	2.16m/s
RMG	1.03m/s	1.37m/s	2.25m/s	0.52m/s	1.37m/s	2.25m/s

However, the efficiency of yard crane loading has been the bottleneck of the yard terminal (Ng and Mak, 2005a; Li *et al.*, 2009) not only for the high operational load but also for the slow movement in the loading procedure. Based on Ng and Mak (2005a), YCs are already the most common equipment in container terminals, where around 63.2% of 114 container terminals in the world, and 75.5% of container terminals in Asia, have equipped YCs (Wiese *et al.*, 2010). Nevertheless, at the operational level, YCs are usually bulky, slow and need to move frequently between different locations (Ng and Mak, 2005a; Li *et al.*, 2009). Table 1.1 presents the speeds of RTG (Liebherr, 2023) and RMG (Konecranes, 2023) for different movements on hoisting, trolley traversing, and grant moving. Both the RTG and RMG cranes move at very low speeds, with a maximum of 2.16m/s and 2.25m/s, respectively. It is mainly because a fully loaded container is very heavy (Hai *et al.*, 2020) (30 tons, Section 1.1.3), making it hard and dangerous to load fast by the crane. Further details about the technical parts of the yard crane can be found in Stahlbock and Voß (2008a).

1.2.2 Container terminal layouts

The container terminal is now a hub of multiple modes of transport, including the ground transport modes of domestic trains or trucks and the offshore shipping modes to seas or inner rivers. In general terms, the terminal hub can be described as an open system of container flows

with two external interfaces. One of the interfaces is connected to the quayside with inbound/outbound container ships. The other is located on the landside interfacing with external vehicles (Steenken *et al.*, 2004). The terminals are now constructed mainly in two configurations of layouts, commonly named Asian and European terminal layouts (Liu *et al.*, 2004; Lee and Kim, 2013; Carlo *et al.*, 2014). As shown in Figure 1.6, the main difference exists in the direction of container stocking and container transfer area, which is the location for vehicles to exchange containers with the yard crane. In the Asian layout, the vehicles usually arrive at destination bays and load/unload containers under the crane. However, in European layout, containers are commonly delivered through the Input/Output point (I/O) at the transfer area near the yard block's edges to the quay or trains.

- Asian configuration: in the terminal port, the direction of container stocking in rows is parallel to the quay, and there is a truck lane along the crane moving direction. This layout is mainly used in non-automated storage yards in which the yard crane moves across the block with a lean distance for vehicles. The crane is stacked vertically to the quay, transferring containers in a parallel direction with the crane moving between the yard block and landside (see Figure 1.6 Asian terminal layout). The layout is quite common in large Asian terminals and, therefore, referred to as the Asian layout.

- European configuration: differently, the direction of container stocking in rows is exactly vertical to the quay, and IOs are commonly implemented. The IOs are set in the transfer areas located at the ends of the yard blocks for handling storage and retrieval requests in both the seaside and landside (see Figure 1.6 European terminal layout). Typically, this configuration is usually used in the automated yards where unmanned cranes and automated guided vehicles (AGVs) or Straddle Carriers (SCs) are

used in the entire delivery process. This configuration layout was first implemented in some European container terminals.

Compared to the Asian layout, the European layout shows some advantages, like lower operational costs (even though with higher investment costs), faster crane speeds, and more storage capacity (Carlo *et al.*, 2014). The main reason counts on the wider yard crane arm, which is able to stack more rows since there are no truck lanes. To cooperate with wider block rows, automated stacking cranes (ASCs) are generally used in European terminals, which allow accelerating and decelerating operations in the loading/unloading process. Moreover, the European layout saves travel distances directly from the yard crane to the I/O since they are next to the seaside and landside area.

Shown in Figure 1.6 and as outlined in previous sections, both the Asian and European container terminal layouts consist of three areas: seaside, yard, and landside for different functions.

- The *seaside* area is the connection between the sea and yard transports located at the near-sea part of a terminal. This area ranges from the berthing dock of the vessels to the IOs on the near side, where hundreds or even thousands of containers in a container ship can be transferred at the berthing time. However, this time is quite limited, especially for big terminal ports. To a certain extent, the seaside is more important since it relates to not only loading/unloading containers between two sides but also the efficiency of all following procedures of quay cranes and container ships.

- The *yard* is normally regarded as the warehouse of containers consisting of several blocks, which is usually the largest area in a terminal block. All storage containers (import container, inbound container) need to be delivered to this area for temporary storage, whether from the landside or the seaside. Similarly, retrieval

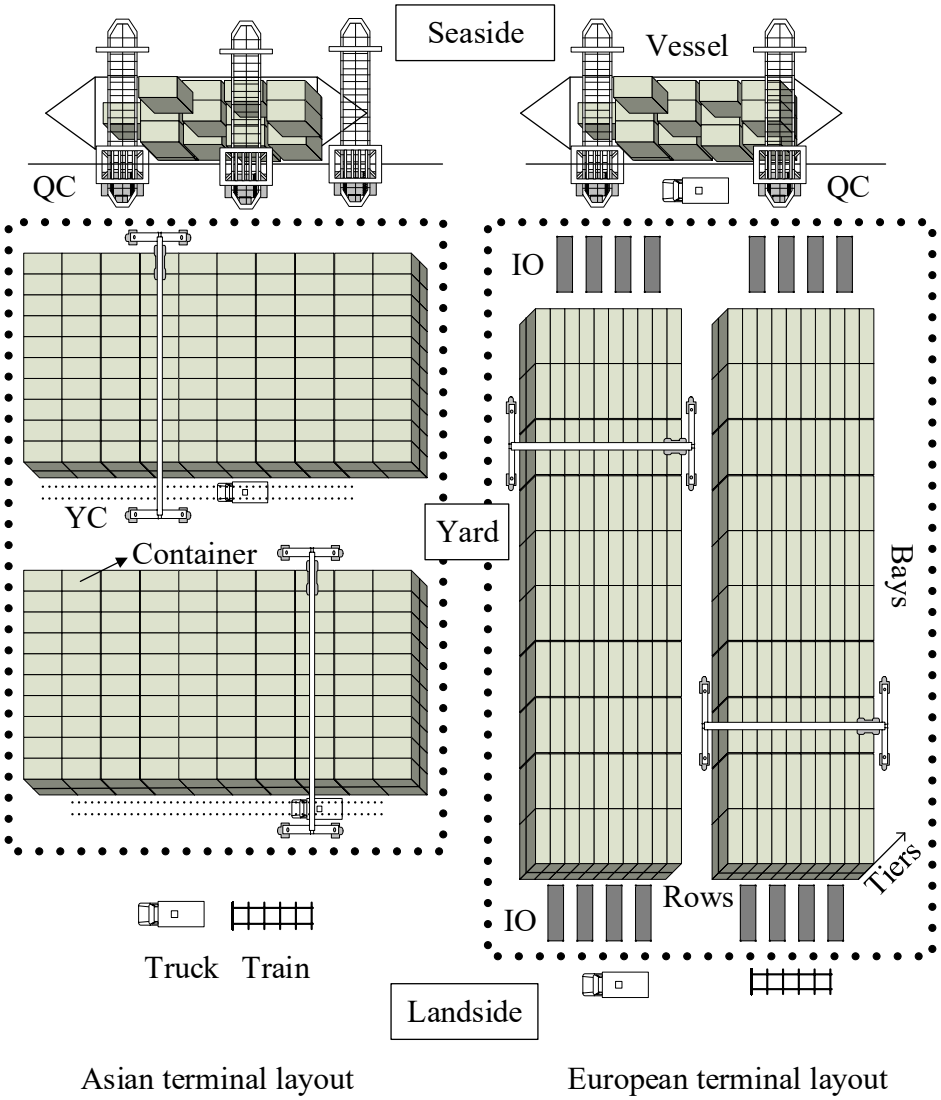


Figure 1.6: The Asian and European terminal layouts. Elaborated by the author.

containers (export container, outbound container) are already stacked in this area before being loaded onto trains, trucks, or vessels. Since almost all containers have to be loaded or unloaded in the area, the efficiency of the yard has become the bottleneck of the container terminal (Ng and Mak, 2005a; Li *et al.*, 2009).

- The *landside* area is the connection between the land and yard transports in a near-land part of a terminal. This area is the container exchange point for the external trains and trucks where containers are transferred from or to the mainland outside the terminal. Different from the seaside, extra cranes are not required for loading or unloading containers in this area, and these operations can be accomplished by YCs or SCs in most cases due to the container moving in a similar height level.

In recent years, automated terminals are increasingly popular in which automated systems are usually equipped with respect to delivering, loading and discharging containers. With these systems, the whole process of completing container requests can be automatically achieved by unmanned machines, including QCs, YCs, AGVs, and SCs. At the same time, new changes can be noticed in the automated terminal: (1) the European terminal configuration is usually employed to simplify the routes and decrease the travel times of vehicles from the yard to the seaside or landside, e.g., Shanghai port, Yangshan terminal 4; (2) all vehicles must deliver in/out the container to the yard by one of the I/O points located in both sides of the block and SCs are implemented to transfer containers between cranes and vehicles. The SC works in the I/O to receive the container from vehicles and to deliver it to the YC for a loaded move. In reverse, the SC receives the container and sends the container to vehicles in the unloading operation. This setting yields a higher efficiency of the YC in I/O points where the crane is allowed to leave the I/O immediately instead of waiting for the container vehicle to come. In this thesis, we study an automated terminal with the European terminal layout and I/O points.

1.2.3 Container terminal and operations research

A typical classification of container terminal planning problems is given by Bierwirth and Meisel (2010) in Figure 1.7. The classification is based

on the time horizon for the decision, which can be categorized into strategic, tactical, and operational decisions (Schmidt and Wilhelm, 2000). Strategic decisions often determine long-term actions, e.g., the location of the container terminal, the number of cranes, and the layout of the terminal (Stahlbock and Voß, 2008a; Pawlik *et al.*, 2011; Cartenì and De Luca, 2012). In a shorter time period, tactical decisions concern the decision-making over several months where the operators need to design the berth position of ships in the quay or the space for container stocking (Giallombardo *et al.*, 2010; Cartenì and De Luca, 2012; Lalla-Ruiz *et al.*, 2014; Jin *et al.*, 2015). Most of the studies commit to the operational decisions in the terminal, where the arrangements work for days or even hours. Based on the analysis of a total of 226 papers on yard terminal planning in Dragović *et al.* (2017), the application of the simulation model at the operational level attracts most of the attention in the literature in which around 75.8% of the papers focus on operational problems from vessel berthing location for container ships to the equipment allocation for containers.

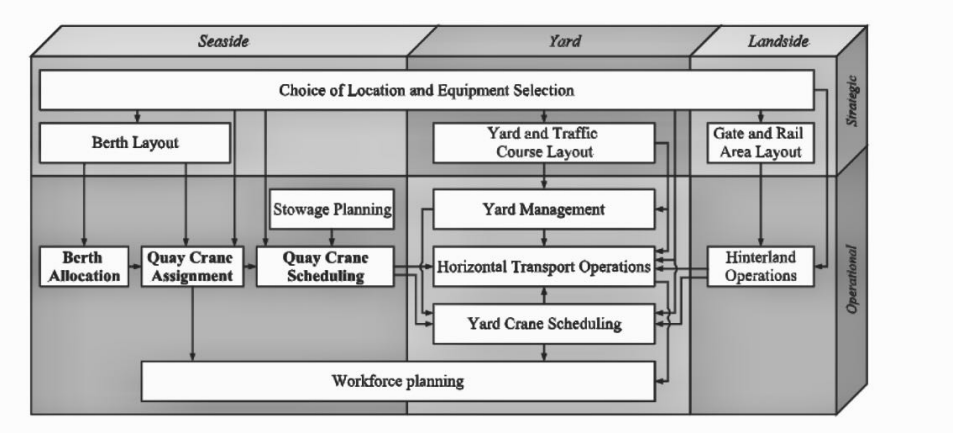


Figure 1.7: Container terminal planning problems classification (Bierwirth and Meisel, 2010).

Similarly, in this thesis, we focus on the operational problems in the terminal port. Almost all the operations about container delivery

in the daily routine of the terminal are included in this level, from the quayside planning problems about the vessels and quay cranes to the yard planning in the yard with vehicles and yard cranes. Some typical problems of yard management at the operational level are listed below:

- Allocation / Assignment problems for the berth and the quay crane

  Berth Allocation Problem (BAP): The problem of allocating berth spaces and berthing times to incoming vessels is known as the berth allocation problem (Rodrigues and Agra, 2022). Since the berth space is limited in the seaside and plenty of containers have to be handled every day in the major terminals, it is critical to control the container traffic flow in an efficient way to minimize the berthing time or weighted flow (Guan and Cheung, 2004). A typical berth at the container terminals can accommodate multiple vessels at the same time in case there is no berthing space available, a vessel needs to wait for mooring.

  Quay Crane Assignment Problem (QCAP): Except for the berth location for the vessels, the assignment for the quay crane for the coming vessels is also required for the terminal operators, which assigns cranes that will carry the required operation on each vessel (Abou Kasm *et al.*, 2020). To cope with the large container vessels which may need several quay cranes, the cranes have to be selected and decided for various ships considering the accessibility of cranes at the berth and the impossibility of exchanging cranes between different berths at the terminal. More specifically, cranes operating on one ship have to be assigned to different sections or hatches of the ship. Therefore, as the most expensive equipment in the terminal port, the quay cranes are crucial to improve the efficiency on the quayside. The assignment of quay cranes aims at minimizing the operational loading time

with the limited number of quay cranes (Diabat and Theodorou, 2014).

- Routing problems

  Vehicle Routing Problems (VRP): Although the VRP is not limited to container terminals, many researchers have integrated the problem into the container loading process (Stahlbock and Voß, 2008b; Moura and Oliveira, 2009; Vidović *et al.*, 2011; Fazi *et al.*, 2020). The problem is to optimize the route for vehicles, like yard trucks or AGVs in the yard terminal. Since the routing is roughly fixed between different areas in the container terminal, the problem mainly aims to minimize the number of vehicles or total travel time for all the container tasks in a period of time.

- Scheduling problems

  Quay Crane Scheduling Problems (QCSP): In this problem, the tasks of containers in a ship, including discharge and loading, are scheduled on a set of employed quay cranes. A mandatory constraint that the QCSP should follow is that these cranes cannot cross each other even though they can be moved to every vessel, in the consideration that all QCs are mounted on a single rail track alongside the quay (Legato *et al.*, 2012). The main objective of the QCSP is to determine the sequence of loading and unloading operations of a vessel with a limited number of quay cranes so that the completion time of the operations is minimized (Al-Dhaheri and Diabat, 2015). Actually, the problem is similar to a series of flowshop scheduling in consideration of parallel machines where every task must be processed once by a QC, while each QCs can process at most one task at one time. It should be noted that due to the relation between the QCSP and QCAP, researchers often integrate both into a Quay Crane Assignment and Scheduling Problem (QCASP) where cranes are assigned to vessels and their

operations are scheduled (Agra and Oliveira, 2018; Rodrigues and Agra, 2022).

Yard Crane Scheduling Problems (YCSP): Different from the QCSP, the scheduling problem on the yard crane focuses more on specific delivery operations for a series of tasks to be finished by YCs. Without loss of generality, operational constraints are used in the basic version of the YCSP (Ng and Mak, 2005b). E.g., one container must be processed by only one YC, and each YC can process at most one task at a time. Researchers have introduced more details to the problem recently, like the storage and retrieval containers (Vis and Roodbergen, 2009), I/O allocation (Gharehgozli *et al.*, 2014), and the relocation of containers (Galle *et al.*, 2018). The problem aims to minimize the makespan of the container scheduling or, alternatively, minimize the weighted delay in most of the literature. The efficiency of the yard crane is critical for maritime terminal throughput since YCs take charge of all of the loading and unloading tasks making the YCs a key equipment, where the low efficiency of the yard crane has become the bottleneck in the container terminal (Ng and Mak, 2005a; Li *et al.*, 2009).

- Rehandling Problems

  Post-Stacking Problems (PSP): The PSP, also named stacking problem, focuses on the rehandling operations after the stacking of containers. The problem determines the positional adjustment after stacking in the yard with the consideration of future operations before the next loading in advance, like retrieving and reshuffling (Lee and Hsu, 2007). For instance, heavier containers should be stacked in higher tiers because they are loaded in lower tiers of vessels and should be retrieved earlier. Three types of post-stacking problems have been identified by Caserta *et al.* (2011a), including the *remarshalling problem*, the *premarshalling problem*

and the *relocation problem*. The remarshalling problem reshuffles containers in the container yard, while premarshalling operations happen only inside the single bay, and the relocation problem adjusts locations in a fast way to retrieve the container when containers have already arrived (Tanaka and Takii, 2015). The rehandling problem seeks an advanced relocation on the container stacking to minimize the number of reshuffles or, alternatively, the reshuffling time in such a way that future loading operations are carried out with maximal efficiency.

1.3 Objectives of the thesis

This thesis mainly focuses on improving the operational efficiency in the scheduling process of a single yard crane by employing different methods for problems with increasingly practical generalizations and constraints. New algorithms employing heuristics and metaheuristics are proposed with constructive methods and local search heuristics to improve the solving efficiency of this problem.

1.3.1 General objectives

Nowadays, terminal ports face two important challenges. On the one hand, the increasing demand for maritime trade puts pressure on terminal ports' processing capacity on both container volumes and vessel sizes. Based on the latest data, the traffic of world container ports has reached 857 million TEUs and may increase further (Shamika *et al.*, 2023). Moreover, newly built cargo ships are getting larger in carrying capacity where the maximum container ship capacity has reached a historical 24,004 TEUs (UNCTADStat, 2022). Bigger ships dealing with more containers require higher loading efficiency in the limited berthing hours. On the other hand, the vision of green and low carbon emissions on terminals (see Section 1.1.4) is generally accepted

by most terminal ports in the world, putting higher expectations on more efficient terminal operations with less energy consumption.

Effective ways have been proposed to cope with these challenges, like increasing terminal operational areas (yard blocks) and expanding the equipment (cranes and vehicles). However, this is not profitable for container operators. Not to mention that the terminal can not infinitely increase in size, as well as the environmental impact of building and operating large terminals. Therefore, in this thesis, we turn to finding effective ways to schedule container operations, like loading and unloading procedures in the terminal yard with existing resources. More specifically, we study a series of basic scheduling problems with single yard crane in one block to optimize the delivery procedure of all container requests. Solving the yard crane scheduling problem (YCSP) with more efficient algorithms decreases the number of unproductive or unnecessary moves of the yard crane, which is exactly beneficial to not only reducing the bottleneck of container terminals but also to fulfilling the vision of a green and sustainable future.

1.3.2 Specific objectives

In this context, we develop, calibrate, and evaluate a series of simple yet effective heuristic and metaheuristic methods to support the decisions of container terminal operators at the operational level in a yard terminal.

Inspired by the motivation in Section 1.1 and the introduction in Section 1.2, we focus on the container scheduling problem of the yard crane where multiple decisions should be made including the loading order of containers and the assignments of Input/Output (I/O) points. The thesis proposes efficient solving methods to the single crane sequencing problem with a given set of container storage and retrieval requests (described in Section 1.1.3) in the yard block. We consider a more realistic YCSP with a dynamic assignment of I/O points where the timetable of the container, like the I/O starting time,

is related to the release time or due date determined by vehicles or ships. Furthermore, the I/O points can be assigned to containers as early as release times for container delivery. In this way, inefficiency from the lateness or the congestion can be the result of timeouts or I/O congestion, which greatly influences container handling operations and thus should be minimized. However, the problem concerning the realistic considerations above is hard to solve. Current exact methods on yard scheduling problems are valid for simple settings on cranes or models, like a single-row crane (Vis and Roodbergen, 2009) or TSP-like model formulation (Gharehgozli *et al.*, 2014). Integer Programming tools, like commercial solver, IBM ILOG CPLEX, can barely solve instances with more than ten containers within one hour (Vallada *et al.*, 2023). Therefore, we aim to develop heuristics and metaheuristics to solve this realistic YCSP more effectively and efficiently.

Firstly, we propose a simple rule-based heuristic method. This approach includes a constructive method for container sequencing and an assignment method for I/O points. For sequencing, we introduce a Simulated Ending Time Rule (SETR), which orders containers based on their finishing times of the yard crane, simulated by relevant time parameters and I/O points. The assignment of I/O points is then determined based on this fixed sequence. For various container types (classified in Section 3.2), the heuristic selects the most time-efficient available I/O point. This method is very simple and requires no calibration. Comparable rule-based heuristics were developed in Vallada *et al.* (2023), where we will demonstrate that the proposed SETR rule outperforms all existing heuristics from Vallada *et al.* (2023).

While the SETR method demonstrates efficiency in solving small instances of the YCSP, it encounters significant challenges when applied to such realistic problems, especially in cases with larger instances. Current exact approaches, such as mixed-integer programming and various branch-and-bound or branch-and-cut techniques, can be difficult to

apply to the problem. In contrast, metaheuristic methods offer a promising alternative by delivering near-optimal solutions within a reasonable timeframe, making them particularly well-suited for addressing such kinds of problems. This thesis introduces two metaheuristic approaches: the Greedy Randomized Adaptive Search Procedure (GRASP) and Iterated Greedy (IG) methods.

The GRASP methods developed in this work are specifically designed for the YCSP. These methods initiate with new constructive methods to yield feasible solutions, using SETR as the initial best solution. A critical component of the proposed methods is an advanced local search algorithm, a common operator in GRASP approaches, which is employed to further enhance the quality of the generated solutions. These procedures iterates repeatedly until a certain of stop criterion is satisfied where the best solution obtained will be output. A series of careful calibrations has been made for all proposed GRASP methods, and comprehension experiments prove the proposed methods outperform the state-of-the-art approaches.

To obtain even better solutions, three simple yet efficient IG methods are developed for the YCSP. IG has shown considerable potential in solving a variety of scheduling problems. Building on the foundational structure proposed by Ruiz and Stützle (2007), new operators are designed to address the unique challenges of this realistic problem, including novel methods for destruction and reconstruction, along with highly effective local search techniques. Moreover, a generic local search framework is developed, enabling the design of the most efficient local search strategies applicable across different heuristics. Each IG method is carefully crafted, starting from a basic vanilla IG and incorporating modifications that, although small, result in significant improvements as demonstrated by an extensive experimental analysis.

1.4 Outline of the thesis

The rest of the thesis is organized into several chapters, outlined as follows:

Chapter 2: Literature review. This chapter provides a comprehensive review of the existing literature. It begins with a general overview of scheduling problems across various research domains, including computing, agriculture, industry, and transportation. The chapter then focuses on related problems, particularly those that connect yard crane scheduling with container delivery processes in port terminals. Eventually, we present a detailed review of the YCSPs, including fundamental settings, single and multiple/combined YCSPs, and the various solution methods applied in the existing literature.

Chapter 3: The yard crane scheduling problem. This chapter introduces the YCSP investigated in this thesis. It begins by outlining the basic configurations and assumptions related to terminal yards, followed by a discussion on the different types of container requests encountered within these yards. The chapter then explores the key decisions that should be made in a real-world container terminal, as well as the objective function adopted for this YCSP. The following sections introduce a Mixed Integer Programming (MIP) mathematical model for this YCSP with the assignments of Input/Output points. Comprehensive descriptions of the investigated YCSP are provided in this chapter, including specific problem settings, relevant nomenclature, loading operations, and the formulation of the objective function.

Chapter 4: Constructive heuristic methods. This chapter explores constructive heuristic methods, beginning with an introduction to the topic and a presentation of solution approaches. It then details the steps involved in the constructive heuristic process, including the generation of container sequences, I/O assignment and timing, sequencing methods, and dynamic assignment techniques. The chapter also discusses a local search method as a complement to the heuristics.

It concludes with computational experiments, providing benchmark instances and analyzing the computational results. Finally, the chapter wraps up with key conclusions drawn from the study.

Chapter 5: Solving the YCSP with Greedy Randomized Adaptive Search Procedures (GRASP). This chapter introduces metaheuristic methods to the investigated YCSP in this thesis. Two GRASP heuristic approaches are proposed along with improvements to the solution construction and local search phases that are specifically tailored for this problem. The proposed algorithms are statistically calibrated using the Design of Experiments (DOE) methodology (Montgomery, 2017). Comprehensive experiments are designed to assess the performance of the method, including validation across small, medium, and large size instances. As will be shown in later sections, the proposed GRASP methods are efficient and competitive. They result in a high rate of optimality for small instances and find much better solutions for medium and large instances when compared to the existing approaches from the literature.

Chapter 6: Solving the YCSP with Iterated Greedy methods. This chapter introduces a series of simple yet powerful Iterated Greedy (IG) methods. These include variations of the destruction and reconstruction operators, coordination with several novel local search procedures and corresponding speed-ups. The proposed IG methods are carefully calibrated and evaluated using comprehensive computational experiments. The results indicate that small changes in the features of the algorithm have a profound impact on performance. Comparisons against the state-of-the-art approaches for this particular problem result in a strong and statistically significant performance advantage for the proposed IG procedures.

Chapter 7: Conclusions and future work. This chapter summarizes the main contributions of the thesis, presents the academic publications and activities during the PhD, and eventually outlines

future research directions.

CHAPTER 2

Literature review

This chapter first provides a general review of scheduling problems in Section 2.1. Next, in Section 2.2, we introduce readers to the review of related problems in yard terminals. Furthermore, the literature review is extended to yard crane scheduling problems in Section 2.3.

2.1 General review on scheduling problems

Scheduling problems are one of the most important fields in operational research (Hoos and Stützle, 2005). In general, scheduling is a decision-making process of distributing resources (like the yard crane in this thesis) to activities (like container requests in our case).

It has been extensively researched by the academic community. The first application on computer processes dates from over seventy years ago when the first computer was invented in the 1940s (Blazewicz *et al.*, 1997). In addition to computers, scheduling problems have

been applied to other areas like agriculture, industry, transportation, etc. Figure 2.1 illustrates a brief research domain and several classical applications of scheduling problems.

Using the keywords "scheduling problem", we have found 18,930 results in journals on the Web of Science (June 23, 2023). These articles reflect the main research areas on scheduling problems, including job shop scheduling (4,034 results), machine scheduling (3,553), flow shop scheduling (2,947), production scheduling (2,530), project scheduling (2,180), job scheduling (1,406, except job shop scheduling), crane scheduling (929) and so on. Scheduling problems on terminals (mainly with quay crane and yard crane scheduling) are less popular than some production scheduling problems, like job shop scheduling and machine scheduling. It is partially because some scholars have also regarded crane scheduling problems in terminals as some kind of routing problems (Kim and Kim, 1999; Kim and Park, 2004; Vis and Roodbergen, 2009) or integrated them with non-scheduling problems (Kaveshgar and Huynh, 2015; Cao *et al.*, 2010; Galle *et al.*, 2018; Zhou *et al.*, 2020; Hsu *et al.*, 2021).

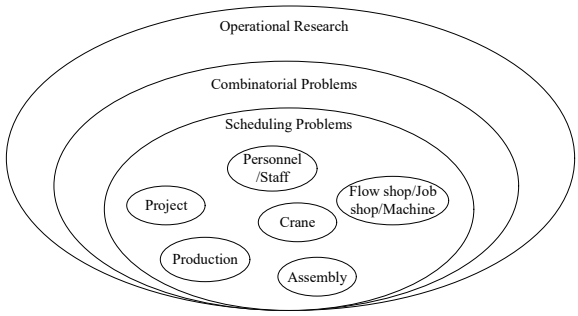


Figure 2.1: Basic research domain and applications of scheduling problems. Elaborated by the author.

2.2 Review on related problems in yard terminals

This section gives a brief literature overview of some related research to the yard crane scheduling problem. A few corresponding problems have already been mentioned in Section 1.2.3 of the operations research on container terminals. Here, we address issues involving or linking the yard crane scheduling problem with container delivery procedures when the container is transferred from/to the terminal port. We conclude that two types of problems are most studied, including quay crane scheduling problems and a series of location adjusting problems.

As mentioned in Section 1.2.3, the quay crane scheduling problem deals with the container loading sequence of berth quay cranes. This happens immediately before or after yard crane operations. The quay crane scheduling problem in the container terminal is first studied in Park and Kim (2003); Ng and Mak (2006). In this problem, the quay crane moves along a common linear rail to handle containers in ship-stacking compartments (bays). Usually, only one quay crane can work on a bay of each vessel at the same time (Ng and Mak, 2006). Generally, this problem aims to minimize the total vessels' stay time in the berth of the port terminal. Gradually, interests from the academic community turn to more complex quay crane settings, hybrid problems, and efficient solving methods. For more realistic quay cranes problems, specific settings of scheduling problems are included, from details of quay cranes to container requests. To name a few, interference of different quay cranes (Lee *et al.*, 2008; Bierwirth and Meisel, 2009), ready/due times or time windows (Legato *et al.*, 2012), and quay crane adjusting/repositioning (Al-Dhaheri *et al.*, 2016). Other extensions include related decisions, like berth scheduling (Han *et al.*, 2010), quay berth assignment/allocation (Liang *et al.*, 2009; Meisel and Bierwirth, 2013), quay crane assignment/allocation (Diabat and

Theodorou, 2014; Fu *et al.*, 2014), and truck scheduling (Kaveshgar and Huynh, 2015). Moreover, plenty of works commit to solving the problem efficiently using various techniques, from exact methods to heuristics or metaheuristics. Moccia *et al.* (2006) have developed a branch-and-cut to solve a small scale problem within a maximum of 25 tasks and 3 quay cranes. Later, more algorithms are designed for high efficiency, to name a few, tabu search heuristic (Sammarra *et al.*, 2007; Zeng *et al.*, 2011; Lee *et al.*, 2011a; Dik and Kozan, 2017), particle swarm optimization (PSO) (He *et al.*, 2015b), genetic algorithm (Lee *et al.*, 2008; Tavakkoli-Moghaddam *et al.*, 2009; Han *et al.*, 2010; Kaveshgar *et al.*, 2012; Chung and Chan, 2013; He *et al.*, 2015b; Wu and Ma, 2017; Abou Kasm and Diabat, 2019; Ma *et al.*, 2021). Finally, more detailed pieces of literature are suggested by Bierwirth and Meisel (2010, 2015); Kizilay and Eliiyi (2021)

Location adjusting is very common in container transport, as well as in yard crane loading procedures. E.g., a container can not be handled directly from the current location if it is stacked under other containers. These problems determine the location adjustment after stacking in the yard with the consideration of future operations before the next loading in advance (Lee and Hsu, 2007). As mentioned in Section 1.2.3, a series of location adjusting problems in the terminal yard has been classified into three problems (Lehnfeld and Knust, 2014), premarshalling, re-marshalling, and relocation problems. All of these problems concern the possible inefficiency to minimize the future loading time of the involved ships. The adjustment of locations for containers can be regarded as an independent operation (premarshalling or re-marshalling) before the real delivery to other transports, like vehicles, trucks and ships. Otherwise, it can be included as one integrated procedure within the real delivery (relocation). Differently, the remarshalling problem reshuffles containers in the container yard, while premarshalling operations happen only inside the single bay, and the relocation problem adjusts locations in a

fast way to retrieve the container when containers have already arrived Tanaka and Takii (2015). In general, all three problems belong to post-stacking problems (Caserta *et al.*, 2020) for a better location plan after being stacked in the block. More detailed surveys can be found in Caserta *et al.* (2011b), Lehnfeld and Knust (2014), or Caserta *et al.* (2020).

2.3 Review on yard crane scheduling problems

Issues unique to yard crane operations in the terminal are receiving more attention from the academic community. Researchers have committed to the yard crane scheduling problem for optimizing the loading and unloading process for a series of container requests in the yard terminal. Table 2.1 lists a total of 27 papers which are mostly involved in the yard crane scheduling problem. Considering the variety of settings of these works, papers are identified with different properties, including port types, the number of cranes, objective functions, classification of requests (S/R), cooperation with requirements, consideration of reshuffling (Reshf.), I/O points (I/O), location assignments and solution methods. Some of the main characteristics are reviewed for a deeper understanding of current research on yard crane scheduling problems in the next section.

2.3.1 Basic settings of yard crane scheduling problems

In recent years, more and more studies have regarded Automatic Container Terminal (ACT) systems, including automatic cranes and vehicles, as the basic terminal configuration. To name a few, Gharehgozli *et al.* (2014, 2015); Hu *et al.* (2016); Galle *et al.* (2018); Zheng *et al.* (2018);

Table 2.1: Literature review on the yard crane scheduling problem with different problem settings.

Literature	Port	Crane	Objective function	S/R*	Cooperation	Reshf.*	I/O*	Loc.*	Approaches **
Lee and Schaefer (1997)	Automated	Single	total travel time	Yes	No	No	No	No	Hungarian method
Ng and Mak (2005a)	Any	Single	job waiting time	No	No	No	No	No	Lower bounding algorithm
Ng (2005)	Any	Multiple	job waiting time	No	No	No	No	No	Dynamic programming
Ng and Mak (2005b)	Any	Single	job waiting time	No	No	No	No	No	B&B
Li *et al.* (2009)	Any	Multiple	job waiting time	Yes	No	No	No	No	Heuristics and rolling-horizon
Vis and Roodbergen (2009)	Any	Single	total travel time	Yes	No	No	No	No	Dynamic programming
Cao *et al.* (2010)	Any	Multiple	makespan	No	Yard truck	No	No	No	Benders' decomposition
Dorndorf and Schneider (2010)	Automated	Multiple	job waiting time	No	No	Yes	No	No	Simulation-based scheduling strategy
He *et al.* (2010)	Any	Multiple	lateness/ total time	No	No	No	No	No	Parallel GA
Vis and Carlo (2010)	Automated	Multiple	makespan	Yes	No	Yes	No	No	Simulated Annealing
Chang *et al.* (2011)	Any	Multiple	lateness	No	No	No	No	No	Rolling-horizon decision strategy
Chen and Langevin (2011)	Any	Multiple	makespan	No	No	No	No	No	GA
Li *et al.* (2012)	Any	Multiple	combined lateness	Yes	No	Yes	No	No	Rolling-horizon algorithm
Gharehgozli *et al.* (2014)	Automated	Single	total travel time	Yes	No	No	Yes	No	Hungarian method + B&B
Liang *et al.* (2014)	Any	Multiple	$Min.$ total travel time and maximization distribution	No	No	No	No	No	GA
Gharehgozli *et al.* (2015)	Automated	Multiple	makespan	Yes	No	No	Yes	No	Neighborhood search heuristic
Wu *et al.* (2015)	Any	Multiple	combined lateness	Yes	No	No	No	No	Clustering-reassigning approach
Hu *et al.* (2016)	Automated	Multiple	makespan	Yes	No	No	No	No	Greedy insertion strategy
Galle *et al.* (2018)	Automated	Single	weighted cost	Yes	No	Yes	No	Yes	Heuristic local search
Zheng *et al.* (2018)	Automated	Multiple	maximum tardiness	Yes	No	Yes	No	No	Heuristic and GA
Chu *et al.* (2019)	Any	Multiple	lateness: flow time	No	No	Yes	No	No	Heuristic and and GA
Gharehgozli *et al.* (2019)	Automated	Single	total travel time	Yes	No	No	Yes	Yes	Three-phase method and heuristic
Zheng *et al.* (2019)	Any	Single	total tardiness	Yes	No	No	No	No	Approximation approach, GA and a rule based heuristic
Yang and Jiang (2020)	Automated	Multiple	total waiting time	No	No	No	No	No	Hybrid ant colony with GA
Zhou *et al.* (2020)	Any	Single	makespan	No	Vehicle positioning	No	No	No	Tabu search and rule besed heuristic
Hsu *et al.* (2021)	Any	Multiple	makespan	No	Yard trucks	No	No	No	Hybrid GA with (subgroups) PSO
Vallada *et al.* (2023)	Automated	Single	weighted cost	Yes	No	Yes	Yes	No	Rule based heuristics, local search

*S/R: differentiated Storage/Retrieval requests. Reshuf.: consider container reshuffling/relocation. I/O: assignment of Input/Output points. Loc.: allocate container positions.
** B&B: Branch and Bound. GA: Genetic Algorithm. PSO: Particle Swarm Optimization.

Gharehgozli *et al.* (2019); Yang and Jiang (2020); Vallada *et al.* (2023). ATC systems are promising to decrease not only manual operations but also to reduce unproductively moves in practice. Comparisons related to automated terminals are referred to Yang *et al.* (2004) or Zhen *et al.* (2011).

In terms of the objective function, minimizing the total travel time is the most popular objective function for the scheduling problem of a single yard crane. The total travel time concerns the sum of time needed to load all requests of containers (Lee and Schaefer, 1997; Vis and Roodbergen, 2009; Gharehgozli *et al.*, 2014; Liang *et al.*, 2014; Gharehgozli *et al.*, 2019). In essence, the total travel time of the single machine can be equal to the makespan, which is mostly adopted for multiple yard cranes scheduling problems. The makespan of the YSCP is to minimize the largest completion time of all container requests (Cao *et al.*, 2010; Vis and Carlo, 2010; Chen and Langevin, 2011; Gharehgozli *et al.*, 2015; Hu *et al.*, 2016; Zhou *et al.*, 2020; Hsu *et al.*, 2021).

The classification of requests in the container terminal is crucial since the locations that the crane has to pass, are determined by the container's properties, like inbound/outbound or seaside/landside. Researchers have classified container requests into storage/inbound and retrieval/outbound requests (Lee and Schaefer, 1997; Li *et al.*, 2009; Vis and Roodbergen, 2009; Vis and Carlo, 2010; Li *et al.*, 2012; Gharehgozli *et al.*, 2014; Wu *et al.*, 2015; Hu *et al.*, 2016; Galle *et al.*, 2018; Zheng *et al.*, 2018; Gharehgozli *et al.*, 2019; Zheng *et al.*, 2019; Vallada *et al.*, 2023). A detailed classification of container requests is shown in Section 3.2. However, some researchers seldom differentiate the storage and retrieval requests. Instead, they regard all requests as homogeneous, and the only difference comes from task times. To name a few, Cao *et al.* (2010); Dorndorf and Schneider (2010); He *et al.* (2010); Chang *et al.* (2011); Caserta *et al.* (2011a); Liang *et al.* (2014); Chu *et al.* (2019); Yang and Jiang (2020); Zhou *et al.* (2020); Hsu *et al.* (2021).

2.3.2 Multiple and combined yard crane scheduling problems

The scheduling problems on the single crane are always compact and put realistic and detailed concerns on the container delivery process, thus explaining the popularity of extending them to multiple cranes or combined versions.

- Multiple cranes

  Scheduling problems of two cranes, dual cranes, or twin cranes, are well investigated in the literature. Commonly, dual cranes are passable with interference rules for the inter-crane and deliver all requests to both sides of blocks (Ng and Mak, 2005b; Li *et al.*, 2009; Vis and Carlo, 2010; Li *et al.*, 2012; Zheng *et al.*, 2018). However, twin cranes can not cross each other and maintain one side of a block separately, e.g., Gharehgozli *et al.* (2015); Hu *et al.* (2016); Gharehgozli *et al.* (2017); Saini *et al.* (2017). Variants like triple cranes (Dorndorf and Schneider, 2010), or cranes passing adjacent blocks (Chen and Langevin, 2011; Chu *et al.*, 2019). More details of multiple crane scheduling problems are given in Speer and Fischer (2017).

- Combined problems

  Another trend is to combine the yard crane scheduling problem with operational problems of relevant equipment, e.g., the quay crane, trucks, and vehicles in the yard terminal. A few representatives are integrated with yard trucks by Cao *et al.* (2010); Hsu *et al.* (2021), with bay allocation by Lee *et al.* (2011b), with quay and truck scheduling by Kaveshgar and Huynh (2015), with container positioning by Galle *et al.* (2018), and with vehicle positioning by Zhou *et al.* (2020).

2.3.3 Single yard crane scheduling problems

Unlike multiple or combined versions, scheduling problems of a single yard crane concern more loading details on the container delivery process. For instance, types of container requests, ready times before loading, assignment of Input/Output points and so on. Some representative studies (Ng and Mak, 2005a,b; Vis and Roodbergen, 2009; Gharehgozli *et al.*, 2014; Galle *et al.*, 2018; Zheng *et al.*, 2019; Vallada *et al.*, 2023) are reviewed.

Ng and Mak (2005a) and Ng and Mak (2005b) have studied the single YCSP on terminal ports with ready times. They formulate the YSCP as a Mixed Integer Programming (MIP) with the crane routing model of Kim and Kim (1999) where the $\mathcal{NP}$-completeness has been shown by Narasimhan and Palekar (2002). However, as we mentioned in Section 2.3.1, Ng and Mak (2005a) and Ng and Mak (2005b) have simplified problem settings without distinguishing the type of requests. In this way, the travel cost of requests and the ready times of container requests are given and fixed. A Branch-and-Bound (B&B) approach (Ng and Mak, 2005b) and heuristic methods (Ng and Mak, 2005a) are proposed to minimize the total waiting time following their basic settings of the problem.

Vis and Roodbergen (2009) have investigated this scheduling problem further specific to a series of storage and retrieval requests, in which the locations of the destination and origin of each request are given before loading or discharging. They devote to minimising the total travelling time of the single crane. Therefore, the problem is then formulated as a kind of Steiner Traveling Salesman Problem (TSP), and a Dynamic Programming method is employed to obtain exact solutions. Whereas Vis and Roodbergen (2009)'s problem is only applicable to a special application where the straddle carrier is employed as the crane and must work within a single row of the block, making it impractical for cross-row cranes, like RTG and RMG shown in Section 1.2.1, which

are employed in most of the modern terminals.

Scheduling of a single yard crane concerning I/O points is addressed in Gharehgozli *et al.* (2014), where I/O points are located at the end of blocks and employed for receiving and sending containers between the yard block and outside. Every container must be allocated to an I/O point, and the best one is given by the shortest distance (Gharehgozli *et al.*, 2014). In this way, the route, as well as the time cost, between requests, can be fixed, making it suitable for a series of TSP models (Gharehgozli *et al.*, 2015, 2019). Multiple-stage approaches are proposed (Gharehgozli *et al.*, 2014) with the Assignment Problem (AP) solution from the Hungarian algorithm (Kuhn, 1955). Next, the optimality has been proved for this TSP problem with subtours-merging techniques, i.e., arcs swapping and the B&B algorithm in the following stages. However, some assumptions in their settings can be too ideal to be realistic. E.g., all containers are always available in the picking-up positions for the next loading. Moreover, the I/O point can be used with zero time cost whenever one I/O point is needed.

The single YCSP is also investigated by Galle *et al.* (2018) with concerns about relocation in retrieval loading. The relocation involves adjusting the stacking locations of containers before retrieving a container which is located under other containers. Galle *et al.* (2018) integrate the relocation into a yard crane scheduling problem and formulate it with binary integer programming. Solutions are obtained from linear programming relaxation and local search heuristics. Since relocation operations require stacking states quite close to real-time (one minute in Galle *et al.*, 2018) for decision-making, Galle *et al.* (2018)'s algorithms are suitable only for small instances. Typically, the number of requests ranges only from 1 to 15 containers (Galle *et al.*, 2018).

Zheng *et al.* (2019) introduced a single YCSP with uncertainty. It assumes that the release time to retrieve containers from the yard

block can be uncertain in different scenarios, and the expectation of total tardiness for different scenarios should be minimized as the objective function. They formulate this YCSP as a two-stage stochastic programming. In the first stage, Zheng *et al.* (2019) decide the loading sequence for only storage requests without information about release times of retrieval requests. Next, decisions with full information are made for the final operation sequence of all requests involved. Note that the proceeding time for every request is constant and equal in their paper, which is a strong assumption for real container terminals.

All the above literature deals with the scheduling of container requests as an independent procedure of the yard crane. Nevertheless, the scheduling on precedent and subsequent requests in real terminals requires the coordination of the crane, I/O points and subsystems, like vehicles, trucks, trains or vessels. To make it closer to real terminals, Vallada *et al.* (2023) have proposed two realistic MIP formulations for the single crane scheduling with consideration of the release of containers and occupation on I/O points. In this case, the container request goes with a time parameter, which is related to the release or due time of the request and is decided at an upper operational level. In collaborating with this time parameter, the I/O point can be used and released in the delivery procedure of the container to meet the constraints on usage intervals of the crane and I/O points. To the best of our knowledge, this is the most realistic formulation of the YCSP with a single yard crane. However, the problem is very complex to solve, where the MIP solver CPLEX can only obtain optimal scheduling sequences with up to 10 containers. For this reason, rule-based heuristics and local search methods are developed in Vallada *et al.* (2023).

2.3.4 Solution methods for yard crane scheduling problems

- Exact methods

Plenty of methods have proven to be efficient in solving problems for different yard configurations in the past decades. Dynamic programming, is employed by Vis and Roodbergen (2009) with a particular one-lane crane with the asymmetric Steiner Traveling Salesman Problem (TSP) model. Furthermore, a multi-stage exact approach is used by Gharehgozli *et al.* (2014) and Gharehgozli *et al.* (2019). They formulated the YCSP as a TSP and then convert it into an Assignment Problem (AP) solved by the Hungarian algorithm. Subtour merging techniques and the Branch and Bound (B&B) algorithms are adopted in the next stage for the subtour merging in the following steps. The above methods are demonstrated to be efficient for small or medium size instances. However, they are incapable of more complex settings or larger-scale instances at reasonable times.

- Heuristic methods

  Many heuristics and metaheuristics are often used for more complex versions, like multiple cranes or combined problems. First of all, the most popular proposal for the YCSP is the Genetic Algorithm, as in He *et al.* (2010), Chang *et al.* (2011), Chen and Langevin (2011), Liang *et al.* (2014), Hu *et al.* (2016), Zheng *et al.* (2018), Chu *et al.* (2019), and Yang and Jiang (2020). Metaheuristics have been used for to the YCSP in the case of multiple cranes, including the Simulated Annealing method proposed by Vis and Roodbergen (2009), Tabu Search developed by Chen and Langevin (2011), and Particle Swarm Optimization (PSO) by Hsu *et al.* (2021). Some kinds of specialized methods are also designed. Cao *et al.* (2010) tackled the combined YCSP with yard truck using the Benders' cut-based method. Gharehgozli *et al.* (2015) developed an adaptive large neighborhood search heuristic to obtain near-optimal solutions for small instances.

In conclusion, the YCSP has been investigated well, and many

solution methods have been proposed for multiple crane versions and combined problems. However, exact approaches are hard to apply to the studied problem in this thesis. Dynamic programming (Vis and Roodbergen, 2009) requires strict settings on single decomposed rows of the block, where the crane has to reach the row end for row changing. Although Gharehgozli *et al.* (2014) composes the assignment of I/O points as an essential part of the YCSP, their methods are only capable of fixed I/O points and do not consider the delay and congestion caused in a realistic situation. Previous experiments in Vallada *et al.* (2023) have shown that the problem is extremely complex, where the commercial solver CPLEX is barely competent to offer exact solutions for instances with more than 10 containers. Metaheuristic solution methods for the yard crane scheduling problems are alternatives.

CHAPTER 3

The yard crane scheduling problem

This chapter introduces the yard crane scheduling problem investigated in this thesis. We first describe the basic configuration of container yards and some assumptions used for this problem in Section 3.1. Next, Section 3.2 introduces the container requests in terminal yards. The decisions and objectives are introduced in Section 3.3. All the variables, sets and parameters are introduced in Section 3.5 and the mathematical model in this thesis is presented in Section 3.6.

3.1 Configurations and assumptions for the terminal yard

Scheduling problems related to yard cranes (YCs) focus on arranging the loading operations of YCs for containers handled in port terminals. In this thesis, we address a realistic scheduling problem that involves determining the optimal loading operations for storing and retrieving containers between the yard block and either the seaside or the landside.

Throughout this thesis, we consider a European yard terminal, whose general layout is depicted in Figure 3.1. The terminal is equipped with Quay Cranes (QCs), YCs, Automated Guided Vehicles (AGVs), Straddle Carriers (SCs), and Input/Output (I/O) points. Containers are transported to different destinations depending on their type, using vehicles like AGVs and trucks. The YC operates within a single yard block (illustrated in Figure 3.2), where containers are organized in rows, bays, and tiers.

Storage (inbound) containers are delivered from outside the terminal to either the landside or seaside and are stored in yard blocks, while retrieval (outbound) containers, already stacked in yard blocks, are transported to the landside or seaside. The YCs are responsible for loading all containers, both inbound and outbound, between the yard blocks and the corresponding I/O points based on their final destinations.

I/O points serve as transfer locations where containers are moved between YCs and vehicles like AGVs or trucks. By decoupling the operations of YCs from vehicles, I/O points allow both to operate independently allowing connections between multiple transport forms. However, the cooperation between YCs and I/O points has to synchronize directly with containers w.r.t. their time parameters, like the release time or vehicle's arrival time. Proper assignment of I/O points for the delivery of containers by the YC will greatly reduce downtime and balance the loading flow in the yard terminal.

We assume that the studied problem meets the following hypothesis, although they are common and frequently adopted in the literature detailed in Chapter 2.

- *Single crane*. A single yard crane is considered for each yard block in the terminal. The crane can be of any type, e.g., RMG or TRG, and no interference exists for this reason.

- *No-break*. We assume that there is no interruption or delay in a

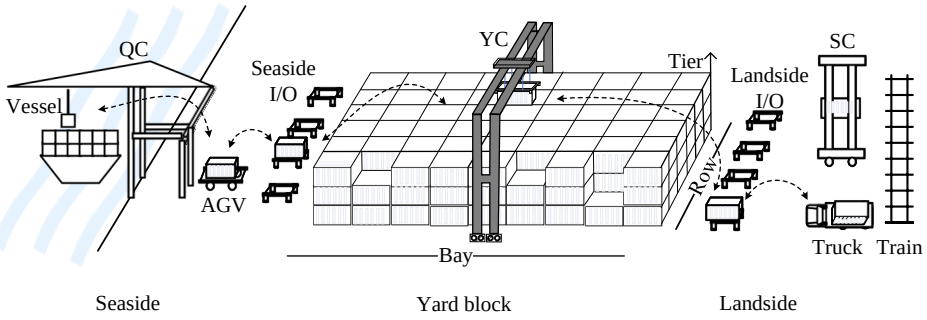


Figure 3.1: Container transfer process involving different equipment like a quay crane (QC), Automated Guided Vehicles (AGVs), a yard crane (YC) and straddle carrier (SC) in a typical container terminal with a European layout. Elaborated by the author.

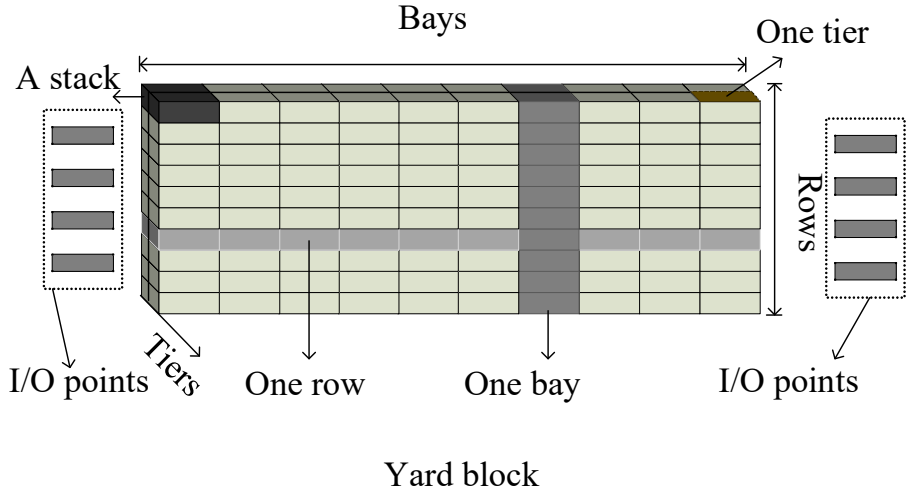


Figure 3.2: Basic nomenclature in a yard block. Elaborated by the author.

loading operation, i.e. a container will complete a fully loaded movement from the initial position to its destination without interruption once it is delivered by the yard crane.

- *Single block*. A single block in the container yard is considered. In European terminal layouts, the bay typically extends farther than the row, making it time-consuming for cranes to move between

blocks in the bay direction. Additionally, due to the limited span of yard cranes (20-70 meters, as discussed in Section 1.2.1), conveying containers among multiple blocks can be impractical.

- *Fixed reshuffling*. Regarding reshuffling or relocation of block location, we suppose that for each container request, the possible moves of reshuffling or relocation are already known where the time cost of every move proceeded by the crane is fixed.

- *Full information assumption*. We assume that all information about the container request is already known before the scheduling. The information includes but is not limited to, container types or positions of initial location and destination.

Under these assumptions, a single YC has to complete all container requests in the yard block. Figure 3.3 illustrates the handling process for two consecutive containers. Each container loading operation consists of two phases: an empty move and a loaded move. The empty move begins with the crane hoisting its spreader, followed by the crane traversing the yard block. Once positioned, the crane lowers the spreader to pick up the container, initiating the loaded move. The container is then transported across the yard block and unloaded to its destination at the end of this request. Note that the crane will go directly to the next container request without any breaks or unexpected delays, shown as the *No-break* assumption. Additionally, any potential reshuffling, accounted for under the *Fixed reshuffling* assumption, is incorporated into the container pickup process. The origin and destination points for both the empty and loaded moves depend on the type of container, as classified in Section 3.2.

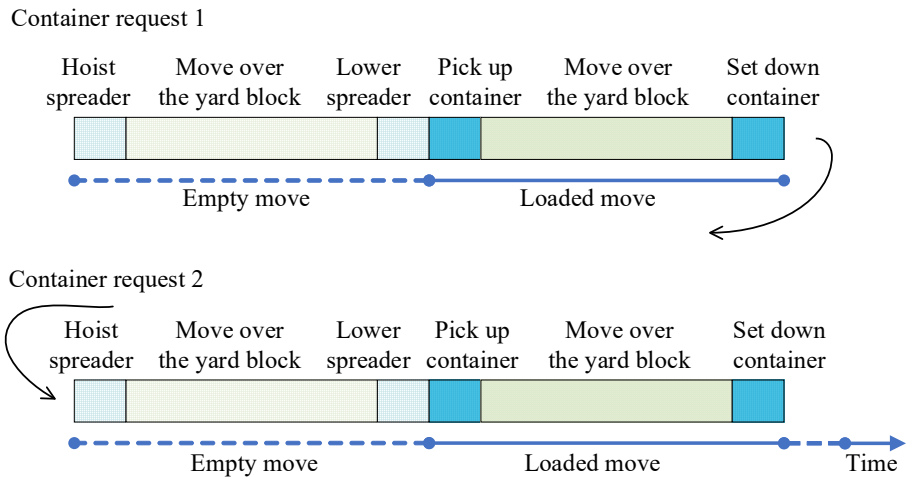


Figure 3.3: Crane movements for two consecutive container requests. Elaborated by the author.

3.2 Types of container in the yard block

A detailed classification of container requests is crucial for distinguishing the varying characteristics and providing a more accurate representation of real operations at yard terminals. However, previous studies have rarely differentiated container request types (Lee and Schaefer, 1997; Ng and Mak, 2005a,b; Ng, 2005; Cao *et al.*, 2010; Dorndorf and Schneider, 2010; He *et al.*, 2010; Chang *et al.*, 2011; Caserta *et al.*, 2011a; Liang *et al.*, 2014; Chu *et al.*, 2019; Yang and Jiang, 2020; Zhou *et al.*, 2020; Hsu *et al.*, 2021). In these works, all tasks only differentiate themselves from others only by their task times. Consequently, crane scheduling is often modelled as a job or machine scheduling problem, where the cost of requests is assumed to be fixed. The simplification is known as the (*constant-time*) assumption. Apparently, this assumption is overly theoretical and does not accurately reflect real-world yard terminals, where time costs are significantly influenced by the distance between the origins and destinations of containers.

According to the different locations of origins and destinations,

container requests in the yard terminal can be divided into Storage/Inbound requests and Retrieval/Outbound requests, as follows:

- Storage/Inbound requests $\mathcal{S}$: requests which require a container c to be loaded from the origin to a destination in the yard block, which is known before. These containers will be loaded from the seaside I/O points (O^S) if they come from the seaside by ships, or to the landside I/O points (O^L) if the containers are delivered into the yard block by trains or trucks. All these containers come from the outside of the yard block, and the I/O points on the corresponding side need to be assigned.

- Retrieval/Outbound requests $\mathcal{R}$: requests which require a container c to be retrieved from the current block position (its origin) to the outside by I/O points (destination). Similar to the storage request, the I/O points should be assigned during the loading operations. The containers had been stored in the block before the task begins. Therefore, their block location (retrieval locations) are already known. The *Fixed reshuffling* in Section 3.1, applies for any reshuffling operation required to relocate the container on the top stack.

Figure 3.4 illustrates the full procedure of archiving the storage (left) and retrieval (right) requests. Basically, the request is launched with an empty move consisting of hosting the spreader, moving over the yard block, and lowering the spreader as shown in Figure 3.3. Next, the crane picks the container up, moves the container over the yard block, and sets the container down to the destination of the container. Each request consists of a container that must be loaded between the yard block and the I/O point. Based on the delivery route from the initial position to the destination, storage and retrieval requests are categorized into four types of containers, C_1, C_2, C_3 and C_4.

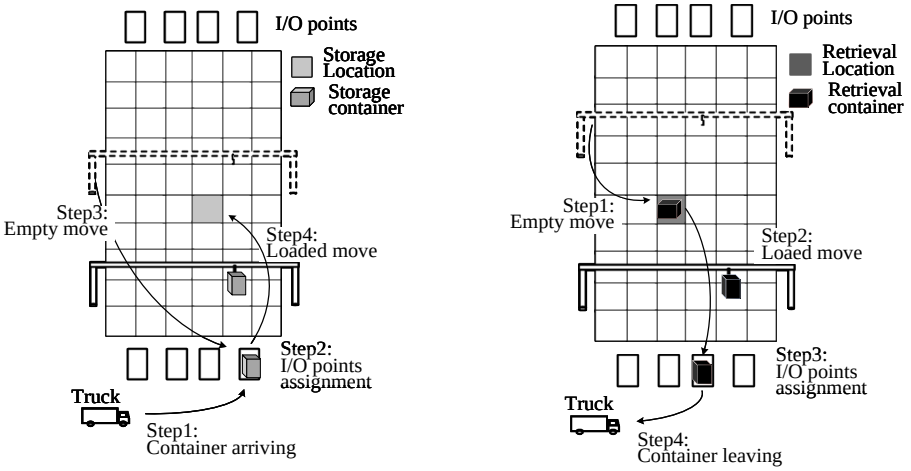


Figure 3.4: Request steps for the storage container (left) and retrieval container (right). Elaborated by the author.

- *Sea to Yard*, $c \in C_1$, the storage container c has been unloaded from ships and handled to the seaside by vehicles, like AGVs or trucks, by a known release time r_c of this container. After r_c, the container will be loaded by the yard crane from the seaside to a known block position via the I/O point on this side. A seaside I/O point needs to be assigned.

- *Land to Yard*, $c \in C_2$, the storage container c has been unloaded from trains or outside trucks and sent to the landside of the yard block by a known release time r_c of this container. After r_c, the yard crane transfers the container from the assigned landside I/O point to a known block position.

- *Yard to Sea*, $c \in C_3$, the retrieval container c has been placed at the yard block in previous movements and is now ready to be retrieved out of the block to the seaside I/O point. The move is carried out by the yard crane with a known due time d_c at which the container should be delivered to the assigned seaside I/O point on time.

- *Yard to Land*, $c \in C_4$, the retrieval container c has been placed at the block position and needs to be sent to the landside. The yard crane will complete the movement from the block to the landside I/O point. After that, the container will be transferred to the vehicle in the assigned I/O point after the vehicle ready time e_c, which is known as well.

In general, we study how to sequence n storage and retrieval requests in the set $C = \bigcup_{i=1}^{4} C_i$ with m I/O points in a single yard block of R rows, B bays and T tiers (a block is shown in Figure 3.2). The single crane is placed at a known initial block position Ini. Moreover, the crane is able to start with any type of container and will terminate at the destination of the last request.

Every storage or retrieval request corresponds to a unique container $c \in C$. Their types (C_1 to C_4) as well as time parameters, release time (r_c for $c \in C_1 \cup C_2$) of storage requests, due time (d_c for $c \in C_3$) and vehicle arriving time (e_c for $c \in C_4$) of retrieval requests, are usually arranged by the operator of yard terminal. For this reason, all the time parameters of containers are known in advance. Besides, locations of storage requests in the block are always given with current free block locations which are always determined by terminal operators before the requests, thus to be also known. Eventually, all the information of container requests meets the *full information assumption* in Section 3.1.

Each container request corresponds to a single loading operation by the yard crane with corresponding origin and destination. As shown in Figure 3.4, the empty move for container c begins at either the initial position Ini of the crane or the destination of the previous container, during which the crane moves with an empty trolley over the yard block. The empty move ends at the origin of the container, where the crane loads the container. Each request involves both an origin and a destination, requiring the crane to transport the container from the

origin to the destination in a loaded move (as illustrated in Figure 3.4).

In pursuit of clarity, the following notations are used interchangeably. In the case of a storage request, container c is delivered to the I/O point (origin) and waits to be stored at block location l_c (destination). In a retrieval request, container c is located at block location l_c (origin) and is to be transported to the I/O point (destination). Either the seaside or landside I/O point can serve as the origin for storage containers or the destination for retrieval containers. Table 3.1 summarizes key information about the different types of containers.

Table 3.1: The I/O and location sets for all types of containers.

Set	Type	Time parameter	I/O Set (O_c)	Origin	Destination
$\mathcal{S}$	C_1	r_c	O^S	Seaside I/O point	l_c
	C_2	r_c	O^L	Landside I/O point	l_c
$\mathcal{R}$	C_3	d_c	O^S	l_c	Seaside I/O point
	C_4	e_c	O^L	l_c	Landside I/O point

Moreover, we note all locations on the block as integer Cartesian coordinates (x, y, z), including the initial crane location Ini, block locations, and locations of I/O points. Where x, y and z represent the numbers of the row, the bay and the tier of the block, respectively. The crane begins at location Ini of any row and bay ($0 < x \leq R$, $0 < y \leq B$), even over the top tier ($0 < x \leq T+1$) of the block. For all block positions, $0 < x \leq R$, $0 < y \leq B$ and $0 < z \leq T$. However, since all I/O points are located at the edges of the block, their bay number y are 0 or $R+1$ for I/O points in the seaside or landside, respectively. Using the coordinate, the distance between any two locations i and j is defined as:

$$t_{ij} = |T+1-z_i| + \max\{|x_i - x_j|, |y_i - y_j|\} + |T+1-z_j| + Res_{time} \quad (3.1)$$

where Res_{time} is a constant reshuffling (refer to the *full information* in Section 3.1) for the container to be retrieved from location i, located in the middle tier of this stack under other containers. Since the

gantry moves on bays and its trolley moves on rows simultaneously, the distance counts only the maximum movement of the crane in one direction, also named the Chebyshev distance. The Chebyshev distance is also popular in Gharehgozli *et al.* (2014, 2019); Vallada *et al.* (2023). Moreover, the distance between two locations can be equivalent to the time cost of Galle *et al.* (2018) as the speeds of the crane in the gantry, trolley, and hoisting can be 1, without loss of generality, if there is no crane interference (Speer and Fischer, 2017).

3.3 Decision making and objective formulation for the yard crane scheduling problem

Two decisions must be made during the loading operation of the yard crane. One is the loading sequence of containers that should be achieved by the yard crane, the other is the I/O point that the yard crane has to pass by to receive and send containers. As we mentioned in Section 3.1, the crane loading process must cooperate with subsystems, like trains, trucks, vehicles and ships, on either container stacking or retrieving. To connect the crane with these subsystems, as shown in Figure 3.4, the I/O points perform like a buffer zone to receive the storage container and send out the retrieval container between the yard block and outside. Due to the different locations of I/O points, it is easy to know that the efficiency of crane loading can be affected by the I/O assignments. Therefore, the assignment on I/O points has become a necessary decision for each container in the scheduling of the yard crane (Gharehgozli *et al.*, 2014, 2019; Vallada *et al.*, 2023).

The I/O points are critical resources during the loading operations of the yard crane, yet the number of I/O points on both the seaside and landside is limited. Each I/O point can only accommodate one

container at a time. For any container c, the usage period of an I/O point begins at time SO_c, when the container is placed, and ends at time FO_c, when the container has left. The utilization of I/O points is closely linked to the operational states of the crane and related subsystems, such as trains, trucks, vehicles, and ships. Since the yard crane and subsystems operate almost in parallel, the start time SC_c for loading c by the crane requires two conditions to be satisfied:

1. It is the turn for the container c where the yard crane has arrived at the origin of c;

2. The container has been released at a free I/O point when c belongs to sets of storage/inbound containers.

Similarly, the crane loading ends at FC_c when the container has been put at the destination. If the destination is the I/O point for retrieval/outbound containers, the I/O should be free when the crane comes. Consequently, containers may not be loaded or completed on time due to a shortage of available I/O points, leading to inefficiencies in crane movements and I/O point utilization during the yard crane scheduling process.

Several possible inefficiencies in the yard crane scheduling are accounted for during the loading process for storage and retrieval requests, including delays, congestion, and earliness for each corresponding container. Delays occur when there is a gap between the actual completion time of a container request and its respective time parameter. Specifically, this involves the release time r_c for containers $c \in C_1 \cup C_2$, the due time d_c for containers $c \in C_3$, and the vehicle arrival time e_c for containers $c \in C_4$. For storage containers ($c \in C_1 \cup C_2$), the request is considered complete at the crane finishing time FC_c, when the crane has unloaded the container to its block location. For retrieval containers ($c \in C_3 \cup C_4$), which are transferred to an I/O point, the job is completed exactly at the I/O finishing time FO_c. Therefore, delays are

defined as $FC_c - r_c$, $FO_c - d_c$, and $FO_c - e_c$ for the respective types of containers, where r_c, d_c, and e_c are the corresponding time parameters for each container.

Congestion may be triggered when no I/O points are available upon the release or arrival of the container, given that the limited number of I/O points can only hold one container at a time. For a storage container $c \in \mathcal{S}$, if no I/O points are available at the release time r_c, the container will block either the seaside ($c \in C_1$) or landside ($c \in C_2$) until an I/O point becomes available at time SO_c. In this case, the congestion for the storage container is measured as $SO_c - r_c$. A similar situation arises for retrieval containers $c \in \mathcal{R}$. For a landside retrieval container ($c \in C_4$), congestion, quantified as $SO_c - e_c$, occurs if no I/O points are free when the vehicle arrives at the landside at time e_c. In both cases, subsystems such as trains, trucks, and other vehicles must wait outside the I/O point of the yard block.

Of special concern is the earliness E_c of retrieval containers ($c \in C_3$) at the seaside. Due to the restricted berthing time of ships, containers shipping to the I/O points on the seaside are usually more urgent than those on the landside. For this reason, the retrieval loading to the seaside is to be executed as soon as the crane comes with the container. However, the delivery before the container due time d_c may cause chaos for ships, e.g., ships may not have arrived at this time. As the chaos may also lower the efficiency on the seaside, which results in the same outcomes as that on the landside, the earliness can hence be a kind of congestion in this case.

All these inefficiencies in container handling need to be penalized. As shown in expression (3.2), the problem aims to minimize the total weight cost of delays and congestion on the operational level. The very same objective function is first adopted in Vallada *et al.* (2023) and later in Wang *et al.* (2025). Using the objective function, a mathematical model of the yard crane scheduling problem is presented in the next

Section 3.6.

$$\min \quad \sum_{c\in\mathcal{S}} \left(w_c^R(FC_c - r_c) + w_c^C(SO_c - r_c)\right) + \sum_{c\in C_3} \left(w_c^R(FO_c - d_c) + w_c^C E_c\right) + \sum_{c\in C_4} \left(w_c^R(FO_c - e_c) + w_c^C(SO_c - e_c)\right) \tag{3.2}$$

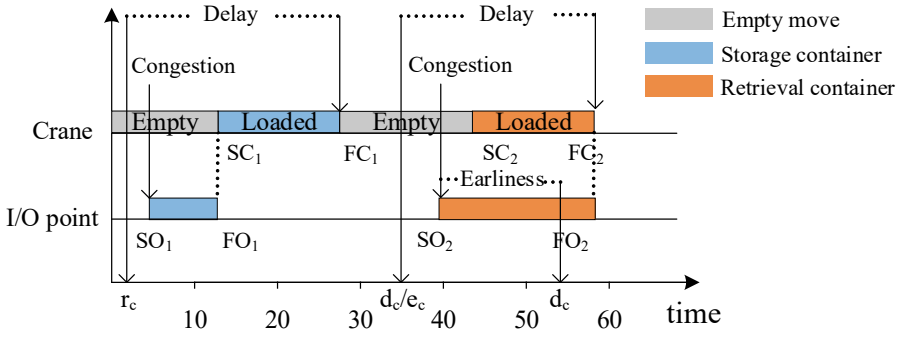


Figure 3.5: Gantt chart with the illustration of the delay, congestion and earliness. Elaborated by the author.

A quick example with one storage and one retrieval container is illustrated with a Gantt chart in Figure 3.6. The yard crane starts from the Ini with an empty move to pick a storage container at its origin O_1, which is an I/O point. The I/O point is already assigned at time SO_1. The crane starts to load this container from O_1, at time SC_1 before which this I/O point has been released at time FO_1. Finally, the storage container is loaded to the destination D_1, a known yard position, at time FC_1. Similarly, the retrieval container is loaded from the origin O_2, its current position at time SC_2 to its destination D_2, i.e., an I/O point at FC_2. After I/O finishing time FO_1 and FO_2, the corresponding I/O points are allowed to be used by other coming containers for the continuing yard crane loading.

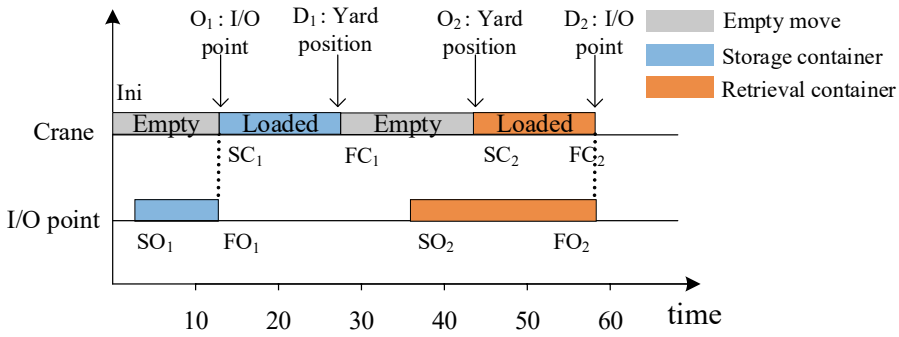


Figure 3.6: Gantt chart of loading operations with loaded and empty moves for the instance with one storage and a storage container where the origins (O_1, O_2), destinations (D_1, D_2) and time usage on the yard crane (SC, FC) and I/O point (SO, FO) are pointed out. Elaborated by the author.

3.4 Nomenclature

3.5 Nomenclature

Sets:

C	set of all containers, including containers $c \in C = \mathcal{R} \cup \mathcal{S}$
$\mathcal{S}$	set of storage containers, including containers $c \in \mathcal{S}, \mathcal{S} = C_1 \cup C_2$
$\mathcal{R}$	set of retrieval containers, including containers $c \in \mathcal{R}, \mathcal{R} = C_3 \cup C_4$
C_1	set of containers that will be loaded from the seaside to the yard block
C_2	set of containers that will be loaded from the landside to the yard block
C_3	set of containers that will be loaded from the yard block to the seaside
C_4	set of containers that will be loaded from the yard block to the landside

K	set of loading sequence, number $k \in K$ indicates the crane loading order
L	block locations set, block location $l_c \in L$ is unique for container c
O^S	set of I/O points on the seaside
O^L	set of I/O points on the landside
O_c	set of all possible I/O points for container c in the corresponding side

Parameters:

n	number of all container requests/containers
m	number of all I/O points
Ini	initial position of the yard crane
R	number of rows in the yard block
B	number of bays in the yard block
T	number of tiers in the yard block
l_c	block location to stack the container c of the storage request or the current location of the retrieval request in the block
r_c	arrival time to I/O points for storage container $c \in \mathcal{S}$
d_c	due date for container $c \in C_3$ when it has to be delivered
e_c	vehicle arriving time for container $c \in C_4$
w_c^R	tardiness weight of container $c \in C$
w_c^C	congestion weight of container $c \in C$
t_{ij}	the time cost between two location i and j including the initial position Ini, and position of containers and I/O points

Variables:

X_{ckj}	1, if container c is processed in order k by assigning I/O point j and block location l; 0, otherwise. $c \in C$, $j \in O_{c'}$, $k \in K$, $l \in L_c$

SO_c	time at which container c starts occupying I/O point
FO_c	time at which container c leave the I/O point
SC_c	time at which the crane starts loading container c
FC_c	time at which the crane finishes loading container c
T_0	time at which the crane leaves the initial location
E_c/T_c	earliness/tardiness of container $c \in C_3$.

3.6 Mathematical model for the yard crane scheduling problem

From the initial work of Ng and Mak (2005a,b), the YCSP has been modelled into different forms. Ng and Mak (2005a) formulated the YCSP as a machine scheduling problem to minimize the total waiting time and also the total time cost in Ng and Mak (2005b) where the time needed to finish a job is always fixed and known in advance. In the case of a single YC (machine), the whole schedule can be seen as the travelling route of the YC. Therefore, YCSPs have often been modelled as routing problems, typically a series of Travelling Salesman Problems (TSP) (Kim and Kim, 1999; Kim and Park, 2004; Vis and Roodbergen, 2009; Gharehgozli *et al.*, 2014, 2015, 2019). In these studies, problems with the loading process are often addressed, where two important ones distinguish the YCSP from previous formulations. Vis and Roodbergen (2009) considered retrieval and storage requests along with the origins and destinations on different sides (sea and land). Gharehgozli *et al.* (2014) introduced the I/O points for the first time to receive and send containers between the yard block and different sides. We regard these as significant improvements in the formulation of, and research into YCSP as they provide a way of bridging the gap between theoretical models and their application in real-world maritime port terminals.

The mathematical formulations of the yard crane scheduling problem with the I/O points assignment have been proposed by Vallada *et al.* (2023). In which two mathematical models, a routing model and a scheduling model, are proposed following the basic assumptions in Section 3.1 and settings in Section 3.3. Basically, both the mathematical models have dealt with the constraints on limited resources of the crane and I/O points, as well as the relationship among time variables and loading constraints for different types of containers. For the sake of intuitiveness, the scheduling model of Vallada *et al.* (2023) is displayed as follows:

Objective function:

$$\min \sum_{c\in C_1\cup C_2}\left(w_c^R(FC_c - r_c) + w_c^C(SO_c - r_c)\right) + \sum_{c\in C_3}\left(w_c^R(FO_c - d_c) + w_c^C E_c\right) + \sum_{c\in C_4}\left(w_c^R(FO_c - e_c) + w_c^C(SO_c - e_c)\right) \tag{3.3}$$

As outlined in Section 3.3, the objective function (3.3) aims to minimize delays, congestion, and earliness across all containers. For storage requests ($c \in C_1 \cup C_2$), these objectives are related to the container's release time d_c, where delays and congestion can occur if the crane does not complete the delivery on time or if the I/O point is not occupied promptly. For retrieval requests at the seaside ($c \in C_3$), delays may arise if the request is completed at the I/O point after the due time d_c.

Resource constraints:

$$\text{s.t.} \quad \sum_{k\in K}\sum_{j\in O_c} X_{ckj} = 1 \qquad \forall c \in C \tag{3.4}$$

$$\sum_{c\in C}\sum_{j\in O_c} X_{ckj} = 1 \qquad \forall k \in K \tag{3.5}$$

Constraint (3.4) ensures that all containers must be assigned with an I/O point j located at a position k in the sequence. For every position

in this sequence, only one container can be located in Constraint (3.5). However, this rule does not apply to I/O points. Instead, the I/O point can be utilized by one more container after finishing the current request.

Time variables relationship:

$$SO_c \leq FO_c \qquad \forall c \in C_1 \cup C_2 \qquad (3.6)$$
$$FO_c = SC_c \qquad \forall c \in C_1 \cup C_2 \qquad (3.7)$$
$$FC_c \leq FO_c \qquad \forall c \in C_3 \cup C_4 \qquad (3.8)$$
$$SO_c = FC_c \qquad \forall c \in C_3 \qquad (3.9)$$
$$SO_c \leq FC_c \qquad \forall c \in C_4 \qquad (3.10)$$
$$SO_c \geq r_c \qquad \forall c \in C_1 \cup C_2 \qquad (3.11)$$
$$FO_c \geq d_c \qquad \forall c \in C_3 \qquad (3.12)$$
$$SO_c \geq e_c \qquad \forall c \in C_4 \qquad (3.13)$$
$$E_c - T_c = t_c - SO_c \qquad \forall c \in C_3 \qquad (3.14)$$

From Constraint (3.6) to (3.14), the logical relationship between different time variables is determined. For storage containers ($c \in C_1 \cup C_2$), the I/O occupation is finished before it was occupied (Constraint 3.6) and at the time of starting yard crane loading (Constraint 3.7). On the other hand, Constraint (3.8) endures that the retrieval container ($c \in C_3 \cup C_4$) has to be loaded by the crane before the I/O finishing time. The I/O occupation for retrieving containers at the seaside starts, based on Constraint (3.9), exactly when the crane has finished loading from the block position. Differently, the I/O occupation should happen before the retrieving movement (Constraint 3.10) for the less emergency on the landside, explained in Section 3.2. Constraint (3.11), Constraint (3.12) and (3.13) define the relationship between time variables of different types of containers and their corresponding time parameters. Besides, the earliness and tardiness of retrieval containers $c \in C_3$ are

defined in Constraint (3.14).

The first crane move:

$$SC_c \geq T_0 + \sum_{j \in O_c} t_{Inij} X_{c1j} \qquad \forall c \in C_1 \cup C_2 \quad (3.15)$$

$$SC_c \geq T_0 + \sum_{j \in O_c} t_{Inil_c} X_{c1j} \qquad \forall c \in C_3 \cup C_4 \quad (3.16)$$

Constraint (3.15) and (3.16) force that the crane must start from the initial location Ini in the first position of request sequence for storage containers ($c \in C_1 \cup C_2$) and retrieval containers ($c \in C_3 \cup C_3$), respectively.

Crane loading constraints:

$$FC_c = SC_c + \sum_{k \in K} \sum_{j \in O_c} t_{jl_c} X_{ckj} \qquad \forall c \in C_1 \cup C_2 \tag{3.17}$$

$$FC_c = SC_c + \sum_{k \in K} \sum_{j \in O_c} t_{l_c j} X_{ckj} \qquad \forall c \in C_3 \cup C_4 \tag{3.18}$$

$$SC_c \geq FC_{c'} + t_{l_{c'} j} - M(2 - \sum_{r \in O_{c'}} X_{c'(k-1)r} - X_{ckj}) \qquad \forall c, c' \in C_1 \cup C_2, \forall k \in K, \forall j \in O_c \tag{3.19}$$

$$SC_c \geq FC_{c'} + t_{jl_c} - M(2 - X_{c'(k-1)j} - \sum_{r \in O_c} X_{ckr}) \qquad \forall c, c' \in C_3 \cup C_4, \forall k \in K, \forall j \in O_{c'} \tag{3.20}$$

$$SC_c \geq FC_{c'} + t_{l_{c'} l_c} - M(2 - \sum_{r \in O_{c'}} X_{c'(k-1)r} - \sum_{r \in O_c} X_{ckr}) \quad \forall c' \in C_1 \cup C_2, \forall c \in C_3 \cup C_4, \forall k \in K, \tag{3.21}$$

$$SC_c \geq FC_{c'} + t_{jr} - M(2 - X_{c'(k-1)j} - X_{ckr}) \qquad \forall c' \in C_3 \cup C_4, \forall c \in C_1 \cup C_2, \forall k \in K, \forall j \in O_{c'}, \forall r \in O_c \tag{3.22}$$

The crane loading Constraints (3.17) to (3.22) reveal the time relations of the crane loaded moves in Figure 3.3. For any type of container, Constraints (3.17) and Constraints (3.18) force that the loaded move goes without delay from the origin to the destination of the request. Next, two loaded moves are connected by empty moves (refer to Figure 3.3) for all combinations between storage and retrieval containers. E.g., Constraints (3.17) guarantees that the next storage loaded move of c proceeds from the block location $l_c^{'}$ of the last storage container c' and the chosen I/O point j for c after finishing the last

loaded move of c'. Similarly, the connections of retrieval requests to retrieval requests, storage requests to retrieval requests and retrieval requests to storage requests are shown in Constraint (3.20) to (3.22), respectively.

I/O conflict constraints:

$$SO_c \geq FO_{c'} - M(2 - \sum_{\tau=1}^{k-1} X_{c'\tau j} - \sum_{\tau=k}^{K} X_{c\tau j}) \qquad \forall c, c' \in C_1 \cup C_3, \forall k \in K, \forall j \in O^S \tag{3.23}$$

$$SO_c \geq FO_{c'} - M(2 - \sum_{\tau=1}^{k-1} X_{c'\tau j} - \sum_{\tau=k}^{K} X_{c\tau j}) \qquad \forall c, c' \in C_2 \cup C_4, \forall k \in K, \forall j \in O^L \tag{3.24}$$

$$X_{ckj} \in \{0, 1\} \qquad \forall c \in C, \forall j \in O_c, \forall k \in K \tag{3.25}$$

$$SO_c, FO_c, SC_c, FC_c, E_c, T_c, T_0 \geq 0 \qquad \forall c \in C. \tag{3.26}$$

Constraints (3.23) and (3.24) ensure that the employment of each I/O point for the next assignment, no matter in the seaside or the landside, should be later than the finishing time of the last use when two containers have employed the same I/O point. Finally, the X_{ckj} is limited to binary, and other involved variables are non-negative in Constraints (3.25) and (3.26).

Distinguishing from previous literature, the above formulation proposed by Vallada *et al.* (2023) has entailed more realistic considerations from real-world terminals. Some differences from prior research are concluded as follows:

- The problem adopts variable job time, which is determined by the chosen location, e.g., the location of the chosen I/O (refer to the distance function (3.1)). However, several previous practices

on yard crane scheduling problems (Ng and Mak, 2005b,a; Cao *et al.*, 2010; He *et al.*, 2010; Chen and Langevin, 2011; Chang *et al.*, 2011) use constant job time, which can be too ideal to be realistic.

- Instead of treating crane loading as a single route, we have detailed the loading operations into two levels: empty and loaded moves of the yard crane. Such a realistic formulation necessitates calculating additional time variables.

- The problem has considered the time parameters of containers cooperated with subsystems like trucks and trucks. Realistic settings real terminals are arranged with these time parameters, e.g., the containers can not be loaded before the release time.

- The problem has entailed the assignments of I/O points in the crane loading, which results in exponential complex scheduling problems. For instances with n containers and m I/O points, the solution space reaches $n!m^n$ solutions.

The problem we investigate, which incorporates container release times and I/O point assignments, is the most realistic yard crane scheduling problem in the existing literature. This high level of realism, however, makes the problem extremely complex and difficult to solve. For instance, the latest commercial solver, IBM ILOG CPLEX, struggles to find optimal solutions for scenarios involving more than ten containers within one hour. Therefore, there is a critical need for more efficient problem-solving methods to effectively address the complexities of yard crane scheduling.

CHAPTER 4

Constructive heuristic methods

4.1 Introduction

This chapter introduces simple and fast rule-based heuristic methods to the yard crane scheduling problem (YCSP) introduced in Chapter 3. The rule-based method is a type of heuristic algorithm that relies on predefined rules or guidelines to make decisions or solve problems. These rules are typically derived from expert knowledge, practical experience, or specific problem characteristics, and they guide the heuristic in generating solutions quickly and efficiently. Rule-based heuristics are generally simple to implement and execute, making them very fast compared to more complex algorithms. They are particularly useful for problems where an exact solution is hard to obtain within a reasonable time frame. A rule-based heuristic does not have optimality guarantees following its heuristic nature. The effectiveness of the heuristic depends on the quality of the rules and how well they capture essential features of the problem. Three effective rules have been developed in Vallada

et al. (2023), including Time Parameter Rule (TPR), Modified Time Parameter Rule (MTPR) and Nearest Container Rule (NCR). In this chapter, a new rule, Simulated End-time Rule (SETR), is proposed and will be demonstrated to work better than the existing rule-based heuristics in the following sections.

The rest of the chapter is organized as follows: Section 4.2 shows the solution representation of the studied YCSP. A heuristic method is developed in Section 4.3 which involves the sequencing of containers and the assignments of I/O points. A local search is also proposed to improve the solution further. Section 4.4 introduces the benchmark instances, the experimental environment, as well as comprehensive experiments on the proposed heuristic methods. Eventually, we conclude the chapter in Section 4.5.

4.2 Solution representation

The YCSP studied in this thesis has two decisions to make. Specifically, the corresponding solution consists of the container sequence and the assignments of I/O points where the loading operation of a certain container must pass by an I/O point. Two presentations of the solution can be illustrated. The first one is two lists consisting of the loading sequence of containers and the I/O points assigned for delivering the corresponding container in the above list, presented as:

$$\begin{cases} c_1, c_2, \cdots, c_n & \text{\#List of loading order for all containers} \\ I/O_{c_1}, I/O_{c_2}, \cdots, I/O_{c_n} & \text{\#List of assigned I/O points for all containers} \end{cases}$$

where $\{c_1, c_2, \cdots, c_n\}$ is the loading order and I/O_{c_k} for any $k \in \{1, 2, \cdots, n\}$ is the I/O point and location for the k-th container. Moreover, we can also present the solution as a permutation of tu-

ples consisting of the container and I/O point as:

$$\{\langle c_1, I/O_{c_1}\rangle, \langle c_2, I/O_{c_2}\rangle, \cdots, \langle c_n, I/O_{c_n}\rangle\}.$$

Both solution representations are equivalent in essence. We adopt the latter one in this thesis. The following sections present heuristic constructive methods to build solutions by first sequencing the containers and then assigning the I/O points.

4.3 Constructive heuristic

This section presents a two-stage heuristic needed to construct a feasible solution, followed by a local search method. In the first step of the heuristic, we permute a sequence of containers. The second step carries out the I/O point assignment and calculates all times for each container. The following sections detail these two steps and the local search procedure.

4.3.1 First Step: generation of a sequence of containers

A dispatching rule, referred to as the Simulated End-time Rule (SETR), is proposed to permute the sequence of containers. The process begins with an empty sequence where all containers are evaluated by computing a value based on expression 4.1. The container with the minimum η_c value is then selected and added to the sequence where η_c is the simulated end-time calculated by

$$\eta_c = FC_c/w_i^R. \tag{4.1}$$

Recall that FC_c is exactly the final crane time for container c that has been introduced to the objective function (expression 3.2). To calculate

the estimated FC_c, an I/O point used for c is required and it is not assigned until the second step. Therefore, an average (middle-placed) I/O point from the seaside/landside is chosen as a substitute where this I/O point is assumed to be always available. This is actually a fast approximation of the I/O assignments, which is used to only calculate the value of η_c. The assignment of I/O points for each container will be introduced in the following sections.

Table 4.1 presents an example involving 4 containers in a block with 4 tiers ($T = 4$). The time parameter corresponds to each container's r_c, d_c, or e_c, depending on its type. We denote the current time as T_{now} where T_{now} is set to 0 for the first move from the initial position (Ini) of the crane. In this example, the initial position of the crane is $(1, 2, 5)$, while the locations of the sea and land middle I/O points are $(1, 0, 1)$ and $(1, 40, 2)$ respectively. Table 4.2 provides the calculated values of FC_c and η_c for each container. To illustrate, we detail the calculation for FC_{c_1} (a storage container) as follows:
$FC_{c_1} = \max(T_{now} + t_{ini,j} + t_{j,k}; r_c + t_{j,k}) = \max(0 + 6 + 12; 4 + 12) = 18$
where $t_{ini,j}$ and $t_{j,k}$ are calculated according to expression (3.1) as

$t_{ini,j} = |5 - 5| + \max\{|1 - 1|; |2 - 0|\} + |5 - 1| = 6$, and
$t_{j,k} = |1 - 5| + \max\{|1 - 1|; |0 - 7|\} + |5 - 4| = 12$.

After calculating all FC_c and η_c values, as shown in Table 4.2, container c_3 has the lowest η_c and is selected as the first in the sequence. At this point, T_{now} is set to $FC_{c_3} = 20$, and the current position of the YC is at the seaside I/O point, $(1, 0, 1)$. From this position, we recalculate the η_c values to determine the next container in the sequence. The final $SETR$ sequence for this example is $\{c_3, c_4, c_2, c_1\}$. This initial step of the heuristic performs $\frac{c(c+1)}{2} - 1$ calculations, resulting in a computational complexity of $\mathcal{O}(c^2)$.

Table 4.1: Input data for the $SETR$ rule application example.

	c_1	c_2	c_3	c_4
Type	C_1	C_2	C_3	C_4
Time Parameter	4	3	1	2
w_i^R	1	3	2	4
Location (x, y, z)	(1,7,4)	(2,3,2)	(3,5,1)	(4,3,2)

Table 4.2: First container selection for the example in the $SETR$ rule.

	c_1	c_2	c_3	c_4
FC_c	18	83	20	49
$\eta_c = FC_c/w_i^R$	$18/1 = 18$	$83/3 = 27.7$	$20/2 = 10$	$49/4 = 12.25$

4.3.2 Second Step: dynamic I/O assignment and timing

Once the container sequence is determined, the next step is to assign I/O points for each container and compute container times for the objective function (expression 3.2). The main idea of the I/O point assignment is to allocate the best available I/O point for each container. All I/O points are evaluated based on their availability at the relevant time parameter, i.e., the release time (r_c for $c \in C_1 \cup C_2$), due date (d_c for $c \in C_3$), and vehicle arrival time (e_c for $c \in C_4$). In extreme cases, if no I/O point is available, the one that becomes free the earliest will be chosen. If multiple points are available, the best one is selected based on the time cost of delivering this container to its destinations. This process starts from the current position of the crane and ends at the origin of the next container by the yard, during which I/O points are assigned. It is repeated until the entire container sequence is completed.

The procedure for assigning I/O points to different container types is illustrated in Figure 4.1. Based on the varying origin and destination locations, we consider three subroute scenarios for different container

types. The following notations are used: the current crane location (L_{crane}), the container's origin (O_c), and its destination (D_c). In Figure 4.1, arrows represent the time cost between connected locations, calculated using equation (3.1). For cases 1 and 2, only one I/O point (depicted as a solid color square or circle) is needed, whereas case 3 requires two I/O points. The selected I/O point(s) are those that minimize the total subroute time, which is the sum of all $t_{i,j}$ values along the subroute. To achieve this, all available I/O points (in cases 1 and 2) or I/O point pairs (in case 3) are evaluated. Once the I/O points are assigned, the start and finish times for I/O point usage (SO_c and FO_c), as well as crane movement times (SC_c and FC_c), are calculated. These times are then used to determine the objective function value (expression 3.2), as detailed in Section 3.3.

The very same instance in Table 4.1 is used again to illustrate the I/O assignment with the SETR sequence, $\{c_3, c_4, c_2, c_1\}$. Let us suppose that there are a total of 4 seaside I/O points $S1$ to $S4$ at $(s, 0, 1), s = \{1, 2, 3, 4\}$ and an equal number of I/O points $L1$ to $L4$ at $(l, 40, 2), l = \{1, 2, 3, 4\}$ in the landside. For container c_3 (type C_3), the crane starts from the initial position (L_{crane}) $Ini = (1, 2, 5)$ at time 0 to $O_{c_3}(3, 5, 1)$. At time d_c 1, all I/O points of $S1$ to $S4$ are available, and c_3 is transported to O_{c_3} at $t_{Ini,O_{c_3}} = 7$. The best I/O point, $S1$, is selected from the available I/O points $S1$ to $S4$ with the cost $t_{Ini,O_{c_3}} + t_{O_{c_3},S1} + t_{S1,O_{c_4}} = 7 + 13 + 10 = 30$. In this case, $SC_{c_3} = 0 + t_{Ini,O_{c_3}} = 7$, $FC_{c_3} = SO_{c_3} = \max\{0 + 7 + 13; 1\} = 20$, $E_{c_3} = \max\{1 - 20; 0\} = 0$, and $FO_{c_3} = \max\{20; 1\} = 20$ with the calculations involving to Section 3.6. Table 4.3 lists all the calculated terms of the objective function (3.2).

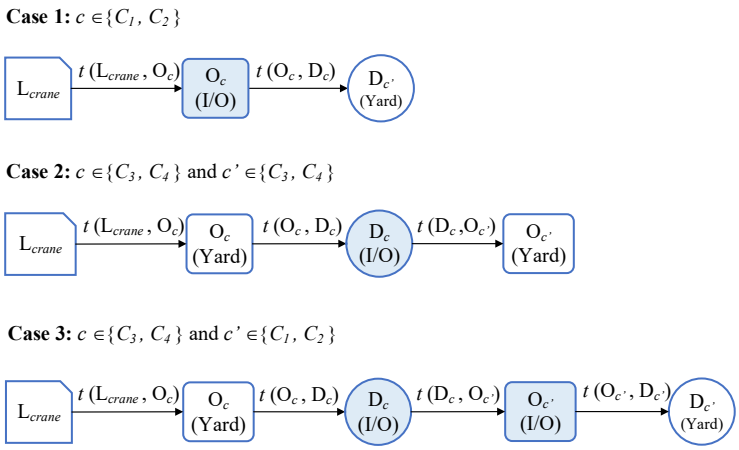


Figure 4.1: Assigning I/O points in a sequence for different types of containers. Elaborated by the author.

Table 4.3: Values for the objective function terms after the assignment of I/O points.

c	I/O	SC_c	FC_c	SO_c	FO_c
c_3	$S1$	7	20	20	20
c_4	$L1$	20	73	73	73
c_2	$L2$	79	122	3	79
c_1	$S2$	132	144	4	132

Furthermore, we compare the proposed dynamic I/O assignments with two approaches using pre-fixed I/O points. It will be shown in Section 4.4.3 that the proposed dynamic assignment is vastly superior to random or nearest I/O fixed assignments. The conclusion follows the intuition that the proposed dynamic I/O assignment method can dynamically adjust the occupancy of I/O points, which is beneficial when decreasing the congestion at I/O points.

4.3.3 Local search method

Local search methods are extensively utilized in optimization problems to enhance solution quality (see Lim *et al.*, 2004; He *et al.*, 2015a,b;

Galle *et al.*, 2018). These methods explore the solution space by examining neighbor spaces with small modifications on solutions. Typically, the local search rearranges elements in the current solution, such as swapping or inserting jobs in the sequence. The best neighbor that results in the best fitness value will be accepted. By focusing on local adjustments, local search methods enable the improvement of the overall solution in an adaptive way.

Note that the fitness function may not be limited to the objective function. Evaluating the exact objective function can be computationally expensive, especially for complex problems. To make the search more efficient, an alternative function that only captures certain key aspects of the objective might be a good proxy or surrogate function. It might quickly evaluate and compare solutions, guiding the search toward improvement without the need for a full, costly evaluation of the objective function. Therefore, proxy or surrogate functions are very useful in the local search, particularly for complex, large-scale problems, such as the investigated YCSP in this thesis.

Algorithm 1 outlines a local search specifically developed for the YCSP. This local search employs an insertion neighborhood using a best-improvement criterion. More specifically, the local search starts from an initial solution π, consisting of a sequence and corresponding I/O assignments. The main loop consists of inserting all containers into all positions of the sequence and evaluating each solution using the proxy function (pf). This proxy is an estimation of the real objective function f of expression 3.2. The only difference between pf and f is that pf does not require the I/O assignment. Instead, pf inherits the originally assigned I/O points for all containers in the original sequence. Note that this original assignment might no longer be feasible, but feasibility is not required to guide the local search. In each insertion, the top β most promising candidates are retained in a $TopBest$ set. Only these β candidates are subsequently evaluated using the actual

objective function (f), which includes the necessary I/O assignments and timing adjustments for the new sequence (refer to Section 4.3.2) to ensure feasibility. The best solution among these β candidates will be appended at the end of the sequence. Thanks to the use of the proxy function, for a given sequence with n containers and m I/O points, the complexity of one pass of the proposed local search decreases from $\mathcal{O}(n^3m)$ to $\mathcal{O}(n^2\beta m)$.

Algorithm 1 : Local search

```
1: Input π
2: Output π_best
3: π_best = π
4: improvement := true
5: while improvement == true do
6:     TopBest := ∅
7:     improvement := false
8:     for i := 1 to n do
9:         Generate set TopBest with the best β insertions of container
           i in all positions such that pf(TopBest) is minimized
10:        for b := 1 to β do
11:            if f(TopBest(b)) < f(π_best) then
12:                improvement := true
13:                π_best := TopBest(b)
14:                π := π_best
```

We illustrate the local search procedure in Figure 4.2 using five containers (Con1 to Con5) and three I/O points (IO1, IO2, and IO3) positioned on the same side, with the parameter β set to 2. The search begins with an initial solution $\{\langle$Con1, IO1$\rangle$, $\langle$Con3, IO2$\rangle$, $\langle$Con2, IO1$\rangle$, $\langle$Con4, IO3$\rangle$, $\langle$Con5, IO1$\rangle\}$, having an objective value of 50. The first container, Con1, explores potential insertion positions from 2 to 5, identifying $TopBest = \{2, 3\}$ as the options with the smallest pf values. Each insertion is evaluated using the I/O assignment method described in Section 4.3.2, aiming to improve the solution. Two candidate solutions are tested, with the second being selected as the incumbent

solution, where the I/O points of Con3 and Con2 are updated. The process continues with Con2, which is tested for all possible insertion positions and is ultimately placed in position 4, updating the I/O point for Con1 accordingly. Subsequently, Con3 will do a similar insertion procedure until no further improvements can be made during the main loop.

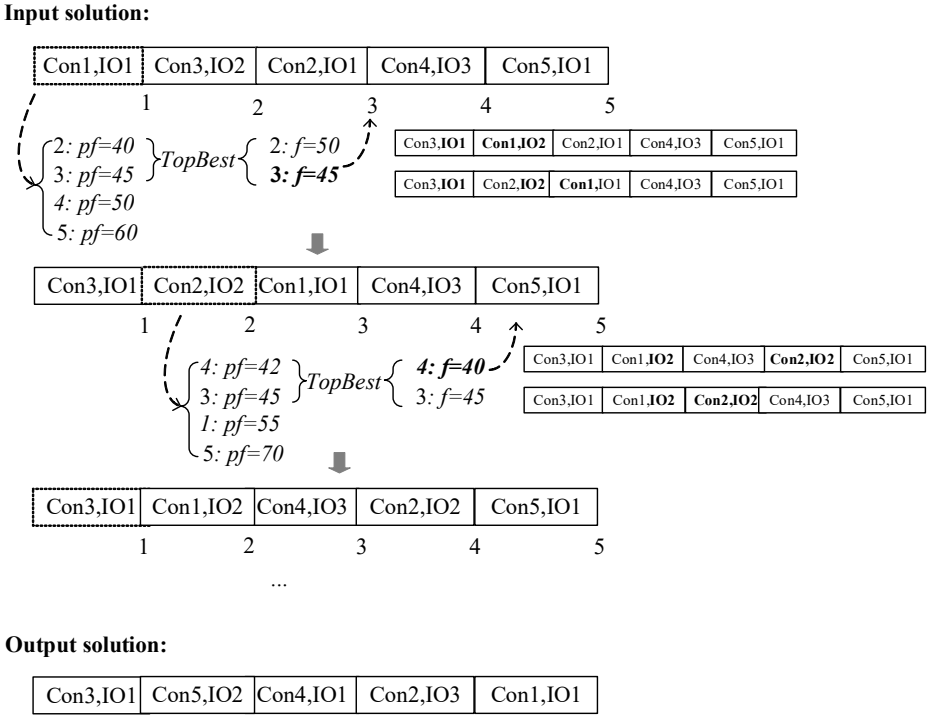


Figure 4.2: Example for the local search method with 5 containers and $\beta = 2$. Elaborated by the author.

4.4 Computational experiments

4.4.1 Computational environment

All experiments conducted in this thesis were completed under the same experimental environment with the same computational settings.

Figure 4.3 illustrates the computational framework operated for all the experiments in this thesis. Some main information is collected as follows:

- Coding language: C#
- Framework: .Net 6.0
- IDE: Microsoft Visual Studio 2019

We have tested all instances in virtual machines (VMs) running in x64 Windows 10 OS, each one configured with 16 GBytes RAM memory and four virtual cores. All the virtual machines are run with the same settings and are powered by several servers under the OpenStack virtualization platform. Each server is equipped with two 18 core Intel Xeon Gold 5220 processors running at 2.2 GHz and 384 GBytes of RAM. Note that all computational tasks are randomly distributed to VMs across these servers, and there is no parallel coding or computing during the whole experiment.

4.4.2 Benchmark instances

The testing data sets consist of the 720 benchmark instances from Vallada *et al.* (2023). All instances are divided into four subsets of a total of 12 size combinations, including small and small tight sets with $n \in \{5, 10\}$ containers, the medium set with $n \in \{15, 20, 30, 40\}$ containers and the large set with $n \in \{50, 100, 150, 200\}$ containers.

Yard block data generation

As aforementioned in Chapter 3, we consider a European terminal configured with I/O points located at two sides of the yard block. Such a yard block is illustrated in Figure 3.2 consisting of R rows, B bays, and T tiers. The yard block size refers to the settings of a real terminal yard in Gharehgozli *et al.* (2014, 2019) with very similar sizes of rows,

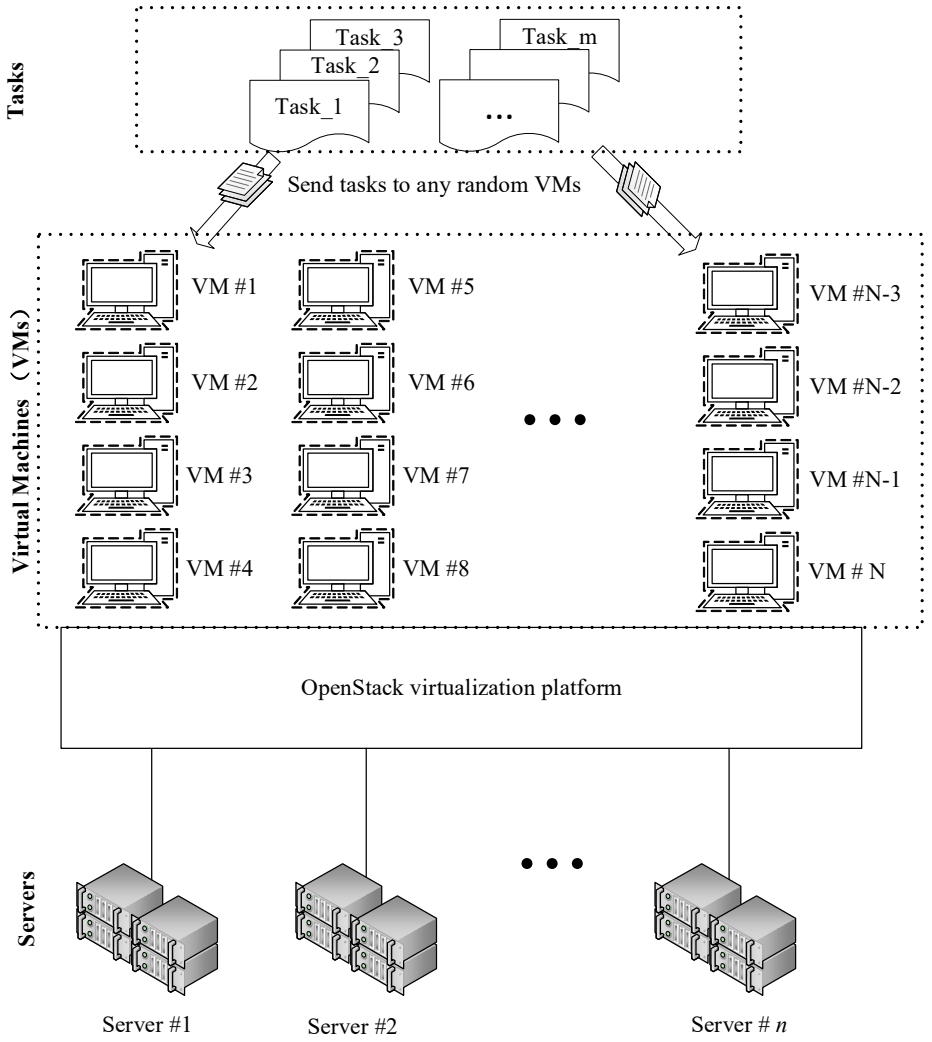


Figure 4.3: Computational framework utilized in this thesis, where we randomly distribute m computational tasks to N virtual machines (VMs) powered on n servers. Elaborated by the author.

bays, and tiers as $R = 10$, $B = 42$ and $T = 4$ respectively. We denote all the locations (x, y, z) with coordinate x, y and z. The yard crane is initially located at Ini where the trolley is at the top tier in the yard block; that is, its z coordinate is set to $T + 1$ (5 in this case). Coordinates x and y are random integers chosen from $0 < x \leq R$ and $0 < y \leq B$ for each individual instances. Finally, the speed of the crane in the gantry, trolley, and hoisting is constant (equal to 1) as there is no crane interference (Speer and Fischer, 2017).

I/O points data generation

Two kinds of settings for the I/O points are adopted. For all small tight instances, there are one and two I/O points on the landside and seaside, respectively. In this case, the location of the I/O points at the landside is $(4, B+1, 2)$ and the location of I/O points on the seaside are $(4, 0, 1)$ and $(5, 0, 1)$. This is an extreme I/O setting used to investigate the congestion in the objective function (expression 3.2) under very small I/O sizes.

For the rest of the instances, there are six and ten I/O points on the landside and seaside, respectively. All I/O points are located on the edges of the block. Their locations in coordinates (x, y, z) were obtained in the following way: For 10 I/O seaside points, locations are $(x, 0, 1)$, where $x = \{1, 2, \ldots, 10\}$. An extra bay (0) is considered only for the I/O points on the seaside. Moreover, the z value is set to 1 for all these I/O points. With respect to 6 landside I/O points, locations are set to $(x, B + 1, 2)$, where $x = \{2, 3, \cdots, 7\}$. Again, an extra bay (43 in this case) is considered only for I/O points on the landside. Moreover, their z value is set to 2 since the containers are delivered to a truck platform.

Container requests data generation

Each instance owns n requests, and each request corresponds to a unique container. An instance example is displayed in Appendix A. The container data is randomly generated, thus being unique for each instance. For each container request, the necessary information includes the container type, yard location, number of reshuffles, time parameters and weights.

- Container type: As introduced in Section 3.2, we divide the container into four types based on different origins and destinations. For each container, the container type is randomly generated from 1 to 4 for C_1 to C_4.

- Yard location: The yard locations of containers indicate the number of rows, bays and tiers where the container is located or will be located. These locations are generated randomly as integer coordinates (x, y, z), $0 < x \leq R, 0 < y \leq B$ and $0 < z \leq T$.

- Number of reshuffles: For retrieval containers, a reshuffle may be needed in some cases, so 2 unit times per reshuffle (Res_{time}) are also included in the instances.

- Time parameter: The time parameters of containers follow a uniform distribution in the range $U(0, \rho \times RoundTrip)$ for storage containers and $U(0, 0.4 \times RoundTrip)$ for retrieval containers, where $RoundTrip = c \times B$ is the maximum total distance for loading all container requests. The parameter ρ is set to three different values $\{0.1, 0, 4, 0.7\}$, reflecting different levels of congestion at the I/O points.

- Weights: Two weight settings, Equal (E) and Non Equal (NE) are adopted for all instant sets. The settings are used to penalize tardiness (w_c^R), and congestion (w_c^C) in the objective function

(3.2). In Equal instances, all weights are set to 1, whereas for Non Equal instances, all weights are fixed to 3 for containers on the seaside and 1 for the landside, respectively.

In summary, there are three values of ρ, the two types of weight settings and the 12 size combinations, resulting in $3 \times 2 \times 12 = 72$ instance combinations. For each combination, we generate 10 instance replicates to enrich the final benchmark instances. This results in a total of 720 benchmark instances for the final evaluation of all the proposed methods proposed in this thesis. Moreover, one example file of the instance is presented in Appendix A.

4.4.3 Computational results of the heuristic methods

We have executed comprehensive experiments on the 720 instances introduced by Vallada *et al.* (2023) in small, small tight, medium and large sets presented in the previous section. The proposed rule-based heuristic SETR in Section 4.3 is compared to existing rules TPR, $MTPR$ and NCR proposed in Vallada *et al.* (2023). Correspondingly, three local search methods TPR_{ls}, $MTPR_{ls}$, NCR_{ls} are proposed by Vallada *et al.* (2023) with the corresponding rule based heuristic and a local search procedure. The local search inserts every container into the best position after testing all possible ones in which the I/O points will be assigned to find the best insertion. Note that such local search (Vallada *et al.*, 2023) methods terminate after n seconds for instances with the same number (n) of containers. As an alternative, we design a similar local search method $SETR_{ls}$ with the SETR constructive heuristic and one run of the local search ($\beta = 3$) in Section 4.3.

Two random methods, $RAND1$ and $RAND2$ are proposed. Both random methods sequence all containers in a random order, while $RAND1$ assigns I/O points for each container, while $RAND2$ assigns

I/O points using the I/O methods in Section 4.3.2. Specifically, all the rule-based heuristics involved are briefly summarized as follows:

- Time Parameter Rule (TPR): A loading sequence of containers is composed in the increasing order of the container's time parameters.
- Modified Time Parameter Rule ($MTPR$): Based on the TPR rule, it modifies the next container with the nearest one starting exactly from the terminated side (yard, seaside or landside) of the current container. For example, a current container ends at the yard side, and the next one should be the one in the following sequence that starts from the yard side ($c \in C_3 \cup C_4$).
- Nearest Container Rule (NCR): Starting from the current position of the yard crane, the next loaded container is chosen from all the unloaded containers with the one resulting in the minimum rank value ($\eta_c = TT_c/w_i^R$) where TT_c is the total time from the current position to the destination of container c.
- Simulated End-Time Rule ($SETR$): The rule has been introduced in Section 4.3.1 which sequences containers based on the simulated finishing time of the yard crane.
- Random method 1 ($RAND1$): It generates solutions using a random sequence of containers and random assignments of I/O points.
- Random method 2 ($RAND2$): Solutions are generated with a random sequence of containers and assignment methods shown in Section 4.3.2.

Computational comparison between rules

We first compare the performance of all different heuristic rules using the Design of Experiments (DOE) approach together with the Analysis of

Variance (ANOVA) statistical technique (Montgomery, 2017). ANOVA is a statistical model used to study if different factors have a significant effect on a response variable. It has to be noted that ANOVA is a parametric technique, and usually, three hypotheses about the data need to be met. These are normality, homoscedasticity and the independence of residuals. While ANOVA is very robust w.r.t. to mild to severe violations of normality and homoscedasticity, special care needs to be put into the independence of the residuals. However, in computer experimentation, this is largely the case as one treatment, entailing an algorithm configuration, runs and terminates, and there is no effect on the next algorithm run. Randomization of experiments also helps with independence. The ANOVA compares the Relative Percentage Deviation (RPD) and demonstrates if the differences observed between these rules have a statistically significant effect. The RPD serves as a measure of gaps between the incumbent solution objective value Sol and the one from the best solution found so far ($Best$), calculated by:

$$RPD = \frac{Sol - Best}{Best} \times 100. \tag{4.2}$$

Figure 4.4 presents the means plot and intervals of RPD results for all test instances using various rule-based heuristics, including TPR, $MTPR$, NCR, $SETR$, $RAND1$ and $RAND2$. These intervals allow for multiple pairwise comparisons. Recall that if any two intervals overlap, the differences of the corresponding means levels are statistically insignificant. Their RPD values are calculated using the best solutions obtained from all methods examined in this chapter. Among the heuristics, the proposed SETR method indicates a significantly lower RPD, clearly outperforming other rule-based approaches. In contrast, the RPD results for the other methods are substantially higher, exceeding 50. Notably, all heuristic-based rules perform considerably better than the random methods, where $RAND2$ yields notably superior solutions compared to $RAND1$. This suggests that the I/O assignment meth-

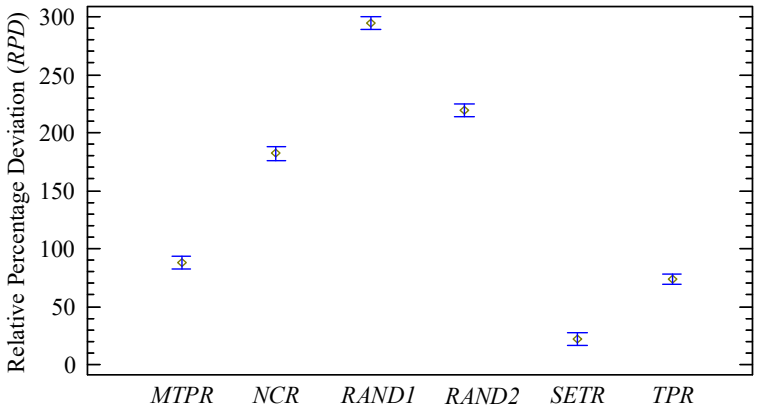


Figure 4.4: Means plot and intervals between different rule-based heuristics with 95% Tukey's Honest Significant Difference (HSD) confidence intervals. Elaborated by the author.

ods discussed in Section 4.3.2 are effective, even though both random methods result in relatively high RPD values.

Computational comparison between I/O assignments

A comparison with a random or static I/O assignment was carried out in order to compare the effect of the dynamic assignment. Two kinds of pre-fixed I/O points assigned to each container were designed:

- RandIO: a random I/O point at the corresponding side;
- NearestIO: the nearest I/O point to the container's yard position at the corresponding side.

We have carried out additional experiments by only replacing the proposed dynamic assignment with the new two types of pre-fixed I/O points. Table 4.4 shows the average RPD grouped by different instance sizes. It proves that the proposed dynamic assignment is vastly superior to random or nearest I/O fixed assignments with a far lower average RPD. The same conclusions can also be found by the means intervals and interactions with the number of containers (Figure 4.5).

The conclusion from this experiment is expected and follows intuition, since the proposed dynamic I/O assignment method is able to dynamically adjust the occupancy of I/O points, which is beneficial when decreasing the congestion at I/O points for the objective function. However, with prefixed I/O assignments, this is not possible. Among the two fixed variants, it is also intuitive that a random assignment will hedge against congestion better than the nearest I/O point that is too greedy and might result in very busy I/O points when the containers to move are all clustered together and close to the same I/O points.

Table 4.4: Average RPD (%) over the best known solution for all instances grouped by the number of containers (n).

n	RandIO	NearestIO	Dynamic
5	0.05	14.43	0.39
10	0.24	21.25	1.00
15	2.80	40.92	**0.25**
20	6.54	41.29	**0.39**
30	13.56	39.61	**0.44**
40	16.69	35.26	**0.75**
50	17.78	31.18	**0.83**
100	20.03	20.72	**0.77**
150	20.40	16.25	**0.57**
200	19.35	13.32	**0.52**
Ave.	7.47	10.23	**0.20**

Results for small and small tight instances

All small and small tight instances can be optimally solved in Vallada *et al.* (2023) by using the latest IBM ILOG CPLEX 20.1 mathematical solver. We first compare the performance of the proposed heuristics on small instances. Table 4.5 lists the optimal solutions found by the $SETR$ and $SETR_{ls}$ and other rule-based heuristics and local search methods, grouped by the container size (n), weights type (w_j) and

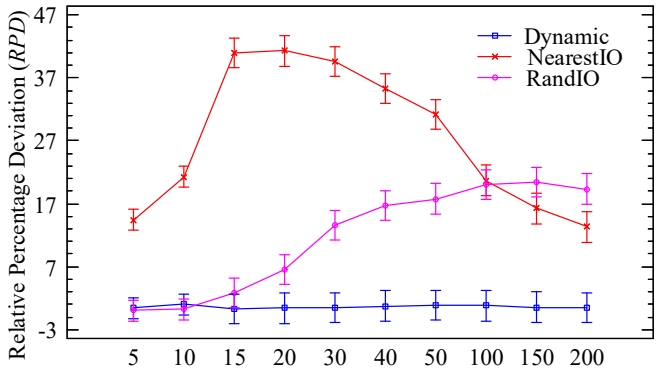


Figure 4.5: Interaction and Tukey HSD intervals at the 95% confidence level between the number of containers (n) and for the three I/O assignment alternatives. Elaborated by the author.

instance parameter (ρ). Consequently, each cell includes 10 instances in Table 4.5. The total numbers of optimal solutions obtained by various methods are counted at the bottom of the table. Among all rule-based heuristics, $SETR$ shows the highest optimality at 38, indicating its strong performance across all small instances. The $MTPR$ and TPR methods, with totals of 3 and 2 respectively, perform significantly lower in comparison, suggesting that they are less effective rules for finding optimal solutions. Moreover, NCR_{ls} shows the highest optimality among all local search methods, which is better than $SETR_{ls}$ although the NCR_{ls} requires much more CPU time, as shown in later sections.

Table 4.6 presents the optimality of rule-based heuristics and local search methods across small tight instances. The total number of optimal solutions achieved by TPR, $MTPR$, and NCR is relatively low, with the highest result observed for $SETR$ (26), suggesting it is the most effective among all compared heuristic rules. In contrast, all local search methods consistently outperform the rule-based heuristics, particularly in TPR_{ls} (45), NCR_{ls} (49), and $SETR_{ls}$ (49), highlighting their superior ability to enhance solution quality.

The previous subsection displayed the number of optimality, while

Table 4.5: Number of optimal solutions obtained by different approaches for small instances with Equal (E) and Non Equal (NE) weights (w_j) grouped by parameter ρ.

				Rule-based heuristic				Local search methods			
n	w_j	ρ	#	TPR	$MTPR$	NCR	$SETR$	TPR_{ls}	$MTPR_{ls}$	NCR_{ls}	$SETR_{ls}$
5	E	0.1	10	1	1	8	3	8	8	9	6
		0.4	10	0	1	2	6	8	6	10	9
		0.7	10	0	1	1	4	6	8	8	8
	NE	0.1	10	0	0	3	2	6	7	7	4
		0.4	10	0	0	3	3	8	7	9	6
		0.7	10	1	0	2	4	8	8	7	8
10	E	0.1	10	0	0	0	0	2	0	0	1
		0.4	10	0	0	0	1	0	1	3	1
		0.7	10	0	0	1	2	1	1	1	2
	NE	0.1	10	0	0	0	0	3	1	2	1
		0.4	10	0	0	0	0	3	2	2	1
		0.7	10	0	0	0	0	3	3	6	2
	Total		10	2	3	20	**25**	56	52	**64**	49

Table 4.6: Number of optimal solutions obtained by different approaches for small tight instances with Equal (E) and Non Equal (NE) weights (w_j) grouped by parameter ρ.

				Rule-based heuristic				Local search methods			
n	w_j	ρ	#	TPR	$MTPR$	NCR	$SETR$	TPR_{ls}	$MTPR_{ls}$	NCR_{ls}	$SETR_{ls}$
5	E	0.1	10	0	1	2	6	3	3	7	9
		0.4	10	1	2	2	4	6	5	5	7
		0.7	10	0	0	2	4	7	6	5	6
	NE	0.1	10	0	0	0	1	6	5	5	3
		0.4	10	0	0	5	5	8	7	7	8
		0.7	10	2	2	0	0	7	7	8	7
10	E	0.1	10	0	0	0	1	1	0	2	2
		0.4	10	0	0	0	1	1	1	0	2
		0.7	10	0	0	0	1	3	1	3	2
	NE	0.1	10	0	0	0	0	0	0	2	0
		0.4	10	0	0	0	0	2	1	2	1
		0.7	10	0	0	0	0	1	2	3	2
	Total		10	3	5	11	23	45	38	49	49

the average gaps to optimal solutions matter as well. We are now using the RPD (expression 4.2) to describe the gap between the incumbent solution objective and the best solution found so far. Table 4.7 displays the RPD results from optimal solutions and CPU times for the small instances. On average, the RPD values of rule-based methods are significantly higher compared to the local search methods, indicating that the additional local search procedure works better for the heuristic rules. Among all rule-based methods, $SETR$ achieves the lowest deviation (2.36%), with RPD 27.88 times lower than TPR (65.8%), 23.25 times smaller than $MTPR$ (54.88%) and 6.3 times lower than NCR (14.84%). Note that even though NCR_{ls} has obtained more optimal solutions than $SETR_{ls}$, as shown in Table 4.5, the latter yields the lowest average RPD (1.99%), highlighting the effectiveness of the proposed methods in optimizing solutions. Taking into account the CPU times, NCR_{ls} requires even significantly more computational effort, with an average CPU time of 7.5 seconds, 104 times more than $SETR_{ls}$, which averages at just 0.06 seconds.

Table 4.7: Average RPD (%) over optimal solutions and CPU times (s) for small instances with Equal (E) and Non Equal (NE) weights (w_j) grouped by parameter ρ.

			Rule-based heuristic					Local search methods					
n	w_j	ρ	TPR	$MTPR$	NCR	$SETR_{ls}$	$time$	TPR_{ls}	$MTPR_{ls}$	NCR_{ls}	$time$	$SETR_{ls}$	$time$
5	E	0.1	30.49	24.37	4.43	5.05	0.01	0.96	0.25	1.30	5.00	0.73	0.05
		0.4	34.10	21.19	10.55	5.43	0.00	4.77	2.16	0.00	5.00	0.33	0.05
		0.7	55.82	57.63	19.51	5.06	0.01	5.31	2.76	0.76	5.00	2.23	0.05
	NE	0.1	71.97	58.54	4.74	3.57	0.01	0.35	0.26	0.26	5.00	1.71	0.06
		0.4	53.36	46.61	9.89	7.91	0.00	2.44	2.76	0.54	5.00	2.45	0.05
		0.7	38.13	57.95	12.31	10.20	0.01	2.37	2.37	2.31	5.00	0.76	0.05
10	E	0.1	75.00	32.44	15.02	8.52	0.00	3.27	4.73	4.41	10.00	2.99	0.06
		0.4	68.08	56.43	22.24	9.01	0.01	4.48	3.99	2.07	10.00	4.90	0.06
		0.7	69.93	55.87	15.27	17.18	0.02	4.69	8.22	4.06	10.00	2.32	0.06
	NE	0.1	98.02	68.58	13.15	21.24	0.01	2.11	7.04	2.54	10.00	3.96	0.06
		0.4	105.47	72.21	32.12	23.23	0.01	1.45	6.94	2.40	10.00	1.72	0.06
		0.7	89.24	106.72	18.88	29.11	0.00	2.88	4.29	3.24	10.00	4.23	0.07
Total Average			65.80	54.88	14.84	**12.13**	0.01	2.92	3.81	**1.99**	7.5	2.36	0.06

Table 4.8: Average RPD (%) over optimal solutions and CPU times (s) for small tight instances with Equal (E) and Non Equal (NE) weights (w_j) grouped by parameter ρ.

			Rule-based heuristic					Local search methods					
n	w_j	ρ	TPR	$MTPR$	NCR	$SETR$	$time$	TPR_{ls}	$MTPR_{ls}$	NCR_{ls}	$time$	$SETR_{ls}$	$time$
5	E	0.1	58.49	25.96	8.92	0.91	0.01	2.56	2.05	1.19	5.00	0.06	0.06
		0.4	50.11	48.24	31.83	8.98	0.02	2.22	3.71	2.52	5.00	0.68	0.03
		0.7	48.66	35.73	22.27	9.96	0.01	2.38	2.17	7.56	5.00	0.77	0.01
	NE	0.1	44.16	39.33	22.17	13.95	0.00	1.82	2.50	1.98	5.00	0.77	0.08
		0.4	62.21	32.98	14.12	7.10	0.00	0.83	2.15	3.68	5.00	0.45	0.09
		0.7	58.76	33.05	49.93	31.75	0.01	1.39	1.39	1.26	5.00	1.26	0.39
10	E	0.1	61.33	47.88	20.96	14.07	0.00	6.44	7.31	5.04	10.00	6.35	0.03
		0.4	96.08	69.67	35.98	17.80	0.01	6.74	8.72	12.06	10.00	4.16	0.05
		0.7	110.16	120.79	60.75	14.51	0.02	7.65	10.51	6.44	10.00	4.36	0.05
	NE	0.1	73.49	51.63	23.77	23.27	0.01	6.99	8.49	2.65	10.00	5.43	0.06
		0.4	80.93	51.54	23.11	19.47	0.01	5.08	10.41	3.63	10.00	5.75	0.06
		0.7	106.75	115.93	42.96	44.81	0.00	8.50	9.06	4.82	10.00	6.23	0.06
Total Average			70.93	56.06	29.73	**17.21**	0.01	4.38	5.71	4.40	7.5	**3.80**	0.08

Results for medium instances

Note that the RPD values shown in Tables 4.9 and 4.10 are computed using the best solution from all compared methods for the medium and large instances. This is because IBM ILOG CPLEX 20.1 is unable to provide optimal solutions for instances larger than 10 containers within one hour. As illustrated in Table 4.9, the $SETR$ method consistently outperforms others for all medium instances, achieving the lowest average RPD values across various problem sizes and parameter settings. The local search methods ($SETR_{ls}$, TPR_{ls}, $MTPR_{ls}$, and NCR_{ls}) further enhance solution quality. Among them, $SETR_{ls}$ consistently delivers competitive or superior RPD results, though it produces slightly worse outcomes than NCR_{ls} for a subset of instances with $n = 20$. Despite this, $SETR_{ls}$ demonstrates remarkable efficiency, with an average computation time of less than one second across all instances. This efficiency underscores its practical value, even for larger problem sizes.

Table 4.9: Average RPD (%) over the best solutions and CPU times (s) for medium instances with Equal (E) and Non Equal (NE) weights (w_j) grouped by parameter ρ.

			Rule-based heuristic					Local search methods					
n	w_j	ρ	TPR	$MTPR$	NCR	$SETR$	$time$	TPR_{ls}	$MTPR_{ls}$	NCR_{ls}	$time$	$SETR_{ls}$	$time$
15	E	0.1	70.79	36.49	11.12	8.94	0.02	5.28	4.98	4.11	50.00	3.37	0.07
		0.4	80.45	45.05	28.50	13.08	0.02	6.42	3.20	4.98	50.00	3.56	0.06
		0.7	67.53	84.05	29.65	12.16	0.04	4.45	3.69	3.87	50.00	1.58	0.07
	NE	0.1	111.62	70.21	27.31	24.63	0.08	1.88	3.40	0.40	50.00	2.54	0.06
		0.4	119.20	78.45	39.12	37.04	0.01	7.62	3.83	5.26	50.00	4.05	0.07
		0.7	116.78	121.67	40.51	49.45	0.06	2.15	10.16	7.78	50.00	8.09	0.07
Average			94.39	72.65	29.37	**24.22**	0.04	4.63	4.88	4.40	50.00	**3.87**	0.07
20	E	0.1	83.13	50.39	11.00	7.26	0.01	4.66	3.78	1.49	100.00	3.00	0.09
		0.4	79.24	57.23	36.66	8.27	0.02	5.46	3.41	5.26	100.00	2.05	0.08
		0.7	107.90	111.49	41.99	13.18	0.03	7.35	9.09	5.00	100.00	3.28	0.07
	NE	0.1	102.35	55.76	30.86	26.01	0.05	2.09	4.67	0.73	100.00	2.16	0.09
		0.4	126.31	66.78	32.57	43.03	0.07	5.17	8.10	3.17	100.00	6.41	0.1
		0.7	155.85	185.47	82.03	71.62	0.11	5.17	11.11	7.24	100.00	7.67	0.09
Average			109.13	87.85	39.19	**28.23**	0.05	4.98	6.69	**3.81**	100.00	4.09	0.09
30	E	0.1	88.47	48.31	19.44	6.57	0.07	3.31	4.96	4.72	150.00	0.77	0.14
		0.4	120.04	67.35	54.94	12.66	0.08	6.74	3.18	4.38	150.00	5.32	0.13
		0.7	121.03	116.97	89.10	15.60	0.10	5.25	4.60	6.03	150.00	1.67	0.14
	NE	0.1	99.67	58.09	43.68	36.91	0.08	4.29	5.14	0.70	150.00	5.79	0.13
		0.4	138.56	76.12	86.18	43.10	0.10	4.39	4.20	3.66	150.00	6.87	0.16
		0.7	144.34	125.48	74.67	85.88	0.10	6.87	8.01	3.73	150.00	2.57	0.15
Average			118.68	82.05	61.33	**33.45**	0.09	5.14	5.01	3.87	150.00	**3.83**	0.14
40	E	0.1	100.89	55.63	37.95	9.68	0.03	5.14	5.96	4.94	200.00	1.27	0.24
		0.4	112.98	53.76	68.36	16.56	0.11	9.11	3.27	6.87	200.00	3.89	0.24
		0.7	141.54	138.29	97.08	18.34	0.06	8.79	4.52	8.27	200.00	2.75	0.22
	NE	0.1	113.25	55.50	59.13	44.97	0.11	3.14	3.34	2.46	200.00	2.89	0.24
		0.4	133.90	81.51	75.38	43.75	0.09	3.80	7.07	3.05	200.00	5.38	0.22
		0.7	174.49	159.18	180.22	79.73	0.05	3.09	6.30	3.63	200.00	9.41	0.22
Average			129.51	90.65	86.35	**35.50**	0.08	5.51	5.08	4.87	200.00	**4.27**	0.23
Total Average			112.93	83.30	54.06	**30.35**	0.06	5.07	5.42	4.24	125.00	**4.01**	0.13

Results for large instances

Table 4.10 reports the RPD results for the most challenging instances, which include up to 200 containers. The results indicate that the rule-based heuristics TPR, $MTPR$, and NCR struggle significantly with these large-scale problems, leading to very high average RPD values, especially for TPR (154.66%) and NCR (145.03%). In comparison, the proposed $SETR$ heuristic performs much better, achieving an average RPD of 36.17%.

Local search methods deliver substantially better results than the rule-based heuristics. On average, all local search methods maintain RPD values below 10%, which is a marked improvement over the rule-based approaches. Specifically, $SETR_{ls}$ is the best-performing local search method, with an exceptionally low average RPD of just 1.74%. This is significantly lower than the RPD results of TPR_{ls} (7.92%), $MTPR_{ls}$ (8.61%), and NCR_{ls} (5.99%). Remarkably, for the largest instances with $n = 200$, $SETR_{ls}$ achieves a zero RPD, indicating that it consistently finds the best solutions across all 60 instances, outperforming all other methods.

In summary, the proposed $SETR$ heuristic and its local search variant $SETR_{ls}$ stand out as highly effective methods. $SETR$ heuristic demonstrates strong competitiveness against the state-of-the-art rule-based heuristics including TPR, $MTPR$, and NCR across all 240 small and small tight instances. While its local search variant, $SETR_{ls}$, significantly outperforms all compared methods. The method requires fewer CUP times and maintains low RPD values, making it strongly competitive with other proposals for the YCSP.

4.5 Conclusions

Improving loading efficiency is a vital concern for terminal operators, requiring effective methods to optimize the scheduling of loading and

Table 4.10: Average RPD (%) over the best solutions and CPU times (s) for large instances with Equal (E) and Non Equal (NE) weights (w_j) grouped by parameter ρ.

			Rule-based heuristic					Local search methods					
n	w_j	ρ	TPR	$MTPR$	NCR	$SETR$	time	TPR_{ls}	$MTPR_{ls}$	NCR_{ls}	$time$	$SETR_{ls}$	$time$
50	E	0.1	103.91	65.13	31.75	9.68	0.07	3.57	6.91	3.81	50.00	1.56	0.35
		0.4	138.64	78.66	59.39	13.18	0.05	6.13	3.66	6.76	50.00	5.55	0.31
		0.7	145.95	116.76	94.31	13.89	0.07	8.07	8.83	5.85	50.00	4.54	0.31
	NE	0.1	113.35	84.33	43.49	47.52	0.09	3.41	3.13	1.62	50.00	3.57	0.40
		0.4	133.35	70.74	98.16	56.50	0.11	4.45	4.27	2.61	50.00	4.69	0.33
		0.7	162.75	118.10	112.90	79.47	0.14	7.33	8.01	3.48	50.00	7.07	0.37
Average			132.99	88.96	73.33	**36.71**	0.09	5.49	5.80	**4.02**	50.00	4.50	0.35
100	E	0.1	106.68	61.39	85.64	7.31	0.10	5.22	7.10	3.84	100.00	0.25	1.03
		0.4	147.43	84.58	142.21	12.36	0.13	6.48	7.19	5.69	100.00	0.03	1.28
		0.7	200.37	111.65	198.32	17.60	0.11	8.42	7.23	6.31	100.00	1.81	1.07
	NE	0.1	100.06	75.07	74.34	49.40	0.10	3.35	4.15	1.48	100.00	1.20	1.15
		0.4	171.59	92.23	143.05	51.43	0.14	5.54	5.03	1.22	100.00	3.04	1.11
		0.7	192.54	93.99	211.90	80.20	0.19	2.98	5.29	0.85	100.00	5.91	1.30
Average			153.11	86.48	142.58	**36.38**	0.13	5.33	6.00	3.23	100.00	**2.04**	1.16
150	E	0.1	105.74	59.85	97.79	6.16	0.65	5.70	6.51	5.73	150.00	0.00	2.87
		0.4	152.81	77.29	163.43	9.32	0.23	8.46	8.43	8.36	150.00	0.00	3.06
		0.7	217.90	115.84	246.81	16.48	0.39	10.42	10.22	9.59	150.00	0.40	2.77
	NE	0.1	108.44	70.47	108.95	47.66	0.36	5.55	6.24	3.92	150.00	0.28	2.47
		0.4	171.42	93.61	178.76	57.11	0.28	8.62	12.53	3.39	150.00	0.30	2.97
		0.7	220.72	121.07	262.45	75.62	0.28	5.43	7.77	2.62	150.00	1.53	2.65
Average			162.84	89.69	176.37	**35.39**	0.37	7.37	8.62	5.60	150.00	**0.42**	2.80
200	E	0.1	100.50	55.64	110.44	4.84	0.37	8.51	10.04	7.25	200.00	0.00	4.50
		0.4	160.20	76.10	180.61	7.83	0.47	16.43	14.64	12.40	200.00	0.00	4.45
		0.7	224.44	109.61	258.27	13.35	0.36	19.63	19.00	18.04	200.00	0.00	5.15
	NE	0.1	109.57	77.68	121.22	50.14	0.61	8.95	11.10	7.75	200.00	0.00	6.43
		0.4	182.29	94.49	189.64	59.17	0.36	16.56	19.95	12.82	200.00	0.00	6.23
		0.7	241.22	110.20	266.90	81.79	0.56	10.96	9.35	8.33	200.00	0.00	4.81
Average			169.70	87.29	187.85	**36.19**	0.46	13.51	14.01	11.10	200.00	**0.00**	5.26
Total Average			154.66	88.10	145.03	**36.17**	0.26	7.92	8.61	5.99	125.00	**1.74**	2.39

unloading operations. In this chapter, we present efficient constructive heuristic approaches for addressing the Yard Crane Scheduling Problem (YCSP) outlined in Section 3. The constructive heuristic methods focus on designing the loading sequence of containers and assigning Input/Output (I/O) points. This results in the Simulated End-Time Rule (SETR) heuristic and its enhanced variant, a local search method ($SETR_{ls}$).

We conducted extensive experiments across 720 problem instances to evaluate all proposed approaches. The results demonstrate the strong performance of the proposed $SETR$ heuristic, which outperforms state-of-the-art rule-based methods such as TPR, $MTPR$, and NCR, as reported by Vallada *et al.* (2023). Notably, $SETR$ achieves more optimal solutions and significantly lower Relative Percentage Deviation (RPD) values. Moreover, $SETR_{ls}$ stands out as the most efficient method, achieving the lowest average RPD among all state-of-the-art approaches.

CHAPTER 5

Solving the YCSP with Greedy Randomized Adaptive Search Procedures (GRASP)

This chapter introduces metaheuristic methods, GRASP (Greedy Randomized Adaptive Search Procedure) to the yard crane scheduling problem in Chapter 3. Two GRASP approaches are proposed along with novel solution construction methods and the local search specifically tailored for this problem.

5.1 GRASP

GRASP (Greedy Randomized Adaptive Search Procedure), a well-established multi-start metaheuristic introduced by Feo and Resende (1995), is widely recognized for solving various combinatorial optimization problems. This method has been extensively explored in the literature, with numerous applications and variations. GRASP excels in scenarios where a constructive heuristic can be utilized and adapted to generate diverse solutions in each run. It has demonstrated notable

success in addressing crane scheduling problems, as evidenced by studies such as Kim and Park (2004) or Jung *et al.* (2006). For a comprehensive and up-to-date review of GRASP, interested readers are referred to Resende and Ribeiro (2016).

GRASP is popular in a series of scheduling problems. To name a few, Job Shop (Rajkumar *et al.*, 2011; Baykasoğlu *et al.*, 2020), Flow Shop (González-Neira and Montoya-Torres, 2017; Yepes-Borrero *et al.*, 2023), parallel machine (Yepes-Borrero *et al.*, 2020) and project scheduling (Alvarez-Valdés *et al.*, 2008); Vehicle Routing Problem (VRP) (Kontoravdis and Bard, 1995; Chaovalitwongse *et al.*, 2003). Some other applications in combinatorial problems can be also found in Traveling Salesman Problem (TSP) (Marinakis *et al.*, 2005); Graph Coloring Problem (Laguna and Martí, 2001); Set Covering Problem (Colombo *et al.*, 2015); P-median problem (Colmenar *et al.*, 2016); Facility Location Problem (Montoya-Torres *et al.*, 2011); Knapsack Problem (Yang *et al.*, 2013). These studies highlight the versatility of GRASP across a broad spectrum of combinatorial optimization problems.

In a nutshell, GRASP is an iterative process that combines both greedy and randomized approaches to generate high-quality solutions. At each iteration, GRASP involves constructing a solution and then refining it. Two operators of the GRASP method are the constructive method and solution-improving approaches. The latter often refers to local search procedures. The constructive method is the most essential component of GRASP. In the construction phase, GRASP builds a solution incrementally by following a greedy approach to select the best possible option at each step according to a predefined criterion. However, to introduce diversity and avoid the limitations of a purely greedy strategy, GRASP incorporates randomness by selecting from a list of top candidates, instead of choosing the best one. This blend of greediness and randomness helps the algorithm explore a broader

range of solution space.

Once a solution is constructed, GRASP moves to the local search phase to improve the constructed solution. Such a procedure finds improvements by exploring the solution neighborhood space, which is usually predefined by the local search. This search phase continues until no further improvements can be made, leading to a local optima. The process is repeated, and each iteration potentially yields a different solution due to the randomized element in the construction phase. Among all iterations, the best solution is selected as the final outcome. One of the strengths of GRASP lies in the ability to balance exploration and exploitation, making it highly adaptable and effective for solving complex optimization problems across a wide range of applications.

This chapter presents two simple constructive methods, leading to two tailored GRASP heuristic approaches specifically designed for this problem. The algorithms are statistically calibrated, and extensive computational studies demonstrate that the proposed GRASP method achieves optimal or near-optimal solutions for small and medium-sized instances. Additionally, it outperforms state-of-the-art Mixed Integer Programming (MIP) results, as well as rule-based heuristics, for larger instances.

The remainder of the chapter is structured as follows: Section 5.2 provides a detailed description of the proposed GRASP metaheuristic methods and the two constructive heuristic techniques used within these methods. Section 5.3 presents the comprehensive experiments conducted among all the testing instances. Finally, Section 5.4 summarizes conclusions and a discussion of the findings.

5.2 GRASP for the YCSP

Algorithm 2 provides a detailed description of the GRASP method designed to tackle the YCSP. The process begins by generating an initial

solution (π) using the deterministic constructive heuristic outlined in Section 4.3. This initial solution is then refined with local search to find the best current solution (π_{best}). This local search algorithm is exactly the same one proposed in Section 4.3.3 and is adopted in the main loop of the GRASP algorithm.

Algorithm 2 : GRASP metaheuristic

```
1: Output π_best
2: π := Constructive heuristic()
3: π_best := Local search(π)
4: while stopping criterion is not satisfied do
5:     π := Constructive method()
6:     π' := Local search(π)
7:     if f(π') < f(π_best) then
8:         π_best := π'
```


An effective construction method ensures extensive exploration of the solution space, creating a solid foundation for the subsequent local search phase. Consequently, we have introduced two distinct constructive methods, each contributing to different GRASP approaches detailed in the following sections.

5.2.1 GRASP constructive method 1

The first constructive method is an extension of the SETR heuristic proposed in Section 4.3. Algorithm 3 details the structure, which is iterative, greedy, and randomized. Here, the first container is randomly selected. In the main loop, the remaining containers are ordered according to the $SETR$ rule (Section 4.3) and the top/best α containers are selected to be part of the Restrictive Candidate List (RCL). One of the containers in the RCL is randomly selected and appended to the end of the container sequence. The process is repeated until a complete sequence of containers is obtained. At this point, the I/O assignment and timing are computed according to Section 4.3.2. This GRASP

method is referred to as $GR1$.

Algorithm 3 : GRASP constructive method 1

1: Output π
2: $\pi = \varnothing$
3: Randomly append container $c_0 \in C$ to π, $C \leftarrow C \setminus c_0$
4: **while** $C \neq \varnothing$ **do**
5: Sort all containers $c \in C$ in non-decreasing order of η_c (expression 4.1)
6: RCL := Pick the first α containers as the candidates of the RCL
7: π := Append one random container $r \in RCL$ to the end of π
8: $C \leftarrow C \setminus r$
9: I/O Assignment and Timing (π)

A quick example illustrates the constructive method 1 in the GRASP with five containers (Con1 to Con5) and three I/O points (IO1, IO2, and IO3) positioned on the same side. Figure 5.1 displays the solutions generated by constructive method 1 with a parameter setting of $\alpha = 2$, where modifications in each step are highlighted in bold. To initiate the sequencing process, a container (Con1) is randomly selected from the container set C. Next, the remaining containers are sorted based on their η values (expression 4.1), and Con3 and Con2 are identified as candidates for the RCL using $\alpha = 2$. From this RCL, Con3 is randomly chosen and appended to the sequence. At this stage, the partial solution is {Con1, Con3}, with the remaining container set updated to {Con2, Con4, Con5}. In subsequent steps, the containers in the updated set are sorted again by their η values, and the RCL is revised to include Con4 and Con5. From this list, Con4 is randomly selected and added to the end of the sequence. This process continues iteratively until all containers in set C have been included in the sequence. Ultimately, the final sequence constructed is {Con1, Con3, Con2, Con4, Con5}. Following the construction of the container sequence, I/O points are assigned to each container using the

I/O assignment methods detailed in Section 4.3.2.

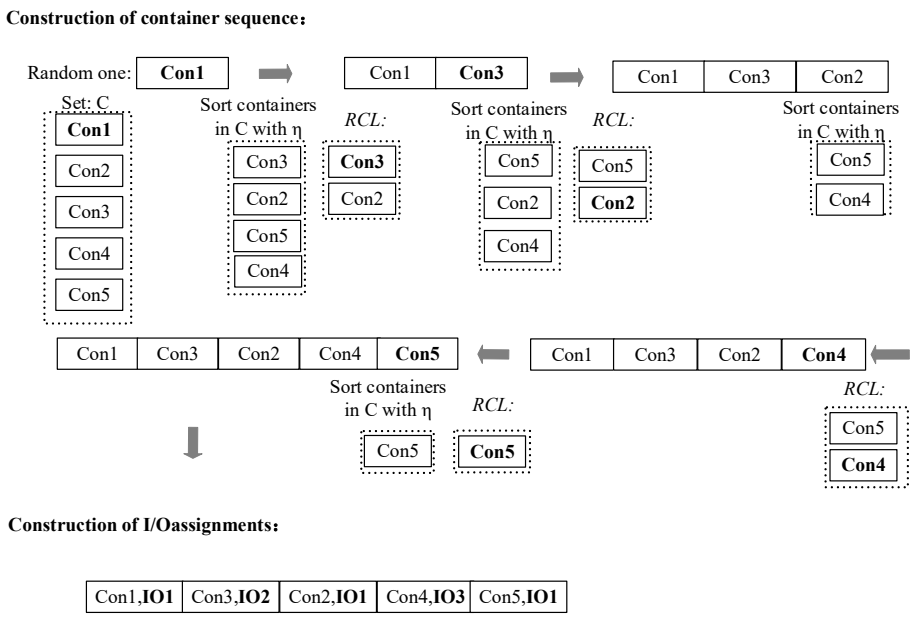


Figure 5.1: Example for the constructive method 1 with 5 containers and $\alpha = 2$. Elaborated by the author.

5.2.2 GRASP constructive method 2

One more GRASP algorithm, $GR2$, is introduced with a different constructive method described in Algorithm 4. Instead of assigning the I/O points for the final sequence, $GR2$ integrates both container sequencing and I/O decisions. More precisely, we maintain the original I/O during container sequencing and then reassign the I/O point for each newly added container by exploring alternative I/O options on the same side. The I/O assignment and timing of Section 4.3.2 are performed to update I/O points if the solution can be improved. Note that the additional adjustment of I/O points results in additional complexity, while it significantly improves the quality of the solutions obtained, as demonstrated in later sections.

Algorithm 4 : GRASP constructive method 2

Input: the incumbent solution π_0; Output: π
$\pi = \varnothing$
$C \leftarrow$ Preserve the I/O point for each container from π_0
Randomly append container $c_0 \in C$ to π, $C \leftarrow C \setminus c_0$
while $C \neq \varnothing$ **do**
    Sort all containers $c \in C$ in non-decreasing order of η_c maintaining the current I/O point (expression 4.1)
    $RCL :=$ Pick the first α containers as the candidates of the RCL
    $\pi :=$ Append one random container $r \in RCL$ to the end of π and replace its I/O point with the best one on this side
    $C \leftarrow C \setminus r$
Update I/O points using I/O Assignment and Timing (π)

A quick example in Figure 5.2 illustrates the GRASP operator of the constructive method 2 using the same instance as shown in Section 5.2.1 with modifications in bold. Differently, the I/O assignments of all containers in set C are preserved at the initial procedure of the construction of the container sequence. A random container Con1 is chosen from the container set C with the preserved I/O point IO1. Next, all the I/O options on this side will be tested to choose the best I/O point for Con1, and IO2 is finally selected and assigned to Con1. In the following steps, the values of η computed by the preserved I/O points are utilized to sort all the rest of the containers, and the Con3 and Con2 are chosen as the candidates of the RCL using $\alpha = 2$. Then, we assume that Con3 is randomly from the RCL and appended as the end of the sequence. We reassign the I/O point for Con3 using all the I/O points on the corresponding side where the best I/O option, e.g., IO1, is assigned to Con3. Gradually, all containers in C will be added to the container sequence, and for each newly added container, their containers will be reassigned with the best I/O options. As a final step, we use the I/O assignment methods in Section 4.3.2 to seek further improvements. The I/O assignments will be updated if the solution

can be improved therein.

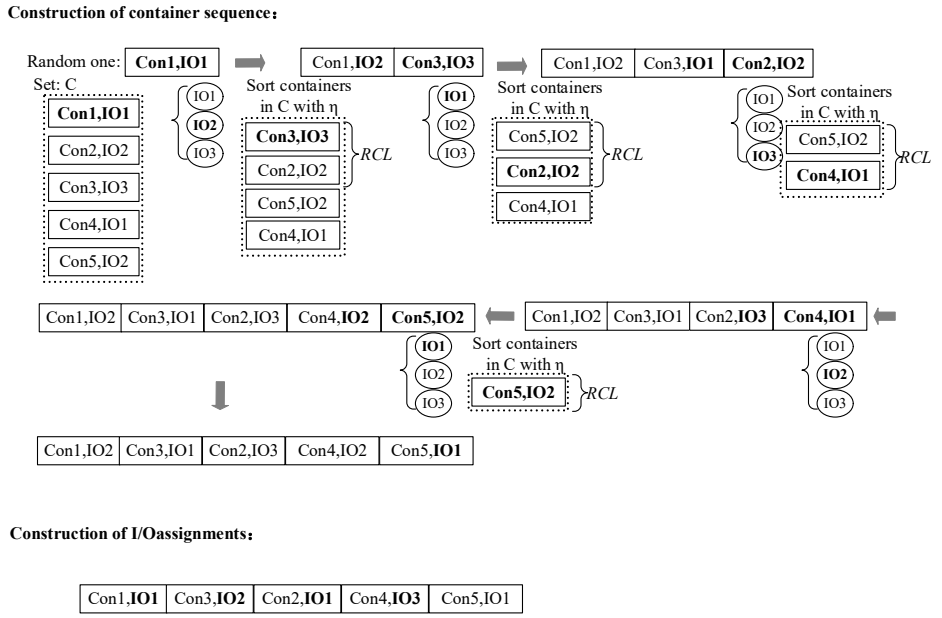


Figure 5.2: Example for the constructive method 2 with 5 containers and $\alpha = 2$. Elaborated by the author.

5.3 Computational experiments

In this section, comprehensive computational experiments are carried out using the benchmark instances introduced in Section 4.4.2. Moreover, a new set of calibration instances is generated in order to validate and tune the parameters of the GRASP algorithm and the local search proposed. In short, we calibrate the proposed methods with the Design Of Experiments (DOE) approach (Montgomery, 2017). All methods are coded in C# using Microsoft Visual Studio 2019 and .Net 6.0. To make more fair comparisons, the computational tests are randomly distributed to virtual machines under the same computation environment as described in Section 4.4.1.

5.3.1 New calibration instances

We carry out an extensive experimental campaign on two sets of benchmark instances including calibrations instances and testing instances. The calibration instances are used to calibrate the proposed methods, and the testing instances are used to evaluate the performance. It is a well-known practice that the calibration instances should be different from the testing instances to avoid bias and over-calibration for the proposed methods. Using the same instances for both calibration and final evaluation would compromise the integrity of the results, as the methods would be overfitted for the evaluation set.

Following the same methodology in Section 4.4.2, we regenerate two replicates of all 72 groups of instance configurations. It results in $72 \times 2 = 144$ calibration instances. As for the testing instances, we use the same 720 benchmark instances proposed by Vallada *et al.* (2023).

5.3.2 Calibration of the proposed GRASP

Rule selection for the constructive algorithm

Even though we have demonstrated that the $SETR$ rule outperforms other existing rules in Vallada *et al.* (2023), it is still necessary to observe how these rules perform in the proposed GRASP methods. We first validate if the proposed constructive heuristic ($SETR$) in section 4.3 is competitive. To test this, we compare it with the best deterministic rules TPR, $MTPR$ and NCR (Vallada *et al.*, 2023). We employ a single factor ANOVA to study if the differences observed between these rules have a statistically significant effect on the dependent variable, which is the Relative Percentage Deviation (RPD), calculated by expression 4.2. Moreover, a random rule ($RAND$) is also included as a baseline. It simply calculates the best solution among 5 random solutions, where each solution consists of a random container sequence, and uses the same I/O assignment methods applied to the other rules.

Figure 5.3 presents the means plot with Tukey's Honestly Significant Difference (HSD) intervals at the 95% confidence for the various rules. Recall that if two intervals are overlapped, their differences are statistically insignificant. In this case, no overlaps are observed among the various rules that are expected. It is not surprising that the $RAND$ rule performs the worst. Conversely, the proposed $SETR$ algorithm achieves the lowest RPD values, which are statistically better than other alternatives, making it the preferred choice for the constructive phase of the GRASP methods.

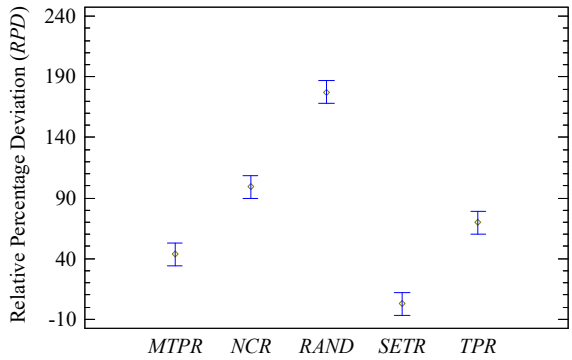


Figure 5.3: Means and Tukey HSD intervals at the 95% confidence level for the proposed $SETR$ rule, comparing rules from the literature and a baseline random rule. Elaborated by the author.

β selection of the local search

The local search method proposed in Section 4.3.3 utilizes a single parameter β. It determines the number of solutions evaluated with the real objective function, which includes the I/O assignment (Section 4.3.2). Four values of β are tested: 1, 3 , 4 and 5. Figure 5.4 presents the corresponding means plot at the 95% confidence intervals. For both $GR1$ and $GR2$, the intervals for β values of 3, 4, and 5 overlap, indicating that there is no statistically significant difference among these values. Thus, $\beta = 3$ is chosen for both GRASP methods. It is

important to note that these values outperform $\beta = 1$, demonstrating that the specialized design for evaluating multiple solutions yields better results than the single evaluation approach ($\beta = 1$), leading to a more effective local search method for the GRASP algorithms.

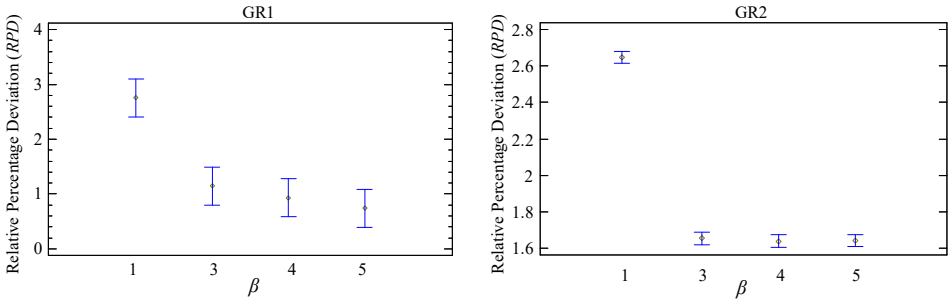


Figure 5.4: Means and Tukey HSD intervals at the 95% confidence level for the tested β values of the proposed local search in $GR1$ (left) and $GR2$ (right). Elaborated by the author.

Size of the RCL (α)

The parameter α determines the size of the candidate list (RCL) used to generate solutions in the constructive phase of the GRASP algorithms. We calibrated α with values ranging from 1 to 7. Figure 5.5 presents the corresponding means plot at the 95% confidence intervals for both $GR1$ and $GR2$. As is typical in GRASP, an α value of 1, where the best candidate is deterministically selected, produces the worst results performing similarly as parameter β. For $GR1$, values of 2 and 3 yield the best performance, while for $GR2$, values of 2, 3, and 4 are statistically equivalent due to overlapping confidence intervals. Consequence, we select $\alpha = 2$ for $GR1$ and $\alpha = 3$ for $GR2$.

5.3.3 Computational results

Comprehensive experiments are executed on the evaluation of the proposed GRASP algorithms in the same computational environment

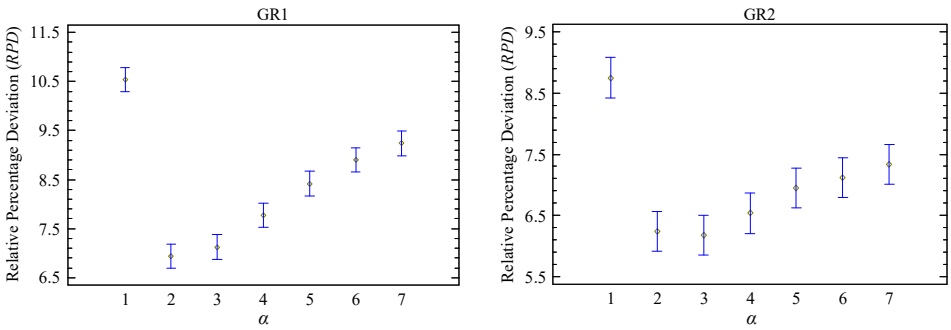


Figure 5.5: Means and Tukey HSD intervals at the 95% confidence level for the tested α values of the proposed $GR1$ (left) and $GR2$ (right). Elaborated by the author.

in Section 4.3. Both proposed GRASP methods are calibrated with 144 calibration instances in Section 5.3.1 and then are evaluated with 720 testing benchmark instances of Section 4.4.2.

The GRASP algorithms are compared against the TPR_{ls}, $MTPR_{ls}$, and NCR_{ls} rules proposed by Vallada *et al.* (2023), as well as the proposed $SETR_{ls}$ method, which combines the $SETR$ constructive approach with local search in Section 4.3.3. Note that $SETR$ is a deterministic heuristic used both as a standalone method and as the starting solution for GRASP (line 3 of Algorithm 2). Additionally, a randomized version of $SETR$ has been adapted as the constructive method in the proposed GRASP (line 5 of Algorithm 2). As for the calibration experiments, the GRASP algorithms are executed independently $R = 5$ times. All GRASP methods run at the CPU time within $n \times \ln n \times m \times \tau$ milliseconds where n and m is the number of containers and I/O points respectively, and τ is CPU time at levels of $\tau = \{10, 20, 30\}$, resulting in CPU times ranging from 0.24 seconds to 8.48 minutes. Using different τ levels, we can evaluate the impact of increased CPU time on the performance of the GRASP algorithms.

Computational results for small and small tight instances

The solutions obtained by the proposed methods can be compared to the optimal solutions generated by the mathematical models in Vallada *et al.* (2023). Tables 5.1 and 5.2 present the number of optimal solutions achieved by each method for small and small tight instances, respectively. For small instances, $GR1$ and $GR2$ achieve 74 (62%) and 80 (67%) optimal solutions, while for small tight instances, they obtain 88 (73%) and 91 (76%) optimal solutions respectively, at all the CPU time levels. In fact, both methods produce consistent results across these instance sets. In contrast, the best heuristic rules achieve only 64 optimal solutions (53%) for small instances and 49 (41%) for the small tight instances.

Table 5.1: Number of optimal solutions obtained by different algorithms for small instances with *Equal* (E) and *Non Equal* (NE) weights (w_j) grouped by parameter ρ.

n	w_j	ρ	#	TPR_{ls}	$MTPR_{ls}$	NCR_{ls}	$SETR_{ls}$	$GR1_{10}$	$GR2_{10}$	$GR1_{20}$	$GR2_{20}$	$GR1_{30}$	$GR2_{30}$
5	E	0.1	10	8	8	9	6	6	9	6	9	6	9
		0.4	10	8	6	10	9	10	10	10	10	10	10
		0.7	10	6	8	8	8	9	9	9	9	9	9
	NE	0.1	10	6	7	7	4	7	8	7	8	7	8
		0.4	10	8	7	9	6	7	8	7	8	7	8
		0.7	10	8	8	7	8	10	10	10	10	10	10
10	E	0.1	10	2	0	0	1	3	3	3	3	3	3
		0.4	10	0	1	3	1	5	5	5	5	5	5
		0.7	10	1	1	1	2	4	5	4	5	4	5
	NE	0.1	10	3	1	2	1	4	4	4	4	4	4
		0.4	10	3	2	2	1	3	3	3	3	3	3
		0.7	10	3	3	6	2	6	6	6	6	6	6
	Total		120	56	52	64	49	74	**80**	74	**80**	74	**80**

To compare the performance of the rules and GRASP algorithms, the average RPD values are presented in Tables 5.3 and 5.4, along with the average CPU times (in seconds) for each algorithm. $GR1_\tau$ and $GR2_\tau$, where $\tau = \{10, 20, 30\}$, refer to $GR1$ and $GR2$ using different CPU time limits as stopping criteria. For small instances, the proposed $SETR_{ls}$ heuristic cannot outperform NCR_{ls} in terms of RPD, but it is significantly faster, with an average time of 0.06 seconds compared to

Table 5.2: Number of optimal solutions obtained by different algorithms for small tight instances with *Equal* (E) and *Non Equal* (NE) weights (w_j) grouped by parameter ρ.

n	w_j	ρ	#	TPR_{ls}	$MTPR_{ls}$	NCR_{ls}	$SETR_{ls}$	$GR1_{10}$	$GR2_{10}$	$GR1_{20}$	$GR2_{20}$	$GR1_{30}$	$GR2_{30}$
5	E	0.1	10	3	3	7	9	10	10	10	10	10	10
		0.4	10	6	5	5	7	9	9	9	9	9	9
		0.7	10	7	6	5	6	9	10	9	10	9	10
	NE	0.1	10	6	5	5	3	8	8	8	8	8	8
		0.4	10	8	7	7	8	9	9	9	9	9	9
		0.7	10	7	7	8	7	8	8	8	8	8	8
10	E	0.1	10	1	0	2	2	8	8	8	8	8	8
		0.4	10	1	1	0	2	5	5	5	5	5	5
		0.7	10	3	1	3	2	7	7	7	7	7	7
	NE	0.1	10	0	0	2	0	4	6	4	6	4	6
		0.4	10	2	1	2	1	5	5	5	5	5	5
		0.7	10	1	2	3	2	6	6	6	6	6	6
	Total		120	45	38	49	49	88	**91**	88	**91**	88	**91**

7.50 seconds for NCR_{ls}. As for the GRASP algorithms, $GR1$ delivers an RPD that is 3.5 times lower than the best constructive heuristic (NCR_{ls}), while $GR2$ achieves an even greater reduction, 4.5 times lower, regardless of the value of τ. With respect to small tight instances, $SETR_{ls}$ outperforms NCR_{ls} while being 125 times faster. Even in the worst case with $\tau = 10$, $GR1$ improves on the best heuristic by a factor of 5.43, and $GR2$ by 5.75, with minimal additional CPU time (0.47 seconds vs. 0.06 seconds on average). It is noteworthy that for both small and small tight instances, the GRASP algorithms produce the same results regardless of the CPU times.

Table 5.3: Average RPD (%) over optimal solutions and CPU times (s) for small instances with *Equal* (E) and *Non Equal* (NE) weights (w_j) grouped by parameter ρ.

n	w_j	ρ	TPR_{ls}	$MTPR_{ls}$	NCR_{ls}	$time$	$SETR_{ls}$	$time$	$GR1_{10}$	$GR2_{10}$	$time$	$GR1_{20}$	$GR2_{20}$	$time$	$GR1_{30}$	$GR2_{30}$	$time$
5	E	0.1	0.96	0.25	1.30	5.00	0.73	0.05	0.73	0.14	1.29	0.73	0.14	2.58	0.73	0.14	3.86
		0.4	4.77	2.16	0.00	5.00	0.33	0.05	0.00	0.00	1.29	0.00	0.00	2.58	0.00	0.00	3.86
		0.7	5.31	2.76	0.76	5.00	2.23	0.05	0.13	0.13	1.29	0.13	0.13	2.58	0.13	0.13	3.86
	NE	0.1	0.35	0.26	0.26	5.00	1.71	0.06	0.22	0.12	1.29	0.22	0.12	2.58	0.22	0.12	3.86
		0.4	2.44	2.76	0.54	5.00	2.45	0.05	0.82	0.73	1.29	0.82	0.73	2.58	0.82	0.73	3.86
		0.7	2.37	2.37	2.31	5.00	0.76	0.05	0.00	0.00	1.29	0.00	0.00	2.58	0.00	0.00	3.86
10	E	0.1	3.27	4.73	4.41	10.00	2.99	0.06	0.54	0.41	3.68	0.54	0.40	7.37	0.54	0.40	11.05
		0.4	4.48	3.99	2.07	10.00	4.90	0.06	0.37	0.37	3.68	0.37	0.37	7.37	0.37	0.37	11.05
		0.7	4.69	8.22	4.06	10.00	2.32	0.06	0.87	0.49	3.68	0.87	0.41	7.37	0.87	0.41	11.05
	NE	0.1	2.11	7.04	2.54	10.00	3.96	0.06	1.39	1.33	3.68	1.39	1.23	7.37	1.39	1.23	11.05
		0.4	1.45	6.94	2.40	10.00	1.72	0.06	0.66	0.66	3.68	0.66	0.66	7.37	0.66	0.63	11.05
		0.7	2.88	4.29	3.24	10.00	4.23	0.07	1.13	1.13	3.68	1.13	1.13	7.37	1.13	1.13	11.05
Total Average			2.92	3.81	1.99	7.50	2.36	0.06	0.57	**0.46**	2.48	0.57	**0.44**	4.97	0.57	**0.44**	7.46

Table 5.4: Average RPD (%) over optimal solutions and CPU times (s) for small tight instances with *Equal* (E) and *Non Equal* (NE) weights (w_j) grouped by parameter ρ.

n	w_j	ρ	TPR_{ls}	$MTPR_{ls}$	NCR_{ls}	$time$	$SETR_{ls}$	$time$	$GR1_{10}$	$GR2_{10}$	$time$	$GR1_{20}$	$GR2_{20}$	$time$	$GR1_{30}$	$GR2_{30}$	$time$
5	E	0.1	2.56	2.05	1.19	5.00	0.06	0.05	0.00	0.00	0.24	0.00	0.00	0.48	0.00	0.00	0.72
		0.4	2.22	3.71	2.52	5.00	1.05	0.06	0.10	0.10	0.24	0.10	0.10	0.48	0.10	0.10	0.72
		0.7	2.38	2.17	7.56	5.00	4.41	0.05	0.06	0.00	0.24	0.06	0.00	0.48	0.06	0.00	0.72
	NE	0.1	1.82	2.50	1.98	5.00	4.48	0.06	0.92	0.89	0.24	0.92	0.89	0.48	0.92	0.89	0.72
		0.4	0.83	2.15	3.68	5.00	1.90	0.06	0.45	0.45	0.24	0.45	0.45	0.48	0.45	0.45	0.72
		0.7	1.39	1.39	1.26	5.00	1.39	0.05	1.26	1.26	0.24	1.26	1.26	0.48	1.26	1.26	0.72
10	E	0.1	6.44	7.31	5.04	10.00	6.35	0.06	0.64	0.64	0.69	0.64	0.64	1.38	0.64	0.64	2.07
		0.4	6.74	8.72	12.06	10.00	4.16	0.05	0.84	0.80	0.69	0.84	0.80	1.38	0.84	0.80	2.07
		0.7	7.65	10.51	6.44	10.00	4.36	0.05	1.26	1.26	0.69	1.26	1.26	1.38	1.26	1.26	2.07
	NE	0.1	6.99	8.49	2.65	10.00	5.43	0.06	0.95	0.59	0.69	0.95	0.57	1.38	0.95	0.57	2.07
		0.4	5.08	10.41	3.63	10.00	5.75	0.06	1.19	1.19	0.69	1.19	1.19	1.38	1.19	1.19	2.07
		0.7	8.50	9.06	4.82	10.00	6.23	0.06	0.80	0.80	0.69	0.80	0.80	1.38	0.80	0.80	2.07
Total Average			4.38	5.71	4.40	7.50	3.80	0.06	0.70	**0.66**	0.47	0.70	**0.66**	0.93	0.70	**0.66**	1.39

Computational results for medium instances

Table 5.5 presents the results of all the medium instances. The best performance is achieved by $GR2$ with $\tau = 30$. In all cases, the GRASP method consistently outperforms the heuristics, with lower average RPD values, regardless of the τ value. Additionally, performance improves as more CPU time is allocated. It is also worth noting that the proposed $SETR_{ls}$ heuristic generally outperforms the other approaches from the literature, achieving better results in most cases while using only a fraction of the CPU time.

Computational results for large instances

For large instances, the performance comparison follows a similar pattern to that of the medium instances. As shown in Table 5.6, both GRASP methods outperform the state-of-the-art heuristics, with the newly proposed $GR2$ significantly surpassing $GR1$. The best results are obtained with $GR2_{30}$, achieving an average RPD of 0.28%. Although its computational time is higher than that of the best existing method, NCR_{ls}, $GR2_{10}$ provides a compelling alternative, delivering an average RPD of 0.86 compared to 11.16 for NCR_{ls}, while using only 98.70 seconds on average versus 125 seconds for NCR_{ls}.

Table 5.5: Average RPD (%) over the best known solution and CPU times (s) for medium instances with *Equal* (E) and *Non Equal* (NE) weights (w_j) grouped by parameter ρ.

n	w_j	ρ	TPR_{ls}	$MTPR_{ls}$	NCR_{ls}	$time$	$SETR_{ls}$	$time$	$GR1_{10}$	$GR2_{10}$	$time$	$GR1_{20}$	$GR2_{20}$	$time$	$GR1_{30}$	$GR2_{30}$	$time$
15	E	0.1	6.75	6.46	5.57	15.00	4.81	0.07	0.72	0.69	6.50	0.72	0.69	13.00	0.72	0.50	19.50
		0.4	8.67	5.34	7.17	15.00	5.73	0.06	0.06	0.06	6.50	0.06	0.06	13.00	0.06	0.06	19.50
		0.7	6.99	6.22	6.41	15.00	4.08	0.07	0.10	0.10	6.50	0.10	0.10	13.00	0.10	0.10	19.50
	NE	0.1	3.52	5.03	2.01	15.00	4.17	0.06	0.32	0.28	6.50	0.31	0.28	13.00	0.31	0.28	19.50
		0.4	11.13	7.25	8.64	15.00	7.49	0.07	0.00	0.00	6.50	0.00	0.00	13.00	0.00	0.00	19.50
		0.7	4.01	12.19	9.71	15.00	10.01	0.07	0.04	0.04	6.50	0.04	0.04	13.00	0.04	0.04	19.50
Average			6.85	7.08	6.59	15.00	6.05	0.07	0.21	**0.20**	6.50	0.20	**0.20**	13.00	0.20	**0.16**	19.50
20	E	0.1	6.14	5.27	2.95	20.00	4.46	0.09	0.53	0.41	9.59	0.43	0.41	19.17	0.43	0.40	28.76
		0.4	7.95	5.92	7.87	20.00	4.55	0.08	0.06	0.01	9.59	0.02	0.00	19.17	0.02	0.00	28.76
		0.7	12.81	14.70	10.62	20.00	8.43	0.07	0.62	0.54	9.59	0.58	0.48	19.17	0.58	0.44	28.76
	NE	0.1	5.88	8.53	4.47	20.00	5.93	0.09	0.15	0.09	9.59	0.03	0.04	19.17	0.03	0.01	28.76
		0.4	8.71	11.87	6.72	20.00	10.11	0.10	0.24	0.21	9.59	0.15	0.14	19.17	0.04	0.11	28.76
		0.7	9.78	15.82	11.94	20.00	12.53	0.09	0.81	0.24	9.59	0.72	0.24	19.17	0.14	0.20	28.76
Average			8.54	10.35	7.43	20.00	7.67	0.09	0.40	**0.25**	9.59	0.32	**0.22**	19.17	0.21	**0.19**	28.76
30	E	0.1	7.10	8.84	8.57	30.00	5.09	0.14	0.42	0.46	16.33	0.26	0.30	32.65	0.22	0.11	48.98
		0.4	12.21	8.46	9.72	30.00	10.70	0.13	0.67	0.49	16.33	0.43	0.29	32.65	0.32	0.18	48.98
		0.7	11.49	10.80	12.32	30.00	7.74	0.14	0.53	0.55	16.33	0.12	0.40	32.65	0.03	0.34	48.98
	NE	0.1	7.68	8.55	4.01	30.00	9.20	0.13	1.08	0.79	16.33	0.81	0.50	32.65	0.73	0.39	48.98
		0.4	8.36	8.17	7.63	30.00	10.88	0.16	0.69	0.57	16.33	0.29	0.34	32.65	0.26	0.21	48.98
		0.7	14.96	16.31	11.64	30.00	10.46	0.15	0.69	0.22	16.33	0.58	0.19	32.65	0.52	0.08	48.98
Average			10.30	10.19	8.98	30.00	9.01	0.14	0.68	**0.51**	16.33	0.42	**0.34**	32.65	0.35	**0.22**	48.98
40	E	0.1	11.88	12.75	11.68	40.00	7.75	0.24	1.28	0.66	23.61	0.90	0.55	47.22	0.83	0.34	70.83
		0.4	16.66	10.41	14.26	40.00	11.02	0.24	0.73	1.15	23.61	0.43	0.74	47.22	0.32	0.26	70.83
		0.7	17.90	13.26	17.41	40.00	11.36	0.22	1.05	0.97	23.61	0.79	0.55	47.22	0.60	0.16	70.83
	NE	0.1	6.78	7.01	6.09	40.00	6.53	0.24	0.33	0.27	23.61	0.24	0.14	47.22	0.18	0.09	70.83
		0.4	8.45	11.89	7.67	40.00	10.01	0.22	0.76	0.70	23.61	0.51	0.44	47.22	0.40	0.07	70.83
		0.7	9.75	13.21	10.26	40.00	16.27	0.22	1.07	0.89	23.61	0.42	0.60	47.22	0.30	0.40	70.83
Average			11.90	11.42	11.23	40.00	10.49	0.23	0.87	**0.77**	23.61	0.55	**0.50**	47.22	0.44	**0.22**	70.83
Total Average			9.40	9.76	8.56	26.25	8.30	0.13	0.54	**0.43**	14.01	0.37	**0.31**	28.01	0.30	**0.20**	42.02

Table 5.6: Average RPD over the best known solution and CPU times (s) for large instances with *Equal* (E) and *Non Equal* (NE) weights (w_j) grouped by parameter ρ.

n	w_j	ρ	TPR_{ls}	$MTPR_{ls}$	NCR_{ls}	$time$	$SETR_{ls}$	$time$	$GR1_{10}$	$GR2_{10}$	$time$	$GR1_{20}$	$GR2_{20}$	$time$	$GR1_{30}$	$GR2_{30}$	$time$
50	E	0.1	8.94	12.38	9.14	50.00	6.78	0.35	1.33	0.89	31.30	1.00	0.46	62.59	0.68	0.34	93.89
		0.4	12.59	9.98	13.21	50.00	11.86	0.31	1.22	0.86	31.30	0.75	0.41	62.59	0.56	0.33	93.89
		0.7	15.08	16.03	12.87	50.00	11.39	0.31	1.43	0.82	31.30	1.11	0.57	62.59	0.94	0.44	93.89
	NE	0.1	6.33	6.02	4.51	50.00	6.47	0.40	1.02	1.00	31.30	0.74	0.56	62.59	0.50	0.26	93.89
		0.4	9.35	9.19	7.50	50.00	9.55	0.33	1.00	1.23	31.30	0.70	0.71	62.59	0.30	0.53	93.89
		0.7	13.54	14.18	9.46	50.00	13.30	0.37	1.08	1.54	31.30	0.63	0.57	62.59	0.51	0.23	93.89
Average			10.97	11.30	9.45	50.00	9.89	0.35	1.18	**1.06**	31.30	0.82	**0.55**	62.59	0.58	**0.36**	93.89
100	E	0.1	8.77	10.73	7.35	100.00	3.66	1.03	1.63	0.48	73.68	1.26	0.23	147.37	1.23	0.09	221.05
		0.4	13.55	14.26	12.78	100.00	6.75	1.28	2.54	0.85	73.68	1.99	0.53	147.37	1.68	0.10	221.05
		0.7	18.04	16.71	15.75	100.00	10.90	1.07	1.45	1.69	73.68	1.06	0.84	147.37	0.98	0.65	221.05
	NE	0.1	6.60	7.44	4.70	100.00	4.38	1.15	0.32	0.42	73.68	0.32	0.27	147.37	0.14	0.17	221.05
		0.4	9.71	9.17	5.26	100.00	7.12	1.11	0.88	0.82	73.68	0.69	0.46	147.37	0.50	0.40	221.05
		0.7	5.74	8.07	3.57	100.00	8.64	1.30	2.03	1.68	73.68	1.35	1.03	147.37	1.18	0.78	221.05
Average			10.40	11.06	8.23	100.00	6.91	1.16	1.47	**0.99**	73.68	1.11	**0.56**	147.37	0.95	**0.36**	221.05
150	E	0.1	9.01	9.84	9.03	150.00	3.15	2.87	1.91	0.21	120.26	1.64	0.10	240.51	1.62	0.00	360.77
		0.4	13.38	13.35	13.27	150.00	4.55	3.06	1.90	0.40	120.26	1.65	0.01	240.51	1.60	0.01	360.77
		0.7	19.18	18.91	18.29	150.00	8.54	2.77	2.01	1.37	120.26	1.29	1.16	240.51	0.82	0.52	360.77
	NE	0.1	8.54	9.26	6.88	150.00	3.14	2.47	0.97	0.80	120.26	0.47	0.50	240.51	0.30	0.32	360.77
		0.4	12.83	16.91	7.44	150.00	4.22	2.97	1.05	0.78	120.26	0.57	0.28	240.51	0.48	0.17	360.77
		0.7	11.75	14.23	8.79	150.00	7.63	2.65	1.19	1.08	120.26	0.78	0.70	240.51	0.52	0.64	360.77
Average			12.45	13.75	10.62	150.00	5.20	2.80	1.51	**0.77**	120.26	1.07	**0.46**	240.51	0.89	**0.28**	360.77
200	E	0.1	10.96	12.53	9.67	200.00	2.26	4.50	2.72	0.45	169.55	2.40	0.15	339.09	2.14	0.00	508.64
		0.4	21.22	19.34	17.03	200.00	4.13	4.45	2.65	0.48	169.55	2.29	0.14	339.09	1.99	0.00	508.64
		0.7	29.34	28.70	27.58	200.00	8.11	5.15	2.75	0.74	169.55	2.12	0.47	339.09	1.60	0.14	508.64
	NE	0.1	10.46	12.64	9.25	200.00	1.39	6.43	0.57	0.29	169.55	0.31	0.16	339.09	0.22	0.03	508.64
		0.4	21.40	24.92	17.50	200.00	4.18	6.23	1.34	0.55	169.55	0.91	0.33	339.09	0.33	0.20	508.64
		0.7	19.74	18.06	16.94	200.00	7.96	4.81	1.17	1.21	169.55	0.61	0.62	339.09	0.28	0.41	508.64
Average			18.85	19.36	16.33	200.00	4.67	5.26	1.86	**0.62**	169.55	1.44	**0.31**	339.09	1.09	**0.13**	508.64
Total Average			13.17	13.87	11.16	125.00	6.67	2.39	1.51	**0.86**	98.70	1.11	**0.47**	197.39	0.88	**0.28**	296.09

Comparison between different methods

We now compare all the methods across the benchmark instances. Figure 5.6 depicts the interaction between the number of containers and the average RPD of the algorithms, based on a two-factor ANOVA analysis. This statistical approach allows us to observe the effects of instance size and algorithm choice on performance. The GRASP methods ($GR1_{30}$ and $GR2_{30}$) consistently achieve the lowest RPD across all instance sizes, highlighting their robust performance.

An interesting trend emerges for instances with 50 or more containers. The performance of methods in Vallada *et al.* (2023) show a clear decline as instance size increases, while the proposed $SETR_{ls}$ heuristic demonstrates the opposite behavior. Its performance improves relative to other deterministic heuristics, thanks to the more efficient and effective local search technique it employs. This advantage becomes particularly evident in larger instances, where $SETR_{ls}$ proves stronger than the approaches used in Vallada *et al.* (2023).

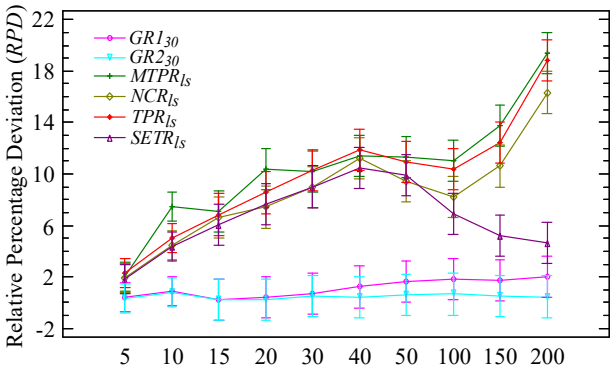


Figure 5.6: Interaction and Tukey HSD intervals at the 95% confidence level between the number of containers (n) and RPD for the algorithms. Elaborated by the author.

As noted in Section 4.3, all methods were tested under the same computational environment. Figure 5.7 illustrates the average CPU

times (in seconds) for the algorithms as a function of the number of containers, n. Only one rule from Vallada *et al.* (2023), namely NCR_{ls}, is depicted in the figure, as the CPU times for all three rules are identical. The proposed $SETR_{ls}$ heuristic consistently needs the minimum CPU time, regardless of instance size. In contrast, the computational time for the other algorithms significantly increases when handling 50 or more containers. Notably, the fastest GRASP version, with $\tau = 10$, outperforms NCR_{ls} not only in speed but also in result quality. While the other two GRASP variations take more time, they achieve even better RPD values.

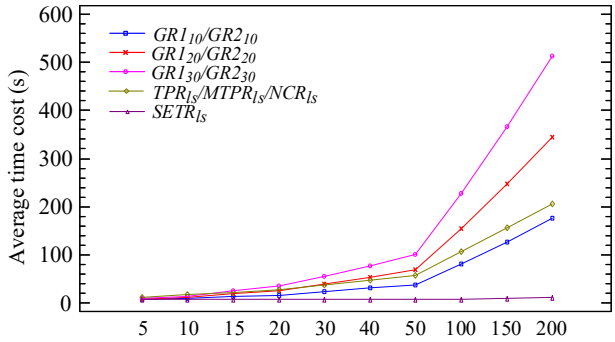


Figure 5.7: Average time (in seconds) according to the number of containers (n). Elaborated by the author.

5.3.4 Sensitivity analysis

Even though GRASP has been demonstrated to outperform the other approaches, it is still worth evaluating how it performs using various I/O assignments and weight settings.

Sensitivity analysis on I/O assignments

We compare the proposed $GR2$ method, referred to as $GR_{AssignIO}$, with two variants that utilize fixed I/O assignments from Section 4.3.2: GR_{RandIO} and $GR_{NearestIO}$. All three methods are tested on the same

instances with a CPU time limit of $\tau = 10$. Figure 5.8 and 5.9 display the means and Tukey HSD intervals and corresponding interaction of the RPD at the 95% confidence level, respectively. The results clearly show that $GR_{AssignIO}$ significantly outperforms the other methods. Specifically, using random I/O points leads to decreased performance as the number of containers increases, while the nearest I/O points exhibit the opposite trend. These findings support the intuition that dynamic I/O assignments help distribute the workload more effectively, thereby reducing congestion and delays. Similar observations suggest that the proposed I/O assignment also contributes to a lower makespan, which means the dynamic I/O assignments adopted are helpful in decreasing the loading time of the yard crane.

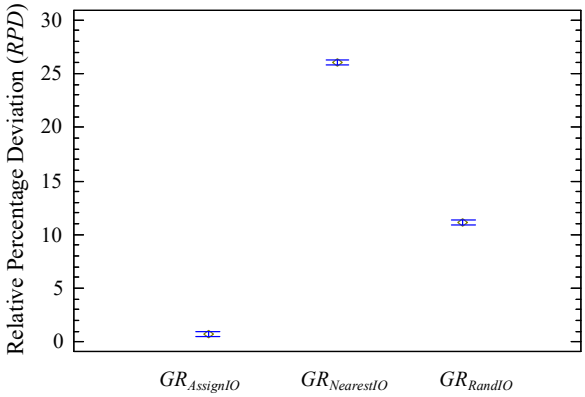


Figure 5.8: Means and Tukey HSD intervals of RPD on the objective function at the 95% confidence level for three I/O assignment alternatives. Elaborated by the author.

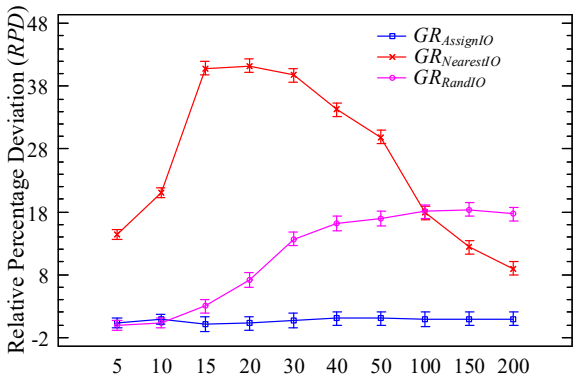


Figure 5.9: Interaction and Tukey HSD intervals of RPD on the objective function at the 95% confidence level between the number of containers (n) and for the three I/O assignment alternatives. Elaborated by the author.

Sensitivity analysis of objective function weights

Sensitivity analysis helps to investigate how the performance can be affected by corresponding factors. We first analyze the influence of various weight settings on the proposed methods. Four groups of weight configurations are tested: one equal (E) set and three non-equal (NE) sets. Each configuration comprises the 720 instances described in Section 5.3.1, but with different weight assignments. In the E set, all containers are assigned a weight of 1. For the NE sets, we consider three different weight distributions:

1. NE_DC: Weights for Delay and congestion. Two weight pairs (1,3) and (3,1) are used for the delay (w_c^R) and congestion (w_c^C).

2. NE_SL: Weights for seaside and landside containers. Weight pairs (1,3) and (3,1) are fixed to the seaside ($c \in \{1, 3\}$) and landside ($c \in \{2, 4\}$) containers with identical weights for delay and congestion.

3. NE_Rand: Random weights. All weights are randomly set between 1 and 3.

We conduct another ANOVA experiment using $GR2$ to examine the effects of equal and non-equal weight settings. Figure 5.10 (left) displays the means and Tukey HSD intervals at the 95% confidence level for the methods tested under equal and non-equal conditions. For a CPU time of $\tau = 10$, the proposed $GR2$ significantly outperforms all other methods, with no notable difference observed between equal and non-equal weights. A similar pattern is seen with TPR_{ls}, $MTPR_{ls}$, and $SETR_{ls}$; however, NCR_{ls} performs better under non-equal weights. As illustrated in Figure 5.10 (right), $GR2$ continues to clearly outperform all other methods across the three non-equal weight settings. Overall, the proposed GRASP method demonstrates robustness across all tested weight configurations.

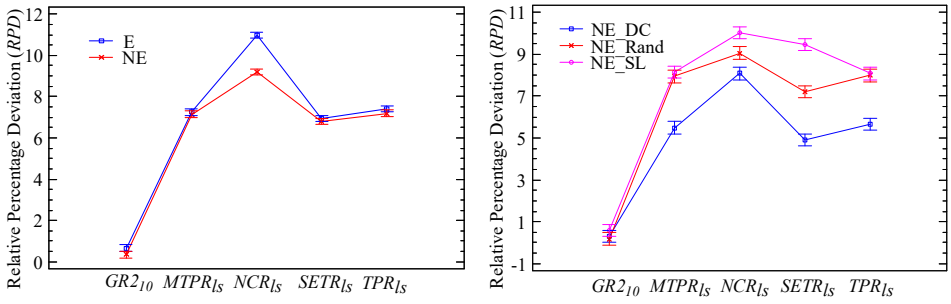


Figure 5.10: Interaction and Tukey HSD intervals for the tested methods under equal (E) and non-equal (NE) weight settings (left) and under 3 kinds of non-equal weight settings (rights) at the 95% confidence level. Elaborated by the author.

5.4 Conclusions

In this chapter, we present two efficient metaheuristics, Greedy Randomized Adaptive Search Procedures (GRASP) $GR1$ and $GR2$. They operate in an iterative two-phase process including solution construction and local search. We propose new constructive methods for each

GRASP method. $GR1$ construct the container sequence with the Simulated End-time Rule (SETR) and assign I/O points with the assignment methods in Section 4.3. While $GR2$ integrates both procedures, utilizing the previous I/O points.

Comprehensive calibration and computational testing demonstrate that the proposed GRASP statistically outperform the state-of-the-art methods (Vallada *et al.*, 2023), including deterministic heuristic and local search procedures ($SETR_{ls}$) in Section 4.3. The GRASP methods manage to obtain an optimality average rate of 71.25% for small instances and find 461 better new best solutions in the 480 medium and large instances, with up to 200 containers. Both GRASP algorithms significantly surpass current heuristics, with $GR2$ consistently achieving superior results across all CPU time settings.

CHAPTER 6

Solving the YCSP with Iterated Greedy methods

This chapter introduces a series of simple yet powerful, Iterated Greedy (IG) methods to the Yard Crane Scheduling Problem (YCSP) in Section 3. We propose several variations of the destruction and reconstruction operators, coordination with novel local search procedures and acceleration techniques. The proposed IG methods are carefully calibrated and evaluated using comprehensive computational experiments. The results indicate that small changes in the features of the IG method have a profound impact on performance, resulting in statistically better approaches. We compare the proposed IG method against the state-of-the-art approaches including the heuristic methods in Chapter 4 and GRASP approaches in Chapter 5. It turns out that the proposed IG methods outperform all other proposals by a wide margin across from small instances to large instances with up to 200 containers.

6.1 Iterated Greedy (IG)

Iterated Greedy (IG), first proposed by Ruiz and Stützle (2007), is a simple yet effective metaheuristic for solving combinatorial optimization problems. In a nutshell, IG operates through two main phases: destruction and reconstruction. In the destruction phase, part of the current schedule is removed, introducing randomness that helps escape local optima. In the reconstruction phase, a greedy approach reintroduces the removed items based on local optimization criteria, while local search techniques can optionally be incorporated to further refine the solution. These procedures allow the IG method to efficiently explore different solution spaces without the comprehensive computation of more sophisticated algorithms.

The popularity of IG methods stems from their simplicity, flexibility, and outperformance, making them highly effective for solving complex optimization problems. The simplicity lies in its straightforward and intuitive process of destruction and reconstruction. It is a basic but effective mechanism to explore and improve solutions. These phases are iterated over time to improve a solution without the requirement of sophisticated operators, like crossover and mutation in genetic algorithms or tabu lists in tabu search. The flexibility of IG allows it to be tailored to the specific characteristics of different optimization problems, making it suitable for a wide range of problems. For example, in Flowshop scheduling, IG can optimize job sequences to minimize makespan, while in parallel machine scheduling, it can balance workloads across machines. Such a simple and flexible procedure follows the iterative nature of IG, enhancing the exploration and exploitation on the solution space. As a result, IG is highly competitive with other metaheuristics like genetic algorithms and simulated annealing (Ruiz and Stützle, 2007), making it extremely popular for solving combinatorial problems and providing high-quality solutions within reasonable computational times.

IG methods have been successfully applied to various combinatorial optimization problems across the literature. The applications of IG mainly concentrate on scheduling, routing, and packing problems. Most notably, IG has been widely applied to various scheduling problems, to name a few, Flowshop Scheduling Problem (Ruiz and Stützle, 2008; Ribas *et al.*, 2011), Permutation Flowshop Scheduling Problem (PFSP) (Ruiz and Stützle, 2007; Pan *et al.*, 2008; Hatami *et al.*, 2015; Ruiz *et al.*, 2019), Hybrid Flowshop Scheduling Problem (Li *et al.*, 2015; Pan *et al.*, 2017; Missaoui and Ruiz, 2022), Parallel Machine Scheduling (Fanjul-Peyro and Ruiz, 2010; Rodriguez *et al.*, 2013), Project Scheduling (Gao *et al.*, 2024). These scheduling often involve sequencing tasks to optimize objectives like minimizing makespan, total flow time, or tardiness, which is typically $\mathcal{NP}$-hard, meaning finding optimal solutions is computationally expensive, especially for large instances. Moreover, IG method has also proven to be highly versatile and has been applied to a wide range of classical combinatorial problems beyond scheduling, such as Traveling Salesman Problems (TSPs) (Karabulut and Tasgetiren, 2014; Cuervo *et al.*, 2014), Vehicle Routing problems (VRPs) (Nucamendi-Guillén *et al.*, 2018) and knapsack problems (García-Martínez *et al.*, 2014; Avci and Topaloglu, 2017). These studies show that the simple iterative process of IG, combined with its flexibility and efficiency, makes it a powerful tool for addressing complex optimization problems across multiple domains.

This chapter proposes novel IG methods to tackle the YCSP generalizations in Section 3. We introduce a series of simple yet powerful IG methods that incrementally introduce improvements in their operators. The proposed IG methods incorporate several enhancements, including variations in destruction and reconstruction operators, integration with novel local search techniques, and problem-specific knowledge speed-ups. We calibrate the proposed IG methods using comprehensive computational experiments and prove that these methods outperform

the state-of-the-art approaches by a wide margin.

The rest of this chapter is organized as follows: Section 6.2 details the operators of the proposed IG methods. Especially, a series of new local search methods that are utilized in the IG algorithms are proposed in 6.2.3. All the proposed IG methods are statistically calibrated and integrated into three IG procedures, along with comprehensive computational experiments and comparisons in Section 6.4. Finally, Section 6.5 concludes the chapter and outlines future avenues of research.

6.2 Operators of the proposed IG

To the best of our knowledge, we are not aware of any application of IG to YCSPs. Algorithm 5 shows the outline of a classical IG (Ruiz and Stützle, 2007). IG procedures typically start from a high-quality solution. Two existing rule-based constructive methods (TPR and SETR) are tested. Among these two, the TPR (Time Parameter Rule), proposed by Vallada *et al.* (2023), is also the simplest, where containers are sorted in ascending order according to their time parameters, r_c, d_c, or e_c. SETR has been introduced in Section 4.3 where containers are sorted in increasing order of the Simulated Ending Times (η_c) according to expression 4.1. The η_c values are estimations of the crane finishing time for container c, (FC_c) with the YC passing the median I/O points (Section 4.3).

For both TPR and SETR solutions, we have adopted the I/O assignment heuristic presented in Section 4.3.2. This looks for the closest I/O points to the destination of the container c among all I/O options available from r_c, d_c, or e_c to SO_c. In the extreme case, if none of the I/O points are available, the one released first is assigned. Both constructive methods will be tested and calibrated in later sections where it will be shown that SETR produces significantly better results than TPR.

As aforementioned, IG consists of several main operators, including destruction, construction, local search, and acceptance criterion. For the investigated YCSP, these operators are introduced as follows:

- Destruction: The procedure destroys the current solution by extracting a number of jobs/containers from the solution, resulting in incomplete solutions.
- Reconstruction: The procedure reconstructs the incomplete solution by inserting the removed jobs/container back to the best position in the solution using greedy methods.
- Local search: The local search improves the current solution by iteratively exploring its neighbor solutions until the local optima is reached.
- Acceptance criterion: The procedure decides if the new solutions after the local search should be accepted for the next iteration and updates the so far best solution.

Algorithm 5 : Iterated Greedy basic outline (Ruiz and Stützle, 2007).

1: π_0 := GenerateInitialSolution
2: π := LocalSearch(π_0)
3: π_{best} :=π
4: **while** stopping criterion is not satisfied **do**
5: $\quad \pi_D$, π_R:= Destruction (π)
6: $\quad \pi'$:= Reconstruction (π_D, π_R)
7: $\quad \pi''$:= LocalSearch (π')
8: $\quad \pi$, π_{best}:= AcceptanceCriterion (π', π'', π_{best})

These main operators iterate until the stopping criterion is met. Note that the destruction and reconstruction are vital for the IG methods, where the resulting new solution may break the local optima obtained from the local search of the last iteration. Therefore, novel operators of destruction and reconstruction are developed in the later sections.

6.2.1 Destruction procedure

Destruction is a fundamental operator in IG procedures. Too weak of a destruction results in the subsequent local search circling back to the previous local optimum or to a similar one. On the contrary, an overly strong destruction degenerates into a random walk. We tested two destruction methods. The first is the aforementioned IG0 destruction first proposed by Ruiz and Stützle (2007) where a number d of containers are randomly extracted from the incumbent solution. The I/O points assigned for all containers are preserved, and all these removed containers will be reinserted during the reconstruction.

The second destruction operator is a novel approach that builds on the previous one adding destruction also to the I/O assignments. More precisely, after d containers are removed, d_2 randomly chosen I/O assignments of containers are destroyed by replacing their I/O points with random ones on their corresponding side. The process to select these d_2 I/O assignments is the following: We select a random I/O point and then remove randomly d_2 I/O assignments of containers. All these d_2 containers had used the chosen I/O point. Note that if there are fewer than d_2 containers with this I/O point assigned, then we remove fewer I/O assignments. So basically, we are randomly removing up to d_2 I/O assignments of containers from the same randomly chosen I/O point. The rationale behind this directed destruction is that in order to deal with congestion, it is better to target specific I/O points at each destruction application. Note that the second destruction operator destroys both container sequences as well as I/O assignments. We simply refer to this second destruction operator as '2$levels$'. We also simply refer to the first operator as '$Original$'. Destruction plays a salient role in diversifying the incumbent solution after finding the local optima, which may allow for explorations of better solutions in another solution space during the following reconstruction. Algorithm 6 shows these processes, and an example of '2$levels$' destruction will

be presented in Section 6.2.2.

Algorithm 6 : $2levels$ destruction and reconstruction

```
1: for i := 1 to d do                    % Destruction on loading sequence
2:     π_D := remove one container at random from π and insert it in π_R
3: π_D := remove one container at random from π_D and insert it in π_R
4: π_IO:= select an I/O randomly and collect d2 containers using it to π_IO
5: for i := 1 to d2 do                   % Destruction on I/O assignments
6:     π_D, π_R := replace the I/O of container π_IO(i) with another random I/O in the same side
7: for i := 1 to d do                    % Reconstruction on loading sequence
8:     π_D := best permutation obtained by inserting container π_R(i) in all possible position of π_D
9: for i := 1 to d2 do                   % Reconstruction on I/O assignments
10:    π_D := best I/O points obtained by try all I/O points for container π_R(i) in all possible option in the same side
```


6.2.2 Reconstruction procedure

In the reconstruction operator we generate a complete solution, usually applying some limited form of local search, so that the new reconstructed solution is not arbitrarily bad. We also examine two reconstruction methods. The first one is again an adaptation of the baseline (referred to as '*Original*') by Ruiz and Stützle (2007). Here, each previously removed container is tested in all positions of the container sequence and finally placed into the position resulting in the best value of the Objective Function (expression 3.2). Assigned I/O points are kept for all containers. Similar to the second destruction operator, we also name $2levels$ the second improved reconstruction operator. Here we also do reassignments of the I/O points after reconstructing all containers. Each reconstructed container has all potential I/O assignments on the corresponding side tested, and the one resulting in the best objective

function is kept. Note that this results in a maximum of $dn + d_2m$ calculations of the OF (expression 3.2), which is significantly more expensive than the *Original* operator. However, and as will be shown later in the experimental evaluation, this additional cost results in statistically significant improvements in the quality of the produced solutions.

A quick example illustrates the '2$levels$' destruction and reconstruction operator with five containers, from Con1 to Con5, and two I/O points, IO1 and IO2 on the same side. Figure 6.1 shows the solutions before and after '2$levels$' destruction with $d = 2$ and $d_2 = 1$ where modifications are in bold. Using $d = 2$, Con1 and Con3 are removed from the solution. Employing $d_2 = 1$, we randomly select I/O point IO1 and remove $d_2 = 1$ random assignments of IO1, in this case, the I/O assignment of Con5 is destroyed and randomly replaced by assignment IO2. During the reconstruction, a random destroyed container (Con1) is chosen and inserted into the best position with the minimal objective function, e.g., position 2 in Figure 6.2. Similarly, Con3 is tested in all positions and finally inserted into the last one. After having all containers reinserted into the sequence, all I/O points of the d_2 containers that had their I/O assignments removed are reconstructed. In this case, all possible I/O points are tested for Con5 and IO1 is selected.

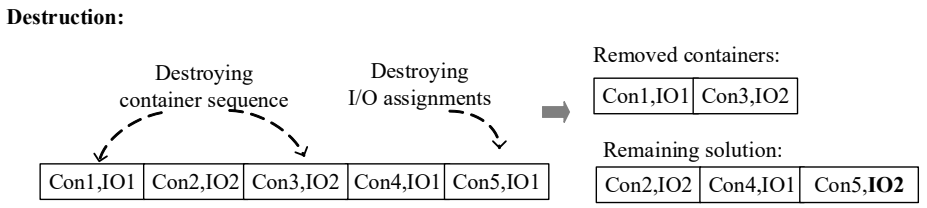


Figure 6.1: Example for 2$levels$ destruction operator with $d = 2$. Elaborated by the author.

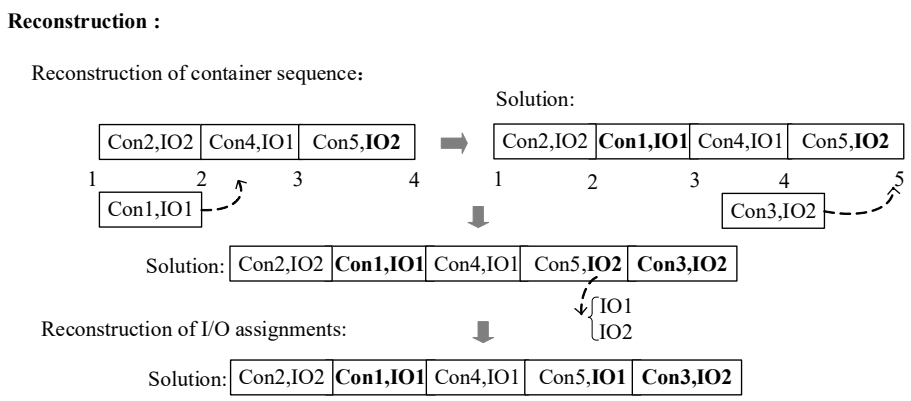


Figure 6.2: Example for $2levels$ reconstruction operator. Elaborated by the author.

6.2.3 Local search

Local search plays a crucial role in improving the solution quality in the iterated greedy method by iteratively exploring and exploiting neighborhood solution space.We examine three different local search procedures, referred to as LS_{GR}, LS1 and VND, respectively. The first one is exactly the local search method adopted for the GRASP approach (Wang *et al.*, 2025) introduced in Section 4.3.3. The local search extracts containers one by one and inserts each container back into the solution at a random position from a subset of the top β best positions. More details can be found in Section 4.3.3. LS_{GR} performed very well in the GRASP and, therefore, can be tested here as a baseline method. The second local search, LS1, is an adapted version of a classical neighborhood search method where more advanced techniques are integrated with variations on different search levels. Based on the LS1, we have proposed a Variable Neighborhood Descent (VND) as the third local search method, which explores different solution spaces on different neighbors of not only the container sequence but also the assignments of I/O points. All these local search methods will be calibrated in Section 6.4.2.

An advanced neighborhood search (LS1)

LS1, detailed in Algorithm 7, basically follows a classical local search of the swap neighborhood. It seeks the best improvement for every single container, where each container swaps positions with every other container, and the best pair is swapped. At each loop, LS1 reshuffles the container indexes so that the same container pairs are not examined in the same order every time. It is important to note that LS1 terminates as soon as the incumbent objective function value is improved. This results in a speedy local search. I/O assignments of containers are preserved along with the swap moves all of the time.

Algorithm 7 : LS1(incumbent solution S^*, incumbent objective f^*)

```
1:  improvement := true
2:  while improvement := true do
3:      improvement = false
4:      C_reshf is the reshuffled index of containers in S*
5:      for each container i in C_reshf do        % local search on container
        sequence
6:          Exchange positions for i with all containers of the same type
7:          Swap i in S* with the container that obtains the best f*
8:          if f* is strictly improved then
9:              improvement := true
10:             break
```

This local search is proposed by a neighborhood search frame and is proved to be the best proposal for the IG method. For the sake of brevity, we put the design process of LS1 and the corresponding experiments in Appendix C. We believe that the procedure of designing a local search itself is meaningful and also inspiring for even other heuristic domains.

Variable Neighborhood Descent (VND)

Variable Neighborhood Descent (VND) is a sophisticated methodology that improves local search by systematically changing neighborhood

structures to escape local optima. VND is rooted in the broader development of the Variable Neighborhood Search (VNS) framework, introduced by Hansen and Mladenović (1997). As a specialized form of VNS, it was developed to focus on descent-based searches across multiple neighborhoods. It builds on classical local search procedures, enhancing them by allowing dynamic changes in neighborhood structures without randomization. This systematic exploration of neighborhoods makes VND more effective in solving combinatorial optimization problems than traditional local search methods, which tend to get stuck in local optimum from a single neighborhood. VND has been applied as a single approach or a part of algorithms to a variety of optimization problems, such as scheduling (Zandieh and Adibi, 2010), routing (Fleszar *et al.*, 2009), and location problems (Mladenović *et al.*, 2003).

In this section, we have proposed a VND local search to find better solutions from two kinds of neighborhood search. The first neighborhood search seeks improvements on the crane loading sequence of containers using LS1, and the other improves the solution from the I/O neighborhood. Algorithm 8 presents the neighborhood search on I/O assignments where, for each container, it tests all the I/O options on the corresponding side and assigns the best I/O point as a result. With these two local searches, the complete VND procedure is detailed in Algorithm 9. In this way, the VND adaptively explores neighborhood spaces of both the sequence of containers and the assignments of I/O points until no improvement can be found by either LS1 or LS_I/O.

Algorithm 8 : LS_I/O(incumbent solution S^*, incumbent objective f^*)

```
1: for each container c in S* do
2:     Test all other I/O options on this side for c
3:     O is the I/O point that obtains the best f*
4:     if f* is strictly improved then
5:         Assign I/O point O to c
```

Algorithm 9 : VND (incumbent solution S^*, incumbent objective f^*)

```
1: improvement := true
2: while improvement := true do
3:     improvement = false
4:     S' ← LS1 (S*, f*)        % local search on containers
5:     if f* is strictly improved then
6:         improvement := true
7:     S'' ← LS_I/O (S', f*) % local search on I/O points
8:     if f* is strictly improved then
9:         improvement := true
```

Acceleration in local search

Problem Specific Knowledge (PSK) is a popular search acceleration technique applied to various optimization problems (Arnold and Sörensen, 2019; Riahi *et al.*, 2019; Zhang *et al.*, 2024). PSK in local search refers to the use of domain-specific insights, rules, or heuristics to guide and accelerate the search process. By incorporating PSK, the algorithm can focus on more promising areas of the solution space, avoid unnecessary computations, and quickly discard non-viable solutions. This tailored knowledge can take the form of problem constraints, optimization goals, or specialized neighborhood moves that are more likely to lead to improvements. By embedding PSK, local search algorithms become more efficient, improving both the speed of convergence and the quality of the solutions in complex optimization problems.

We have incorporated a PSK speed-up in the VND. The PSK was observed when we examined the results of LS1, where most of the improvement moves correspond to pairs of containers of the same type. This is surprising as only a quarter of the possible container pairs are made up of containers of the same type for any of the calibration instances (in Section 5.3.2). After careful experiments on these instances, we observed that by limiting container moves only to neighbors of the same type, we obtained very little (if any) degradation in solution quality

and a considerable speed-up in the local search. This is expected as mixing types may result in the YC moving from one side to the other, traversing all the block. Note that to reduce any potential degradation of the solution quality, we allow a very small number of movements between different containers. More specifically, we swap all containers of the same type (25% of all containers) and 5% of the remaining 75% pairs of containers of different types, resulting in 28.75% of container pairs being examined during the local search. The PSK will be calibrated as a level of the proposed local search in Section 6.4.2 where it turns out that the PSK yields statically better results.

6.2.4 Acceptance criterion

The last step during each loop of the IG is to decide if the newly obtained solution replaces the incumbent or not. The original acceptance criterion of Ruiz and Stützle (2007) depends on a temperature parameter. In the interest of having very simple IG procedures, we adopt the parameterless acceptance criterion of Hatami *et al.* (2015), where a solution is accepted if $rand \leq e^{-(RPD)}$. Where $rand$ is a uniformly distributed random number in the $[0, 1]$ range and RPD is the Relative Percentage Deviation (RPD) calculated by expression 4.2. Of course, the new solution is accepted as the incumbent and the new best if it outperforms the best solution.

procedure 10 : Acceptance criteria (AC, Hatami *et al.*, 2015)

```
if f(π'') < f(π) then
    π = π'';
    if f(π) < f(π_b) then
        π_b = π;
else if random ≤ e^{-RPD} then
    π = π'';
```

6.3 Proposed IG methods

This section proposes three simple yet efficient IG methods with new IG operators in Section 6.2 and 6.2.3 following the basic IG framework in Algorithm 15 in Section 6.2. We first present a vanilla IG template, a basic IG0 in Section 6.3.1 in a basic outline of Ruiz and Stützle (2007). In section 6.3.2, a new IG1 is presented using a more efficient local search method proposed in Section 6.2.3. Based on IG1, we designed new operators of destruction and reconstruction for a new approach, IG2 in Section 6.3.3 where the VND local search in Section 6.2.3 is used. Eventually, Section 6.3.4 develops the third IG method, IG3 upon the IG2 with the new proposed PSK in Section 6.2.3. Table 6.1 shows the basic configurations of all proposed IG methods, which are decided by well-designed calibration experiments on several main factors of IG algorithms in Section 6.4.2.

Table 6.1: Configurations for the proposed IG algorithms where $Original$ refers to the original proposal of Ruiz and Stützle (2007) and $2levels$ is the new proposed destruction operator in Section 6.2.1.

Algorithm	Destruction	Local Search	PSK
IG0	$Original$	LS_{GR}	No
IG1	$Original$	LS1	No
IG2	$2levels$	VND	No
IG3	$2levels$	VND	Yes

6.3.1 The proposed vanilla IG: IG0

Let us first introduce a vanilla IG, that will work as a baseline following the original work of Ruiz and Stützle (2007). Algorithm 11 shows the outline. In a nutshell, this basic IG, referred to as IG0 henceforth, uses the same initial solution and local search procedures of the recently proposed Greedy Randomized Adaptive Search Procedure (GRASP)

in Chapter 5. The solution is then destroyed and reconstructed to escape the previously found local optimum. More precisely, a number d of containers is randomly removed from the current permutation π of containers in the YC schedule in the destruction operator. These are then later reinserted back into the permutation into the best positions, one by one, during the reconstruction. Note that the local search proposed in Section 4.3.3 is adopted to insert each container randomly into one of the top β positions among all possible options. In this procedure, I/O points are assigned using the same heuristic employed when building the initial solution. After the local search, the new solution is considered in accordance with the acceptance criterion and the procedure iterates until the stopping criterion is met.

Using the same initial solution and local search, we believe IG0 constitutes a fair baseline for comparison against the proposed GRASP in Chapter 5. The destruction and reconstruction operators, as well as the acceptance criterion, are further detailed in the following sections.

6.3.2 The first proposed IG method: IG1

Furthermore, we introduce a new proposal of IG1 based on the basic version IG0 with the intensification of more powerful local search methods. The same initial solution of the IG0 is introduced, which is the same proposal of the GRASP in Chapter 5. Meanwhile, IG1 adopts the original destruction and reconstruction of Ruiz and Stützle (2007). Also note that the local search is a crucial component of the IG algorithm due to that it has a great affection on the solution quality and algorithm efficiency. IG1 adopts a new local search LS1 presented in Section 6.2.3. As will be shown in later sections, the newly proposed IG1 results in a better performance than IG0 by significantly improving the quality of solutions. We explain that the new local search is able to search for promising regions of the solution space, leading to faster convergence to high-quality solutions.

procedure 11 : Iterated Greedy: IG0

1: π_{best} := Generate an initial solution (Section 4.3)
2: π := LS_{GR} (π_b) % Local search (Section 4.3)
3: π_{best} :=π
4: **while** stopping criterion is not satisfied **do**
5: π_D=π % Destruction phase (Ruiz and Stützle, 2007)
6: **for** $i := 1$ to d **do**
7: π_D := remove one container at random from π_D and insert it in π_R
8: **for** $i := 1$ to d **do** % Construction phase (Ruiz and Stützle, 2007)
9: π_D := best permutation obtained by inserting container $\pi_R(i)$ in all possible position of π_D
10: π'' := LS_{GR} (π_D) % Local search (Section 4.3)
11: **if** $f(\pi'') < f(\pi)$ **then** % Acceptance Criterion (Section 6.2.4)
12: $\pi = \pi''$;
13: **if** $f(\pi) < f(\pi_{best})$ **then**
14: $\pi_{best} = \pi$;
15: **else if** $random \leq e^{-RPD}$ **then** % *Random* is a random number in $[0, 1]$
16: $\pi = \pi''$;
17: Return π_{best}

procedure 12 : Iterated Greedy: IG1

```
1: π_best := Generate an initial solution (Section 4.3)
2: π := LS1(π_b) % Local search (Section 6.2.3)
3: π_best :=π
4: while stopping criterion is not satisfied do
5:     π_D=π                % Destruction phase (Ruiz and Stützle, 2007)
6:     for i := 1 to d do
7:         π_D := remove one container at random from π_D and insert
           it in π_R
8:     for i := 1 to d do          % Construction phase (Ruiz and
       Stützle, 2007)
9:         π_D := best permutation obtained by inserting container
           π_R(i) in all possible position of π_D
10:    π'' := LS1 (π_D)            % Local search (Section 6.2.3)
11:    if f(π'') < f(π) then       % Acceptance Criterion (Section
       6.2.4)
12:        π = π'';
13:        if f(π) < f(π_best) then
14:            π_best = π;
15:    else if random ≤ e^{-RPD} then   % Random is a random
       number in [0, 1]
16:        π = π'';
17: Return π_best
```

6.3.3 The second proposed IG method: IG2

A new IG method, referred to as IG2, is designed with innovation on main IG operators of destruction, reconstruction, and local search. Basically, IG2 adopts intensification of $2levels$ in Section 6.2.1 and VND in Section 6.2.3 where the solution will be destructed and reconstructed in 2 levels, including the loading sequence of containers and their I/O assignments. Note that additional perturbations on I/O assignments of

containers are allowed in IG2 compared with IG1, where the operators *Original* and LS1 work only on the loading sequence of containers.

procedure 13 : Iterated Greedy: IG2

1: π_{best} := Generate an initial solution (Section 4.3)
2: π := VND (π_b) % Local search (Section 6.2.3)
3: π_{best} :=π
4: **while** $CPUtime \leq n \times \ln n \times m \times \tau$ milliseconds **do**
5: $\quad \pi'$:=π % $2levels$ (Section 6.2.1)
6: $\quad$ **while** *improvement* := **true do** % VND local search (Section 6.2.3)
7: $\quad\quad$ *improvement* = **false**
8: $\quad\quad$ C_{reshf} is the reshuffled index of containers in π'
9: $\quad\quad$ **for each** container i in C_{reshf} **do** % LS on container sequence
10: $\quad\quad\quad$ Exchange positions for i with all the rest of containers
11: $\quad\quad\quad$ π'' := Swap i in S^* with the container that obtains the best f^*
12: $\quad\quad\quad$ **if** f^* is strictly improved **then**
13: $\quad\quad\quad\quad$ *improvement* := **true**
14: $\quad\quad\quad\quad$ **break**
15: $\quad\quad$ π'' := LS_I/O(π'') % LS on I/O assignments
16: $\quad$ **if** $f(\pi'') < f(\pi)$ **then** % Acceptance Criterion (Section 6.2.4)
17: $\quad\quad$ $\pi = \pi''$;
18: $\quad\quad$ **if** $f(\pi) < f(\pi_{best})$ **then**
19: $\quad\quad\quad$ $\pi_{best} = \pi$;
20: $\quad$ **else if** $random \leq e^{-RPD}$ **then** % *Random* is a random number in $[0, 1]$
21: $\quad\quad$ $\pi = \pi''$;
22: Return π_{best}

6.3.4 The third proposed IG method: IG3

Furthermore, we introduced the PSK speed-up in Section 6.2.3 to IG2, resulting in a new IG method IG3. Algorithm 14 shows the outline that allows the container to swap with the other containers of the same type in line 10. However, each container has to be swapped with all the rest of the containers in IG2. It will turn out that in later sections, this modification results in statistically better performance than IG2, especially for the large instances.

Note that we have adopted the same stopping criterion as the GRASP methods in Chapter 5 for all proposed methods. All IG methods will run within a CPU time within $n \times \ln n \times m \times \tau$ milliseconds where n and m is the number of containers and I/O points respectively, and τ is CPU time level ranging from $\{10, 20, 30\}$. Besides, all the proposed adopt the same acceptance criterion introduced in Section 6.2.4. A careful calibration for all these IG methods will be presented in Section 6.4.2.

procedure 14 : Iterated Greedy: IG3

1: π_{best} := Generate an initial solution (Section 4.3)
2: π := VND (π_b) % Local search (Section 6.2.3)
3: π_{best} :=π
4: **while** $CPUtime \leq n \times \ln n \times m \times \tau$ milliseconds **do**
5: π' :=π % $2levels$ (Section 6.2.1)
6: **while** *improvement* := **true do** % VND local search (Section 6.2.3)
7: *improvement* = **false**
8: C_{reshf} is the reshuffled index of containers in π'
9: **for each** container i in C_{reshf} **do** % LS on container sequence
10: Exchange positions for i with all containers of the same type % using PSK (Section 6.2.3)
11: π'' := Swap i in S^* with the container that obtains the best f^*
12: **if** f^* is strictly improved **then**
13: *improvement* := **true**
14: **break**
15: π'' := LS_I/O(π'') % LS on I/O assignments
16: **if** $f(\pi'') < f(\pi)$ **then** % Acceptance Criterion (Section 6.2.4)
17: $\pi = \pi''$;
18: **if** $f(\pi) < f(\pi_{best})$ **then**
19: $\pi_{best} = \pi$;
20: **else if** $random \leq e^{-RPD}$ **then** % *Random* is a random number in $[0, 1]$
21: $\pi = \pi''$;
22: Return π_{best}

6.4 Computational and statistical experimentation

6.4.1 Experimental instances and settings

We carry out an extensive experimental campaign on the same benchmark instances introduced in Section 4.4.2. To better calibrate the proposed IG method, we have adopted the same calibration instances used for the GRASP methods in Chapter 5. Additionally, all IG and GRASP methods are computed in the same computational environment introduced in Section 4.4.1. Using the same instances and the same computational environments, we can make a fair comparison between the proposed GRASP and IG methods.

6.4.2 Statistical calibration

Statistical calibration experiments are carried out using the Design of Experiments (DOE) approach together with the Analysis of Variance (ANOVA) statistical technique (Montgomery, 2017). We first investigate how each operator performs in the IG method, where all operators of the IG method are tested in Section 6.4.2. Using the calibration results, we have proposed three IG methods, and each of them is then calibrated in Section 6.4.2.

Calibration of all IG operators

We carry out a complete factorial DOE with 8 factors (all combinations of the factor's levels and variants are studied). The results are analyzed in a multi-factor ANOVA. The 8 studied factors potentially affect the performance of the proposed IG methods. Instance factors n and m are uncontrollable as they are given by the instance sizes but are included in the experiment so as to be able to study how the different IG factors interact with instance sizes. Each instance is run several times with

different random seeds to get replicates in the experiment. The different replicate numbers are used as a witness factor, which is expected to be insignificant, i.e., the replicate number should not be statistically significant. The five remaining factors are controllable and are:

1. Heuristic method used for obtaining the initial solution (IniMethod) at two levels: TPR and SETR;
2. Number of destroyed containers (d) in the destruction operator, tested at 5 levels: {3, 6, 9, 12, 15};
3. Destruction size of I/O points (d_2), tested at 4 levels: {0, 2, 4, 6}. Note that level 0 is a witness level as it implies not destroying I/O points at all;
4. Local search methods (LSType) at three variants: LS_{GR}, LS1 and VND;
5. Local search speed-up at two variants: 1 (enabled) and 0 (disabled).

The RPD is the response variable in the ANOVA. In total, there are $2 \times 5 \times 4 \times 3 \times 2 = 240$ combinations for the controlled factors. We employ the same stopping CPU time of the GRASP in Chapter 5, where $\tau = 10$ for the 144 calibration instances. As a result, we have run $240 \times 144 \times 10 = 345,600$ experiments with a total CPU time usage of 3,652 hours, or about 152 days.

The detailed ANOVA tables from the experiment are omitted in this section for the sake of brevity. Instead, we provide them in Appendix B with complete supporting data for the calibrations. Among the five controllable factors, LSType is the most significant, resulting in a very large F-ratio. This is illustrated in Figure 6.3 by wide level gaps and very narrow (almost non-existent) confidence intervals around the means. It is shown that VND clearly outperforms the other two local search methods. Overly large F-ratios tend to affect the normality and

homoscedasticity hypotheses (Montgomery, 2017). Basically, statistical testing is not needed when the results are very evident. Therefore, LSType is fixed at the corresponding level for each algorithm in the calibrations of the different IG procedures. This results in three IG variants: IG1, IG2, and IG3, with key modifications from the baseline IG0 where their basic setting can be found in Table 6.1.

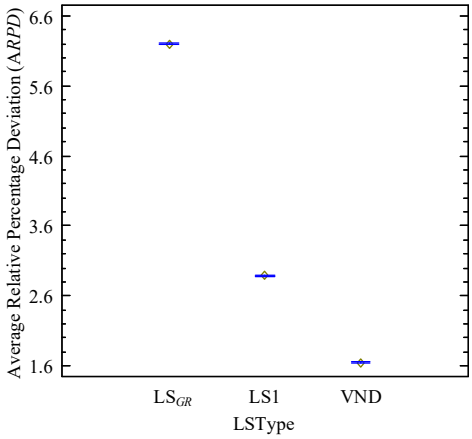


Figure 6.3: Means plots for the tested local search procedures factor (LSType) in the ANOVA calibration with 95% Tukey's Honest Significant Difference (HSD) confidence intervals. Elaborated by the author.

Calibration of the proposed IG methods

The IG methods are calibrated with the remaining controllable factors. For the sake of brevity, we put all the ANOVA tables as well as means plots of factors as an appendix in Appendix B. The following will conclude with some analysis results of the calibration experiments on the proposed IG methods. IG0 and IG1 have only two factors, IniMethod and d, that need calibration. IniMethod is more significant than d for IG0 and IG1 with SETR being the better constructive heuristic. As a matter of fact, SETR outperforms TPR for all the proposed IG procedures. Their means plots for IG0 and IG1 are given in Figure 6.4 and 6.5 respectively, where 9 is the best destruction size.

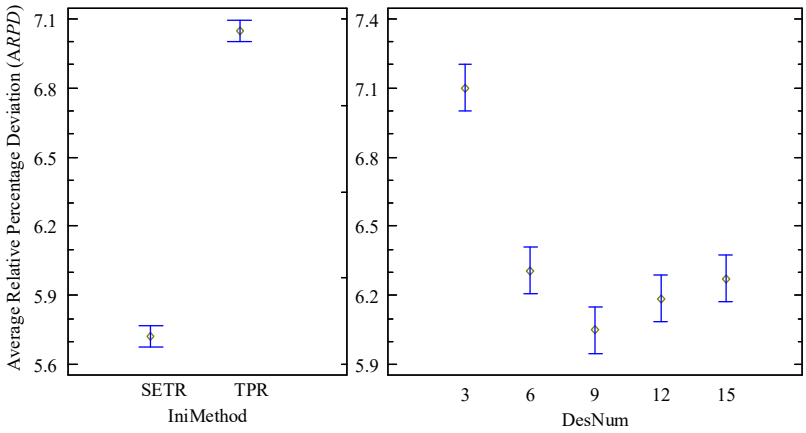


Figure 6.4: Means plots for all the factors in the ANOVA calibration for IG0 with 95% Tukey's Honest Significant Difference (HSD) confidence intervals. Elaborated by the author.

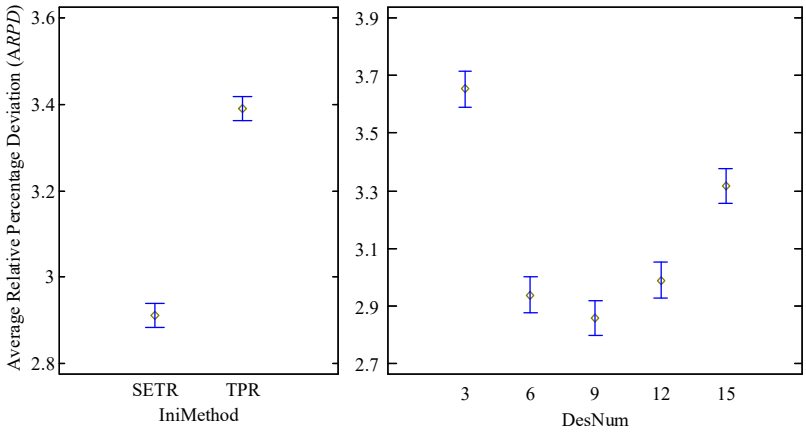


Figure 6.5: Means plots for all the factors in the ANOVA calibration for IG1 with 95% Tukey's Honest Significant Difference (HSD) confidence intervals. Elaborated by the author.

Figure 6.6 presents the means plots for IG2 with subfigures to the right in decreasing order of statistical significance. From the plot of destruction size, the number of containers to be removed is set at 12. As can be observed, d_2 at levels 2 to 6 is better than at level 0, meaning

that the proposed 2*levels* destruction operator with I/O destruction is better than *Original*. We set DesIO at 2 as it is slightly better than 4, although 2 and 4 are statistically equivalent at 95% HSD confidence across all instances (although one could zoom in for specific instance size intersections, we prefer not to over-calibrate). Performing a similar analysis for IG3 results in similar means plots given in Figure 6.7. Here d_2=2 is statically better than values 4 and 6 and much better than 0, showing again the improvements from new operators of 2*levels* destruction and reconstruction. Of special concern is the factor PSK. The means plot in Figure 6.6 indicates that the local search speed-up provides some small, but statistically significant performance gains.

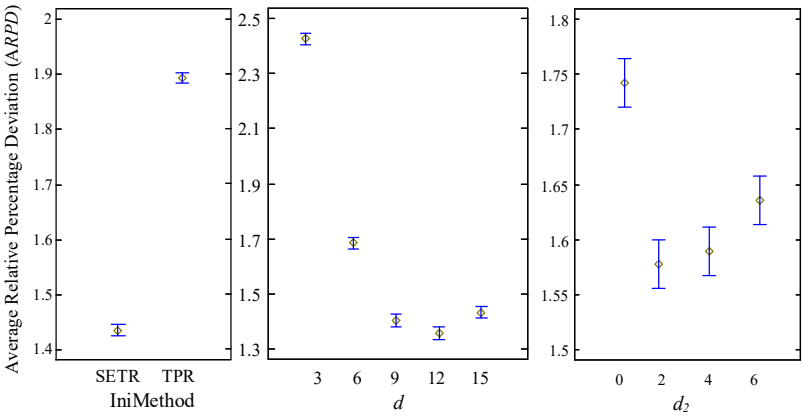


Figure 6.6: Means plots for all the factors in the ANOVA calibration for IG2 with 95% Tukey's Honest Significant Difference (HSD) confidence intervals. Elaborated by the author.

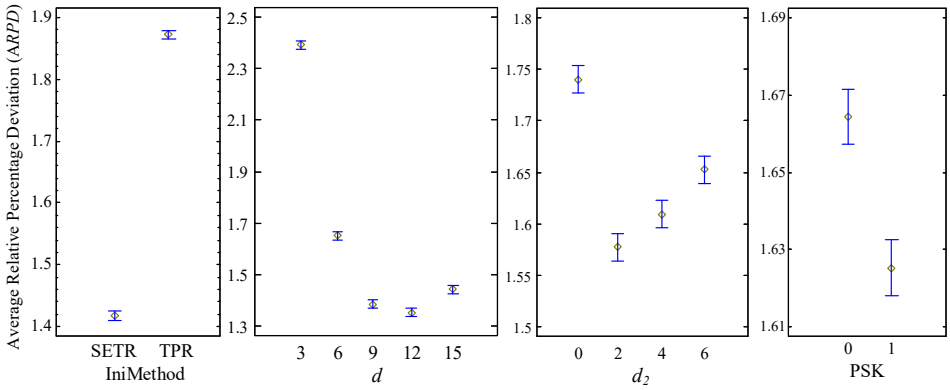


Figure 6.7: Means plots for all the factors in the ANOVA calibration for IG3 with 95% Tukey's Honest Significant Difference (HSD) confidence intervals. Elaborated by the author.

6

6.4.3 Comparison of methods

We now compare the proposed IG algorithms against the 3 Local Search Method (LSM) of Vallada *et al.* (2023) and the GRASP (GR) of Chapter 5. Each algorithm is independently run five times for all 720 test instances of Section 4.4.2. Tables 6.2 and 6.3 show average results of small and larger instances, respectively. The results are grouped by container size n where each cell is the average of 60 instances and 5 runs (300 results). Note that the results of LSM consist of the best solutions among 3 algorithms of Vallada *et al.* (2023), TPR_{ls}, $MRTP_{ls}$ and NCR_{ls}. Small instances are run with $\tau = 10$ only as more time is not needed. Medium and large instances are run three independent times with $\tau = 10$, 20 and 30. It has to be stressed that LSM uses n seconds as a stopping criterion, which is always higher than the stopping criterion employed for the other algorithms. This is still true even for $\tau = 30$.

Regarding the small instances results in Table 6.2, we see that IG0 has found 211 optimums out of 240 instances, far more than LSM and GR. Note that LSM is at the same time slower. The number of

Table 6.2: Number of optimums (#Opt.) and Average Relative Percentage Deviation (*ARPD*) for small and small tight (ST) instances. CPU times (in seconds) for LSM are given in parenthesis whereas for all other methods, which share CPU times, they are given in the last column ($\tau = 10$). Best results in bold.

	n	LSM (s)	GR	IG0	IG1	IG2	IG3	Time (s)
#Opt.	5(ST)	43 (15)	53	**58**	**58**	**58**	**58**	**0.24**
	10(ST)	19 (30)	35	**57**	**57**	**57**	**57**	**0.69**
	5	52 (15)	49	**58**	**58**	**58**	**58**	**1.29**
	10	20 (30)	25	38	**56**	**56**	**56**	**3.68**
Total		134 (90)	162	211	**229**	**229**	**229**	**5.90**
ARPD	5(ST)	0.96 (15)	0.46	0.07	**0.06**	**0.06**	**0.06**	**0.24**
	10(ST)	3.24 (30)	0.95	0.05	**0.04**	**0.04**	**0.04**	**0.69**
	5	0.50 (15)	0.32	**0.04**	**0.04**	**0.04**	**0.04**	**1.29**
	10	1.15 (30)	0.83	0.19	**0.02**	**0.02**	**0.02**	**3.68**
Average		1.46 (22.5)	0.64	0.09	**0.04**	**0.04**	**0.04**	**1.48**

optimums increases with IG1, IG2 and IG3 to 229 (70.9% higher) with a 97.3% smaller *ARPD* of 0.04%. It is an interesting observation that the proposed IG methods can solve to optimality most small instances within a few seconds, while CPLEX 20.1 may need more than one hour (Vallada *et al.*, 2023).

As for the medium and large instances results in Table 6.3, IG0 has produced 685 new best solutions for the 720 instances from the previous results of the GRASP (GR) in Chapter 5. IG0 improves the *ARPD* by 64.6% and by 35.7% from LSM and GR for the medium and large instances and across all τ values respectively. We stress that IG0 is a very simple IG method using the same initial solution and local search as GR.

Among all proposed IG methods, IG3 performs the best. Let us start with IG1. IG1 outperforms IG0 for all instances. The overall *ARPD* of IG1 is 67.8% lower than IG0, widening the gap with LSM

Table 6.3: Average Relative Percentage Deviation (*ARPD*) and time of medium and large instances for the compared methods grouped by CPU time (τ) and number of containers (n). CPU times (in seconds) for LSM are given in parenthesis whereas for all other methods, which share CPU times, they are given in the last column. Best results in bold.

τ	n	LSM (s)	GR	IG0	IG1	IG2	IG3	Time (s)
10	15	4.01 (45)	1.20	0.59	0.02	**0.00**	**0.00**	**6.50**
	20	6.82 (60)	2.74	1.67	0.15	0.01	**0.00**	**9.59**
	30	11.61 (90)	5.87	3.83	0.81	0.12	**0.11**	**16.33**
	40	15.82 (120)	8.84	5.60	1.63	**0.24**	0.29	**23.61**
	50	16.90 (150)	11.08	6.52	2.35	**0.41**	0.53	**31.30**
	100	21.51 (300)	14.84	9.32	3.83	0.95	**0.86**	**73.68**
	150	25.45 (450)	15.65	11.03	4.18	1.14	**0.88**	**120.26**
	200	32.85 (600)	16.07	12.09	4.03	1.36	**1.14**	**169.55**
Average		16.87 (226)	9.54	6.33	2.12	0.53	**0.48**	**56.35**
20	15	4.01 (45)	1.20	0.48	0.01	**0.00**	**0.00**	**13.00**
	20	6.82 (60)	2.66	1.59	0.10	0.01	**0.00**	**19.17**
	30	11.61 (90)	5.59	3.73	0.60	0.11	**0.08**	**32.65**
	40	15.82 (120)	8.55	5.27	1.44	**0.14**	0.23	**47.22**
	50	16.90 (150)	10.52	6.21	2.11	**0.26**	0.34	**62.59**
	100	21.51 (300)	14.35	8.37	3.45	0.67	**0.41**	**147.37**
	150	25.45 (450)	15.28	10.29	3.63	0.64	**0.48**	**240.51**
	200	32.85 (600)	15.71	11.24	3.51	0.68	**0.53**	**339.09**
Average		16.87 (226)	9.23	5.90	1.86	0.31	**0.26**	**112.70**
30	15	4.01 (45)	1.20	0.44	0.01	**0.00**	**0.00**	**19.50**
	20	6.82 (60)	2.54	1.46	0.14	0.01	**0.00**	**28.76**
	30	11.61 (90)	5.52	3.65	0.47	0.10	**0.06**	**48.98**
	40	15.82 (120)	8.25	5.15	1.16	**0.11**	0.20	**70.83**
	50	16.90 (150)	10.31	6.04	2.02	**0.21**	0.28	**93.89**
	100	21.51 (300)	14.12	8.03	3.08	0.42	**0.30**	**221.05**
	150	25.45 (450)	15.07	9.94	3.71	0.41	**0.31**	**360.77**
	200	32.85 (600)	15.50	10.85	3.65	0.38	**0.31**	**508.64**
Average		16.87 (226)	9.06	5.69	1.78	0.20	**0.18**	**169.05**
Tot. Ave.		16.87 (226)	9.28	5.97	1.92	0.35	**0.31**	**112.70**

and GR. IG2 and IG3 go much further, with very small $ARPD$ values. We attribute this to the perturbation on I/O assignments from new IG operators like 2$levels$ destruction and the VND local search. On average, IG2 produces 81.7% smaller $ARPD$ than IG1. Clearly, IG3 is the best proposed alternative. As a matter of fact, it completely outperformed the GRASP, producing better results in all 720 instances. In general, IG3 results in 98.2% and 96.8% lower $ARPD$ values when compared to LSM and GR respectively.

While we expect almost all of these differences to be statistically significant, we carry out statistical testing to formally assess this outperformance. This time, a non-parametric statistical test, Wilcoxon signed-rank (García *et al.*, 2009), is employed. We compare the $ARPD$ results of all 720 instances. The number of instances that IG3 performs better, worse or equal to other methods is labelled with '-', '+' and '=', respectively, in Table 6.4, where R- represents the sum of negative/better ranks and R+, positive/worse ranks. Recall that the values of R+ and R-, to a certain extent, reflect the degree of difference between IG3 and the compared method. With a 95% confidence level, IG3 offers statistically superior results to LSM, GR, IG0, IG1 and IG2 for all three CPU time stopping criteria.

We now study the behaviour and effect of CPU time stopping criteria and instance sizes on the proposed IG algorithms. A new full factorial design studying the proposed methods, n and τ as factors along with RPD as a response variable is carried out. We focus on the high performing methods and disregard LSM. Figure 6.8 plots the interactions with 95% Turkey HSD intervals between the compared methods and container sizes, as well as CPU times. As can be observed, with larger instances, the gap between IG2/IG3 and GR and IG0 widens. This supports the hypothesis that the VND local search is very effective even for large problems. Figure 6.9 shows the interactions with 95% Turkey HSD intervals between the compared methods and CPU times.

Table 6.4: Significance of the Wilcoxon's signed-rank test between IG3 and the other algorithms at $\alpha = 0.05$ significance level for Average Relative Percentage Deviation (*ARPD*) across 720 instances for each τ stopping time.

τ	Method	-/+/=	R+	R-	p-value
10	LSM	580/1/139	1	169070	0.000
	GR	546/0/174	0	149331	0.000
	IG0	487/1/232	3	119313	0.000
	IG1	381/6/333	640	74438	0.000
	IG2	199/130/391	22524	31761	0.007
20	LSM	580/1/139	1	169070	0.000
	GR	546/0/174	0	149331	0.000
	IG0	478/2/240	52	115388	0.000
	IG1	368/6/346	691	69434	0.000
	IG2	188/134/398	22674	29329	0.047
30	LSM	580/1/139	1	169070	0.000
	GR	546/0/174	0	149331	0.000
	IG0	473/2/245	48	113002	0.000
	IG1	354/5/361	287	64333	0.000
	IG2	172/140/408	21905	28498	0.044

All proposed IG methods statically improve among different CPU time levels while results between $\tau = 20$ and 30 do not differ much, indicating convergence.

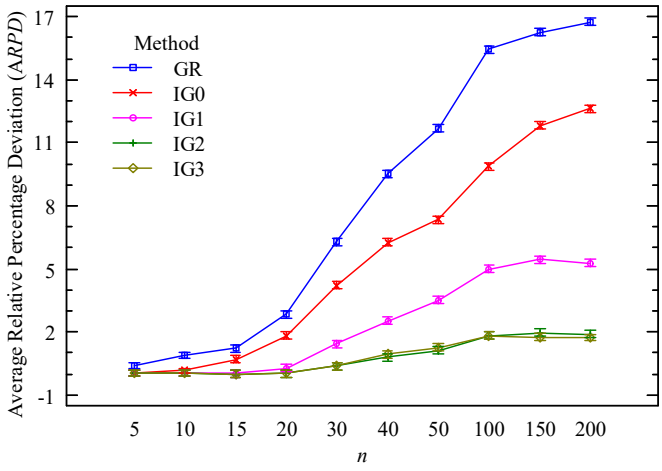


Figure 6.8: Means plot of the interaction between container size and compared methods with 95% Tukey's Honest Significant Difference (HSD) confidence intervals. Elaborated by the author.

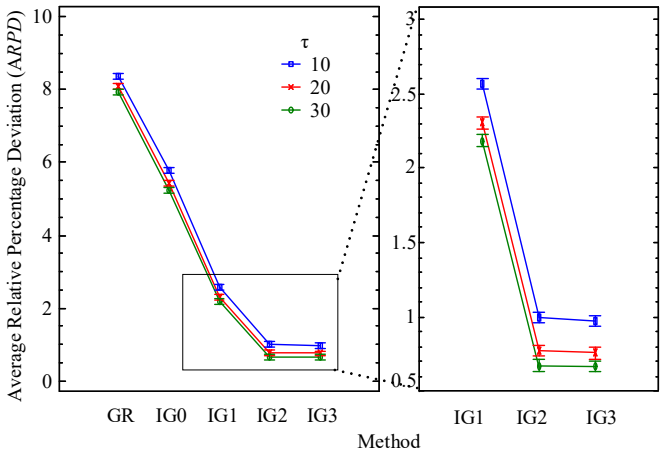


Figure 6.9: Means plot of the interaction between compared methods and CPU times with 95% Tukey's Honest Significant Difference (HSD) confidence intervals. Elaborated by the author.

6.5 Conclusions

In this chapter, we proposed four Iterated Greedy (IG) methods, each incorporating progressively enhanced improvements in their operators. These methods feature simple yet highly effective modifications to the destruction and reconstruction processes, applied to both container sequences and I/O assignments. Additionally, we introduced two powerful local search strategies, including a Variable Neighborhood Descent (VND). All IG methods are designed to be simple, easy to implement, calibrate, and extend.

Through detailed calibration and comprehensive comparisons with existing literature, we demonstrated that the simplest variant, IG0, significantly and statistically outperforms the best-performing method (LSM) from Vallada *et al.* (2023) and the GRASP approach described in Chapter 5. Specifically, IG0 achieves improvements of 64.6% and 35.7% in the Average Relative Percentage Deviation (ARPD) metric across all instances, respectively. The IG variants with additional improved operators maintain their simplicity while delivering even better performance than IG0. These improvements come from simple diversification and intensification mechanisms. Among all proposed methods, IG3 outperform all existing proposals, surpassing LSM by 98.2% and GRASP by 96.8% on the ARPD metric.

CHAPTER 7

Conclusions and future work

This chapter concludes the thesis, summarizes the academic activities and publications during the PhD, and outlines potential directions for future research.

7.1 Summary of the thesis

This thesis aims to enhance the delivery efficiency of yard cranes in real-world container yards, addressing the critical bottleneck in terminal ports. The study has investigated a highly realistic and complex Yard Crane Scheduling Problem (YCSP) that involves Input/Output assignments and is $\mathcal{NP}$-hard. Despite its practical importance, this problem is notoriously challenging to optimize. We have developed a series of heuristic and metaheuristic methods to tackle the YCSP efficiently. First, we proposed a Simulated End-time Rule (SETR) and its corresponding local search method (SETR_{ls}). Subsequently, two types of Greedy Randomized Adaptive Search Procedures (GRASP) and

three simple yet powerful Iterated Greedy (IG) methods were proposed, further improving the efficiency of the YCSP. More specifically:

An efficient SETR heuristic was first developed, featuring a constructive approach for sequencing containers and assigning Input/Output (I/O) points. This rule-based heuristic is simple and easy to implement. Despite its simplicity, experimental results demonstrated that the SETR heuristic consistently outperforms all existing heuristic rules reported in the literature.

Building on the SETR heuristic, we proposed two advanced GRASP approaches designed to enhance performance, particularly for larger instances. These methods incorporate newly developed constructive methods expanded from SETR and employ a novel local search to refine solutions. The approaches were carefully calibrated using the Design of Experiments (DOE) methodology (Montgomery, 2017), resulting in strong GRASP methods that surpass all rule-based heuristics and local search methods.

Three simple yet effective IG methods were proposed to achieve greater efficiency. Starting with a basic IG framework, we introduced novel operators tailored for the YCSP. These include advanced destruction and reconstruction techniques, alongside several newly proposed local search methods. Each IG method was meticulously designed and carefully calibrated that, while seemingly minor, leads to significant performance improvements, as validated by extensive experimental analysis.

In general, we commit to developing efficient heuristics and metaheuristics with new operators to optimise the realistic scheduling process of container delivery with practical thinking of I/O assignments. All proposed methods are carefully calibrated and extensively evaluated with comprehensive experiments among small to large instances. It turns out that the proposed methods have significantly outperformed the state-of-the-art proposals, which benefits not only more efficient

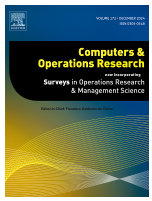
terminal ports but also a green and sustainable world.

7.2 Academic activities and publications

The author has participated in several academic visits, conferences, and seminars throughout the doctoral training program. A particularly significant academic visit was undertaken at the University of Sevilla, under the supervision of Prof. Dr. Federico Perea Rojas-Marcos, in collaboration with the Instituto de Matemáticas de la Universidad de Sevilla (IMUS). This three-month residency, spanning from September to December 2023, provided invaluable academic experience, offering fresh perspectives and deep insights that have made a substantial contribution to the development of this dissertation.

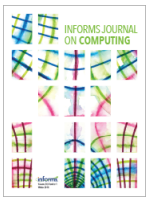
During the academic period from 2020 to 2024, several publications were made, focusing on solution methodologies for the Yard Crane Scheduling Problem (YCSP). The article titled "**Solving the Yard Crane Scheduling Problem with Dynamic Assignment of Input/Output Points**" presents effective heuristic approaches and GRASP algorithms, which serve as foundational contributions to Chapters 4 and 5 of this thesis. Furthermore, the working paper "**Iterated Greedy for the Yard Crane Scheduling Problem with Input/Output Assignment**" introduces a series of Iterated Greedy methods, which are integral to the content of Chapter 6.

Wang, Hongtao; Ruiz, Rubén; Villa, Fulgencia; Vallada, Eva (2025). Iterated Greedy for the Yard Crane Scheduling Problem with Input/Output Assignment. European Journal of Operational Research. 3rd revision.



Wang, Hongtao; Villa, Fulgencia; Vallada, Eva; Ruiz, Rubén (2025). Solving the yard crane scheduling problem with dynamic assignment of input/output points. Computers & Operations Research, 173, 106853. DOI:10.1016/j.cor.2024.106853.

Another paper was sufficiently improved during my PhD, while the topic is out of this thesis. It is an expansive work of the master thesis of the author focusing on a classical maximal covering location problem (MCLP) with new mathematical formulations and novel solving methodology.



Wang, Hongtao; Zhou, Jian (2025). Cross-entropy Method for the Maximal Covering Location Problem. INFORMS Journal on Computing. DOI:10.1287/ijoc.2024.0611.

Participation in several conferences and seminars during my Ph.D. has been a remarkable experience. I am grateful for the feedback and discussions with experts and peers during the following events:

- Wang, Hongtao; Ruiz, Rubén; Villa, Fulgencia; Vallada, Eva. 2024. Iterated Greedy for the yard crane schduling problem. 33rd European Conference on Operational Research (EURO 2024), June 30th to July 3rd of 2024, Technical University of Denmark (DTU), Copenhagen, Denmark.

- Wang, Hongtao; Villa, Fulgencia; Vallada, Eva; Ruiz, Rubén. 2024. Metaheuristic methods for the yard crane scheduling problem. DSA International Summer Conference 2024 (DSA 2024), June 6th to 7th of 2023, Ciudad Politécnica de la Innovación (CPI) and Universitat Politècnica de València (UPV), Valencia, Spain.

- Wang, Hongtao; Villa, Fulgencia; Vallada, Eva; Ruiz, Rubén. 2023. Heuristic and metaheuristic for the yard crane scheduling problem. PhD Seminar session of Instituto Matemáticas Universidad de Sevilla (IMUS 2023), November 16th of 2023, Sevilla University, Sevilla, Spain.

- Vallada, Eva; Wang, Hongtao; Villa, Fulgencia; Ruiz, Rubén. 2023. Heurísticas para la resolución de un problema de secuenciación de movimientos de la grúa de un patio de contenedores en una terminal portuaria. XL Congreso Nacional de Estadística e Investigación Operativa (SEIO 2023). November 7th to 10th of 2023, Elche, Alicante, Spain.

These conferences have broadened my horizons to the academic community, which have helped refine my ideas and provided inspiration for my research.

My sincere gratitude extends to all my supervisors, teachers, colleagues, peers, and friends whom I have had the privilege of meeting during conferences and seminars throughout my academic journey. Your insightful conversations, collaboration, and shared passion for learning have been a source of inspiration and motivation. The exchange of ideas, constructive feedback, and networking opportunities provided by these interactions have greatly enriched my academic experience. I am deeply thankful for the support, encouragement, and camaraderie that each of you brought into my journey, making these events not only educational but also memorable and innovative.

7.3 Future lines of research

1. **Integrating Yard Crane Scheduling with Other Terminal Operations**
   From a global perspective, combining and integrating the scheduling procedures of yard cranes with other operational processes in the terminal can be a highly promising area of research for bridging the gap between theoretical research and real-world operations. This includes coordinating yard crane movements with other necessary operations in the yard terminal, such as container positioning/relocation, vessel berth assignments, and quay crane loading and unloading. More detailed operations can be found in Section 1.2.3 in the thesis, where we have partially introduced some of this research in Section 1.2.3. In conclusion, the integration of yard crane scheduling with other terminal operations represents a valuable and impactful research direction. By integrating/synchronizing the yard crane scheduling procedure with practical applications, it is promising to improve operational efficiency, enhance resource utilization, and increase the overall throughput of container terminals.

2. **Exploitative application to multiple yard cranes**
   We have investigated the realistic YCSP for a single yard crane. However, expanding this research to include the scheduling problems involving multiple yard cranes presents a valuable opportunity. Specifically, this includes scenarios with both passable dual cranes and impassable twin cranes, which are the most common multi-crane configurations (see Section 2.3.2). Addressing the complexities of scheduling with multiple cranes involves several realistic considerations. For example, the assignment of input/output (I/O) points becomes more intricate, requiring advanced coordination to avoid conflicts and ensure efficient op-

erations. Additionally, the interaction between cranes, especially in unpassable twin cranes, necessitates sophisticated algorithms to manage potential interferences and optimize crane movements. Generally, delving into YCSPs within multiple yard cranes holds great potential for improving container yard performance. This makes it a highly relevant and impactful area for future research, promising to enhance the overall effectiveness of container handling processes.

3. **Recycling locations during the yard crane scheduling**
   During the yard crane scheduling process, the assignment of block locations is crucial. A promising area of research in this context is the optimization of these block locations. Specifically, after a retrieval container has been loaded by the yard crane, the now-vacant block location can be immediately repurposed for storing a new storage container. This setting can significantly enhance the efficiency of yard crane operations by decreasing the delivering time required for container movements. Instead of leaving block locations unused after retrieval, they can be quickly reassigned, reducing the need for additional movements and storage space. Implementing such a dynamic scheduling system requires sophisticated algorithms capable of real-time decision-making to ensure that the block locations are efficiently recycled.

APPENDIX A

Instance example

An non-equal instance file with 10 containers
number of cranes
1
number of containers
10
number of rows
10
number of columns
42
number of tiers
4
number of I/O land
6
number of I/O sea
10
gantry speed

1
trolley speed
1
hoisting speed
1
reshuffling time
2
Crane Initial Position % coordinates (x,y,z)
4 11 5
I/O Land information
#x y z
2 43 2
3 43 2
4 43 2
5 43 2
6 43 2
7 43 2
I/O Sea information
#x, y, z
1 0 1
2 0 1
3 0 1
4 0 1
5 0 1
6 0 1
7 0 1
8 0 1
9 0 1
10 0 1
Containers request information
#Containers requests information, one line per container

#Type	TypeLoc	x	y	z	RS	rj	dj	ej	wj	wcongj
3	O	5	18	3	0	-1	62	-1	3	3
3	O	8	25	2	0	-1	137	-1	3	3
1	D	3	3	3	0	194	-1	-1	3	3
2	D	8	2	3	0	65	-1	-1	1	1
4	O	6	4	4	0	-1	-1	26	1	1
2	D	9	33	2	0	154	-1	-1	1	1
2	D	9	30	2	0	182	-1	-1	1	1
2	D	8	26	2	0	178	-1	-1	1	1
2	D	2	3	1	0	184	-1	-1	1	1
3	O	1	13	2	1	-1	5	-1	3	3

A

APPENDIX B

ANOVA tables and calibration of methods

ANOVA tables for all proposed GRASP methods

This section compensates the Analysis of Variance (ANOVA) tables for the calibration experiments in Section 5.3.2. The corresponding two GRASP methods, including GR1 and GR2, are proposed in Section 5.2.

As described in Section 5.3.2, several factors involving GRASP methods are investigated in the ANOVA, as follows:

- Container size n: {5, 10, 20, 30, 40, 50, 100, 150,200}
- Constructive parameter size α: {1, 2, 3, 4, 5, 6, 7}
- Local parameter β: {$SETR_{ls}$, $SETR_{ls3}$, $SETR_{ls4}$, $SETR_{ls5}$}

The container size is an uncontrollable factor that is decided by the instances, while the replicate number can be a witness factor that should

always be insignificant. Constructive parameter size (α) and local search parameter (β) are two main factors that should be calibrated. Table B.1 and B.2 display the ANOVA tables for GR1 and GR2, respectively. Recall that if the P-value is less than the chosen significance level, commonly 0.05 at 95% confidence level, it suggests a statistically significant difference among the group means. Consequently, the constructive parameter and local search parameter are proven to be very significant. For more details of the calibration experiments, interested readers are referred to Section 5.3.2.

Table B.1: Analysis of Variance (ANOVA) table for the calibration of GR1

Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Con_Num (n)	438418	9	48713.10	594.43	0.0000
B:Constr_Param(α)	15116.7	6	2519.46	30.74	0.0000
C:LS_Param (β)	235644	4	589111	7188.68	0.0000
D:Replication	6.32	4	1.58	0.02	0.9993
INTERACTIONS					
AB	9679.76	54	179.26	2.19	0.0000
AC	807250.	36	22423.60	273.63	0.0000
AD	61.6625	36	1.71	0.02	1.0000
BC	16641.3	24	693.39	8.46	0.0000
BD	24.7154	24	1.03	0.01	1.0000
CD	12.5068	16	0.78	0.01	1.0000
RESIDUAL	2.0476E6	24986	81.95		
TOTAL (CORRECTED)	5.21269E6	25199			

Table B.2: Analysis of Variance (ANOVA) table for the calibration of GR2

Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Con_Num (n)	453342	9	50371	493.33	0.0000
B:Constr_Param(α)	13380.5	6	2230.08	21.84	0.0000
C:LS_Param (β)	2.23192	3	743974	7286.43	0.0000
D:Replication	5.34	4	1.33	0.01	0.9997
INTERACTIONS					
AB	9157.39	54	169.58	1.66	0.0017
AC	782469	27	28980	283.83	0.0000
AD	47.78	36	1.33	0.01	1.0000
BC	16853.50	18	936.31	9.17	0.0000
BD	28.58	24	1.19	0.01	1.0000
CD	10.12	12	0.84	0.01	1.0000
RESIDUAL	2.03871	19967	102.10		
TOTAL (CORRECTED)	5.08674E6	20160			

ANOVA tables for the proposed IG methods

This section compensates the Analysis of Variance (ANOVA) tables for all proposed IG methods in the calibrations, including a vanilla version IG0 and three proposed IG methods, IG1, IG2, and IG3.

ANOVA table for IG0 in Section 6.3.1

As described in the calibration experiments in Section 6.4.2, several factors involving IG0 are investigated in the ANOVA. They are container size n, destruction size on containers (DesNum), Initial constructive methods (IniMethod) and the replicate number (Replication). The container size is an uncontrollable factor that is decided by the instances, while the replicate number can be a witness factor that should always be insignificant. DesNum and IniMethod are two main factors that should

be calibrated. Table B.3 display that the DesNum and IniMethod are statistically significant, and their means plots with 95% Tukey's Honest Significant Difference (HSD) confidence intervals are shown in Figure 6.4.

Table B.3: Analysis of Variance (ANOVA) table for the calibration of IG0

Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Con_Num (n)	500284	9	55587.10	7501.03	0.0000
B:DesNum	1817.44	4	454.36	61.31	0.0000
C:IniMethod	5816.71	1	5816.71	784.92	0.0000
D:Replication	12.67	9	1.41	0.19	0.9953
INTERACTIONS					
AB	4705.68	36	130.71	17.64	0.0000
AC	12841	9	1426.78	192.53	0.0000
AD	189.35	81	2.34	0.32	1.0000
BC	1058.68	4	264.67	35.71	0.0000
BD	67.96	36	1.89	0.25	1.0000
CD	16.73	9	1.86	0.25	0.9867
RESIDUAL	105238	14201	7.40		
TOTAL	630146	14399			

ANOVA table for IG1 in Section 6.3.2

Recall that the only difference between IG1 and IG0 is that IG1 adopted a new proposed local search method, IG1. A similar ANOVA experiment is carried out on the same factors, including container size n, destruction size on containers (DesNum), Initial constructive methods (IniMethod) and the replicate number (Replication). DesNum and IniMethod are two main factors that should be calibrated as well. Table B.4 is the final ANOVA table where DesNum and IniMethod are very significant. Figure 6.5 shows the means plots with 95% Tukey's Honest Significant Difference (HSD) confidence intervals where both IG0 and IG1 share

the same best levels for these two research factors.

Table B.4: Analysis of Variance (ANOVA) table for the calibration of IG1

Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Con_Num (n)	114735	9	12748.30	4715.44	0.0000
B:DesNum	1161.45	4	290.36	107.40	0.0000
C:IniMethod	764.56	1	764.56	282.80	0.0000
D: Replication	10.10	9	1.12	0.42	0.9279
INTERACTIONS					
AB	2625.96	36	72.94	26.98	0.0000
AC	1224.91	9	136.10	50.34	0.0000
AD	109.75	81	1.35	0.50	0.9999
BC	101.61	4	25.40	9.40	0.0000
BD	20.53	36	0.57	0.21	1.0000
CD	10.78	9	1.20	0.44	0.9123
RESIDUAL	38392.80	14201	2.70		
TOTAL	158688	14399			

ANOVA table for IG2 in Section 6.3.3

The calibration experiments for IG2 introduce a new factor, the destruction size on Input/Output (I/O) assignments (DesIO), parameterized as d_2. The controllable factors in these experiments, including DesNum, IniMethod, and DesIO, are all statistically significant, as shown in the final ANOVA table (Table B.5). We observe that all controllable factors are quite significant, which is as expected. For more detailed details about the calibration experiments, interested are referred to Section 6.4.2.

Table B.5: Analysis of Variance (ANOVA) table for the calibration of IG2

Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Con_Num (n)	143473	9	15941.50	10839.35	0.0000
B:DesNum	7525.13	4	1881.28	1279.17	0.0000
C:IniMethod	2731.14	1	2731.14	1857.03	0.0000
D:DesIO	256.68	3	85.56	58.18	0.0000
E:Replication	13.87	9	1.54	1.05	0.3986
INTERACTIONS					
AB	6939.36	36	192.76	131.07	0.0000
AC	5643	9	627	426.33	0.0000
AD	548.38	27	20.31	13.81	0.0000
AE	105.34	81	1.30	0.88	0.7623
BC	375.2	4	93.80	63.78	0.0000
BD	84.08	12	7.01	4.76	0.0000
BE	25.11	36	0.70	0.47	0.9969
CD	3.41	3	1.14	0.77	0.5090
CE	16.14	9	1.79	1.22	0.2774
DE	9.59	27	0.36	0.24	1.0000
RESIDUAL	84314	57329	1.47		
TOTAL	249504	57599			

ANOVA table for IG3 in Section 6.3.4

The ANOVA experiments of IG3 are executed on factors: the destruction size (DesNum, parameterized as d), on Input/Output (I/O) assignments(DesIO, parameterized as d_2). The only difference between calibrations on IG2 and IG3 is that IG3 has investigated a new factor, PSK. Table B.6 shows the final ANOVA table where all tested factors are statistically significant.

Table B.6: Analysis of Variance (ANOVA) table for the calibration of IG3

Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:Con_Num (n)	305465	9	33940	24109.36	0.0000
B:DesNum	15969	4	3992.31	2835.90	0.0000
C:IniMethod	5511.77	1	5511.77	3915.24	0.0000
D:DesIO	398.75	3	132.92	94.41	0.0000
E:PSK	40.55	1	40.55	28.81	0.0000
F:Replication	15.68	9	1.74	1.24	0.2665
INTERACTIONS					
AB	16003.10	36	444.53	315.77	0.0000
AC	11371.80	9	1263.54	897.54	0.0000
AD	761.03	27	28.19	20.02	0.0000
AE	572.56	9	63.62	45.19	0.0000
AF	128.90	81	1.59	1.13	0.1983
BC	781.79	4	195.45	138.83	0.0000
BD	108.58	12	9.05	6.43	0.0000
BE	24.58	4	6.14	4.36	0.0016
BF	38.45	36	1.07	0.76	0.8507
CD	1.579	3	0.53	0.37	0.7719
CE	0.02	1	0.02	0.01	0.9143
CF	8.91	9	0.99	0.70	0.7063
DE	9.79	3	3.26	2.32	0.0734
DF	14.64	27	0.54	0.39	0.9983
EF	16.86	9	1.87	1.33	0.2148
RESIDUAL	161756	114902	1.40		
TOTAL	513633	115199			

B

APPENDIX C

Design procedure and calibration of LS1

Although Section 6.2.3 has successfully proposed an efficient local search LS1, it is still worth highlighting its design process. LS1 is designed from a proposed local search framework following the basic neighbourhood search with integrations of more advanced techniques that guide searching in neighborhood spaces. We calibrate the local search framework using the Design of Experiments (DOE) with Analysis of Variance (ANOVA) (Montgomery, 2017). It turns out that LS1 outperforms the best among all local search variants obtained by this framework. We believe that the procedure of designing a local search itself is meaningful and also inspiring for even other heuristic domains.

A proposed local search framework

Algorithm 15 shows the local search framework following a basic neighborhood search. To use the framework, several key procedures inside should be specified, including the steps for selecting orders and defining

perturbations. E.g., the perturbation method in Step 3 (Line 5). These procedures always have a huge influence on the efficiency of neighborhood search. A sound search strategy ensures that the search space is explored effectively, avoiding redundant or unnecessary evaluations of the same solutions. All these might speed up the convergence to an optimal or near-optimal solution by guiding the search process efficiently. We have utilized the following strategies in this proposed neighborhood search framework.

- **Sequential** or **random** order for the selection of containers in Step1 and Step2. A sequential order follows the original loading order of containers by the yard crane while the random select containers one by one in random.

- **Swapping** or **insertion** neighborhood for the perturbation between a and b in Step3. The swapping neighborhood swaps a pair of positions of containers a and b. However, the insertion neighborhood inserts container a to the position b of the current solution. E.g., a sequence {1,2,3,4,5} will be {1,4,3,2,5} after the swapping of (2,4), yet {1,3,4,2,5} after the insertion of (2,4).

- **The first improvement** or **the best improvement** for the inner loop in Step6. The first improvement strategy will break here, and the best improvement will not.

- Similarly, the **the first improvement** or **the best improvement** for the outer loop in Step8. The first improvement strategy will break here, and the best improvement will not.

- **Local** or **global** termination at the end of the algorithm in step10. For the local termination, the algorithm terminates after a full round of neighborhood search, while the global termination allows the next round of neighborhood search starting from step1 until no improvements can be found.

In general, we have designed the neighborhood search with variants on 5 levels, which results in a total of $2 \times 2 \times 2 \times 2 \times 2 = 32$ factorial design of the local search methods. All these local search methods work on neighborhoods of the container sequence where the assignment of each container is preserved without loss of generality.

Algorithm 15 : A neighborhood search framework

```
1: Step1: Select a container a in order from a container set A
2: for each container a in A do
3:     Step2: Select a container/position b in order from a container/-
       position set B
4:     for each container/position b in B do
5:         Step3: Perturb the solution with a and b to obtain a new
           solution S(a,b)
6:         Step4: Evaluate the new solution S(a,b)
7:         if the solution is strictly improved then
8:             Step5: Update the currently best solution
9:             Step6: Determine whether to break the inner loop (first/
               best improvement)
10:    Step7: Update the new solution with the final perturbation for
       container a
11:    Step8: Determine whether to break the outer loop (first/ best
       improvement)
12: Step9: Update the best solution so far
13: Step10: Determine whether to stop the algorithm (local/ global
    stop criterion)
```

To illustrate its output, Algorithm 16 presents a very simple swap neighborhood obtained from the local search framework. This local search adopts the **Sequential** order of containers from the incumbent solution, **Swapping** neighborhood, the **best improvement** for both the inner and outer loop, and local termination.

Algorithm 16 : Local search example (incumbent solution S^*, incumbent objective f^*)

```
1: for each container i in S* do
2:     for each container j in S* do
3:         Solution S ← swap the container pair (i, j)
4:         f is the objective value of solution S
5:         if f < f* then
6:             S* = S
7:             f* = f
```

Calibration of the local search framework

As aforementioned, we calibrate the local search framework in a classical Iterated greedy of Ruiz and Stützle (2007) with the Design of Experiments (Montgomery, 2017). Table C.1 presents the ANOVA table of the neighborhood search framework in Section C. In summary, there are a total of five factors calibrated, including SelectionOrder, InnerImprovement, OuterImprovement, TerminationType, and PerturbationType. Note that factor Replication is a witness factor defined with random seeds, where this factor should always be statistically insignificant. The factor ContaineSize, as defined by the instances, is an uncontrollable factor.

Table C.1 shows the results of the ANOVA experiments. It turns out that except for the replication number, all other factors are statistically significant which is excepted. The investigation of these factors is helpful in recognizing the most promising local search method. Step by step, we set the factor for each factor in the order of decreasing significance coefficiency (p-Value) as follows:

- InnerImprovement
  We recognize the InnerImprovement as the most significant among all five factors. Figure C.1 shows that the best improvement in the inner loop is more significant than the first improvement. Figure C.2 shows the interactions with the container size where the best

Table C.1: Analysis of Variance (ANOVA) table for the calibration of the proposed local search framework

Source	Sum of Squares	Df	Mean Square	F-Ratio	p-Value
Main effects					
A:ContaineSize	162008.00	7	23144.10	882.80	0.0000
B:SelectionOrder	433.24	1	433.24	16.53	0.0000
C:InnerImprovement	11034.20	1	11034.20	420.88	0.0000
D:OuterImprovement	9036.92	1	9036.92	344.70	0.0000
E:TerminationType	442.84	1	442.84	16.89	0.0000
F:PerturbationType	5575.94	1	5575.94	212.69	0.0000
G:Replication	9.96	4	2.49	0.09	0.9841
Interactions					
AB	729.64	7	104.23	3.98	0.0002
AC	15404.30	7	2200.61	83.94	0.0000
AD	13373.60	7	1910.52	72.87	0.0000
AE	306.82	7	43.83	1.67	0.1108
AF	9542.05	7	1363.15	52.00	0.0000
AG	76.47	28	2.73	0.10	1.0000
BC	716.97	1	716.97	27.35	0.0000
BD	1140.24	1	1140.24	43.49	0.0000
BE	1.87	1	1.87	0.07	0.7895
BF	41.98	1	41.98	1.60	0.2057
BG	1.09	4	0.27	0.01	0.9998
CD	9965.73	1	9965.73	380.13	0.0000
CE	242.86	1	242.86	9.26	0.0023
CF	320.31	1	320.31	12.22	0.0005
CG	3.33	4	0.83	0.03	0.9981
DE	706.01	1	706.01	26.93	0.0000
DF	19.0651	1	19.0651	0.73	0.3938
DG	6.27541	4	1.56885	0.06	0.9934
EF	13.3276	1	13.3276	0.51	0.4758
EG	3.32656	4	0.83164	0.03	0.9981
FG	8.30298	4	2.07575	0.08	0.9887
Residual	399806	15250	26.2168		
Total (corrected)	640971	15359			

improvements are increasingly significant with the container sizes.

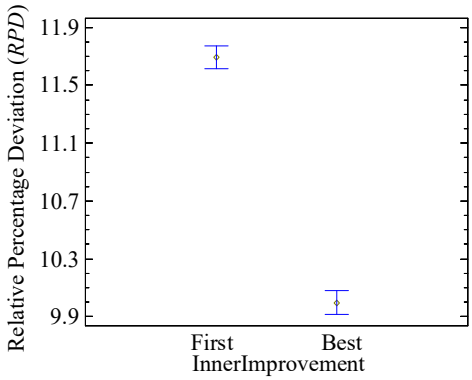


Figure C.1: Means plots for the factor InnerImprovement of the neighborhood search framework in the ANOVA calibration with 95% Tukey's Honest Significant Difference (HSD) confidence intervals. Elaborated by the author.

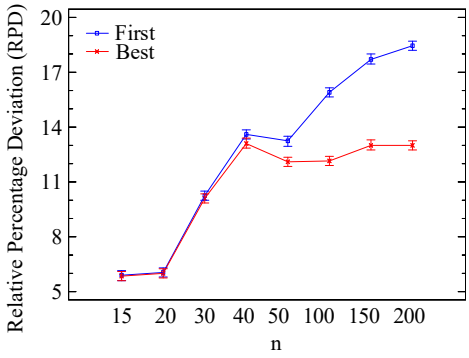


Figure C.2: Means plot of the interaction between container size and the first/best improvements in the inner loop of the neighborhood search framework with 95% Tukey's Honest Significant Difference (HSD) confidence intervals. Elaborated by the author.

- OuterImprovement

  OuterImprovement is the second significant factor in the ANOVA Table C.1. Note that the F-ratio (344.7) of OuterImprovement

is smaller than its interaction CD (380.13) with InnerImprovement, which implies that this factor is strongly affected by the former factor. Figure C.3 shows the interaction plots between two factors, where the first improvement outperforms OuterImprovement. Figure C.4 illustrates the interactions with the container size where the best improvements are increasingly significant with the container sizes.

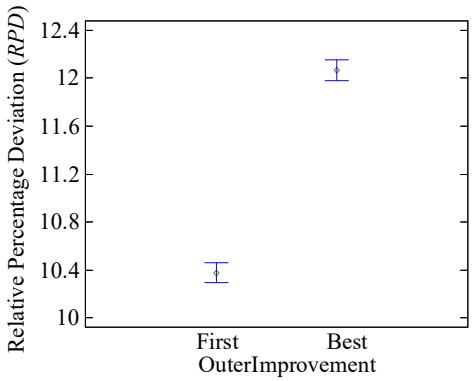


Figure C.3: Means plots for the factor OuterImprovement of the neighborhood search framework in the ANOVA calibration with 95% Tukey's Honest Significant Difference (HSD) confidence intervals. Elaborated by the author.

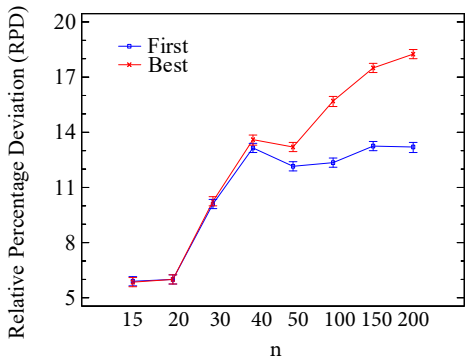


Figure C.4: Means plot of the interaction between container size and the first/best improvements in the outer loop of the neighborhood search framework with 95% Tukey's Honest Significant Difference (HSD) confidence intervals. Elaborated by the author.

- PerturbationType
  Based on the F-ratio of factors and their interactions, the next factor that should be calibrated is the PerturbationType. The operations, swapping and insertion result in huge differences between each pair of containers in the neighborhood search framework. Figure C.5 shows that the swapping neighborhood works significantly better than the insertion neighborhood. This can also be seen from the interaction plot Figure C.5 with the container size.

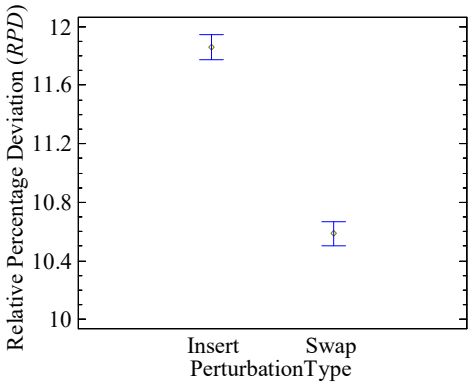


Figure C.5: Means plots for the factor InnerImprovement of the neighborhood search framework in the ANOVA calibration with 95% Tukey's Honest Significant Difference (HSD) confidence intervals. Elaborated by the author.

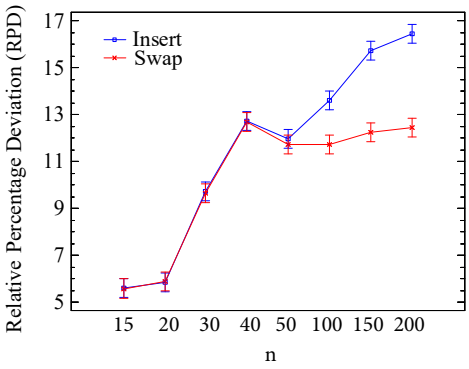


Figure C.6: Means plot of the interaction between container size and the first/best improvements in the inner loop of the neighborhood search framework with 95% Tukey's Honest Significant Difference (HSD) confidence intervals. Elaborated by the author.

- SelectionOrder

  There are two factors left, SelectionOrder and TerminationType. Note that all their F-ratio are smaller than their interactions with OuterImprovement, where the F-ratio between SelectionOrder and is 43.49 which is bigger than that of TerminationType, 26.93.

In this case, the SelectionOrder is calibrated first. Figure C.7 displays the corresponding interaction plots where the random order works slightly better when we adopt the first improvement policy for OuterImprovement.

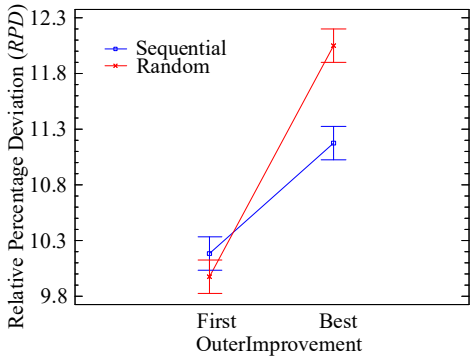


Figure C.7: Means plot of the interaction between OuterImprovement and SelectionOrder with 95% Tukey's Honest Significant Difference (HSD) confidence intervals. Elaborated by the author.

- TerminationType
  Similarly, the last factor TerminationType is highly correlated with OuterImprovement where its F-ratio is smaller than that of the interaction with OuterImprovement. Figure C.8 displays the interaction plots. It turns out that global termination outperforms local termination, thereby being chosen in LS1.

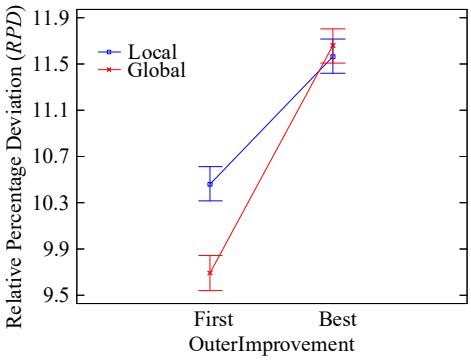


Figure C.8: Means plot of the interaction between OuterImprovement and TerminationType with 95% Tukey's Honest Significant Difference (HSD) confidence intervals. Elaborated by the author.

In summary, adopting all these levels results in a powerful local search, LS1, which has been introduced in Section 6.2.3 and has been demonstrated to outperform other proposals significantly in Section 6.4.2.

Bibliography

Abou Kasm, O. Diabat, A., 2019. The quay crane scheduling problem with non-crossing and safety clearance constraints: An exact solution approach. Computers & Operations Research 107, 189–199. (Cited on page 32).

Abou Kasm, O., Diabat, A. Cheng, T., 2020. The integrated berth allocation, quay crane assignment and scheduling problem: mathematical formulations and a case study. Annals of Operations Research 291 (1), 435–461. (Cited on page 18).

Agra, A. Oliveira, M., 2018. Mip approaches for the integrated berth allocation and quay crane assignment and scheduling problem. European Journal of Operational Research 264 (1), 138–148. (Cited on page 20).

Al-Dhaheri, N. Diabat, A., 2015. The quay crane scheduling problem. Journal of Manufacturing Systems 36, 87–94. (Cited on page 19).

Al-Dhaheri, N., Jebali, A. Diabat, A., 2016. The quay crane scheduling problem with nonzero crane repositioning time and vessel stability constraints. Computers & Industrial Engineering 94, 230–244. (Cited on page 31).

Alvarez-Valdés, R., Crespo, E., Tamarit, J. M. Villa, F., 2008. Grasp and path relinking for project scheduling under partially renewable resources. European Journal of Operational Research 189 (3), 1153–1170. (Cited on page 94).

Arnold, F. Sörensen, K., 2019. Knowledge-guided local search for the vehicle routing problem. Computers & Operations Research 105, 32–46. (Cited on page 126).

Avci, M. Topaloglu, S., 2017. A multi-start iterated local search algorithm for the generalized quadratic multiple knapsack problem. Computers & Operations Research 83, 54–65. (Cited on page 117).

Baykasoğlu, A., Madenoğlu, F. S. Hamzadayı, A., 2020. Greedy randomized adaptive search for dynamic flexible job-shop scheduling. Journal of Manufacturing Systems 56, 425–451. (Cited on page 94).

Bellwood, P., Fox, J. J. Tyron, D., 2006. The Austronesians: historical and comparative perspectives. ANU Press.
URL `https://books.google.es/books?id=9uyuHAXBuRkC` (Cited on page 2).

Bierwirth, C. Meisel, F., 2009. A fast heuristic for quay crane scheduling with interference constraints. Journal of Scheduling 12, 345–360. (Cited on page 31).

Bierwirth, C. Meisel, F., 2010. A survey of berth allocation and quay crane scheduling problems in container terminals. European Journal of Operational Research 202 (3), 615–627. (Cited on pages xx, 16, 17, and 32).

Bierwirth, C. Meisel, F., 2015. A follow-up survey of berth allocation and quay crane scheduling problems in container terminals. European Journal of Operational Research 244 (3), 675–689. (Cited on page 32).

Blazewicz, J., Ecker, K. H., Pesch, E., Schmidt, G. Weglarz, J., 1997. Scheduling computer and manufacturing processes. Journal of the Operational Research Society 48 (6), 659–659. (Cited on page 29).

Cao, J. X., Lee, D.-H., Chen, J. H. Shi, Q., 2010. The integrated yard truck and yard crane scheduling problem: Benders' decomposition-based methods. Transportation Research Part E: Logistics and Transportation Review 46 (3), 344–353. (Cited on pages 30, 34, 35, 36, 40, 47, and 64).

Carlo, H. J., Vis, I. F. Roodbergen, K. J., 2014. Storage yard operations in container terminals: Literature overview, trends, and research directions. European Journal of Operational Research 235 (2), 412–430. (Cited on pages 13 and 14).

Cartenì, A. De Luca, S., 2012. Tactical and strategic planning for a container terminal: Modelling issues within a discrete event simulation approach. Simulation Modelling Practice and Theory 21 (1), 123–145. (Cited on page 17).

Caserta, M., Schwarze, S. Voß, S., 2011a. Container rehandling at maritime container terminals. Handbook of Terminal Planning 49, 247–269. (Cited on pages 20, 35, and 47).

Caserta, M., Schwarze, S. Voß, S., 2020. Container rehandling at maritime container terminals: A literature update. Springer, ISBN: 978-3-030-39990-0.
URL https://books.google.es/books?hl=zh-CN&lr=&id=8xL9DwAAQBAJ (Cited on page 33).

Caserta, M., Voß, S. Sniedovich, M., 2011b. Applying the corridor method to a blocks relocation problem. OR spectrum 33, 915–929. (Cited on page 33).

Casson, L., 1995. Ships and seamanship in the ancient world, 1st Edition. Baltimore: Johns Hopkins University Press.
URL https://books.google.es/books?hl=zh-CN&lr=&id=sDpMh0gK2OUC (Cited on page 2).

Chang, D., Jiang, Z., Yan, W. He, J., 2011. Developing a dynamic rolling-horizon decision strategy for yard crane scheduling. Advanced Engineering Informatics 25 (3), 485–494. (Cited on pages 34, 35, 40, 47, and 64).

Chaovalitwongse, W., Kim, D. Pardalos, P. M., 2003. Grasp with a new local search scheme for vehicle routing problems with time windows. Journal of Combinatorial Optimization 7, 179–207. (Cited on page 94).

Chen, L. Langevin, A., 2011. Multiple yard cranes scheduling for loading operations in a container terminal. Engineering Optimization 43 (11), 1205–1221. (Cited on pages 34, 35, 36, 40, and 64).

Chu, F., He, J., Zheng, F. Liu, M., 2019. Scheduling multiple yard cranes in two adjacent container blocks with position-dependent processing times. Computers & Industrial Engineering 136, 355–365. (Cited on pages 34, 35, 36, 40, and 47).

Chung, S. H. Chan, F. T., 2013. A workload balancing genetic algorithm for the quay crane scheduling problem. International Journal of Production Research 51 (16), 4820–4834. (Cited on page 32).

Colmenar, J. M., Greistorfer, P., Martí, R. Duarte, A., 2016. Advanced greedy randomized adaptive search procedure for the obnoxious p-median problem. European Journal of Operational Research 252 (2), 432–442. (Cited on page 94).

Colombo, F., Cordone, R. Lulli, G., 2015. A variable neighborhood search algorithm for the multimode set covering problem. Journal of Global Optimization 63, 461–480. (Cited on page 94).

Cuervo, D. P., Goos, P., Sörensen, K. Arráiz, E., 2014. An iterated local search algorithm for the vehicle routing problem with backhauls. European Journal of Operational Research 237 (2), 454–464. (Cited on page 117).

Diabat, A. Theodorou, E., 2014. An integrated quay crane assignment and scheduling problem. Computers & Industrial Engineering 73, 115–123. (Cited on pages 19 and 31).

Dik, G. Kozan, E., 2017. A flexible crane scheduling methodology for container terminals. Flexible Services and Manufacturing Journal 29, 64–96. (Cited on page 32).

Dorndorf, U. Schneider, F., 2010. Scheduling automated triple cross-over stacking cranes in a container yard. OR Spectrum 32 (3), 617–632. (Cited on pages 34, 35, 36, and 47).

Dragović, B., Tzannatos, E. Park, N. K., 2017. Simulation modelling in ports and container terminals: literature overview and analysis by research field, application area and tool. Flexible Services and Manufacturing Journal 29 (1), 4–34. (Cited on page 17).

Fanjul-Peyro, L. Ruiz, R., 2010. Iterated greedy local search methods for unrelated parallel machine scheduling. European Journal of Operational Research 207 (1), 55–69. (Cited on page 117).

Fazi, S., Fransoo, J. C., Van Woensel, T. Dong, J.-X., 2020. A variant of the split vehicle routing problem with simultaneous deliveries and pickups for inland container shipping in dry-port based systems. Transportation Research Part E: Logistics and Transportation Review 142, 102057. (Cited on page 19).

Feo, T. Resende, M., 1995. Greedy randomized adaptive search procedures. Journal of Global Optimization 6, 109–133. (Cited on page 93).

Finlay, R., 2008. The voyages of zheng he: ideology, state power, and maritime trade in ming china. Journal of the Historical Society 8 (3), 327–347. (Cited on page 3).

Fleszar, K., Osman, I. H. Hindi, K. S., 2009. A variable neighbourhood search algorithm for the open vehicle routing problem. European Journal of Operational Research 195 (3), 803–809. (Cited on page 125).

Fu, Y.-M., Diabat, A. Tsai, I.-T., 2014. A multi-vessel quay crane assignment and scheduling problem: Formulation and heuristic solution approach. Expert Systems with Applications 41 (15), 6959–6965. (Cited on page 32).

Galle, V., Barnhart, C. Jaillet, P., 2018. Yard crane scheduling for container storage, retrieval, and relocation. European Journal of Operational Research 271 (1), 288–316. (Cited on pages 20, 30, 33, 34, 35, 36, 37, 38, 52, and 72).

Gao, Z., Zhang, L., Yu, P., Zhang, Z. Li, Z., 2024. A matheuristic-oriented iterated greedy algorithm for multi-mode resource-constrained project scheduling problem under uncertainty. Computers & Industrial Engineering 193, 110333. (Cited on page 117).

García, S., Molina, D., Lozano, M. Herrera, F., 2009. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the cec'2005 special session on real parameter optimization. Journal of Heuristics 15, 617–644. (Cited on page 143).

García-Martínez, C., Rodriguez, F. J. Lozano, M., 2014. Tabu-enhanced iterated greedy algorithm: a case study in the quadratic multiple knapsack problem. European Journal of Operational Research 232 (3), 454–463. (Cited on page 117).

Gharehgozli, A., Yu, Y., de Koster, R. Du, S., 2019. Sequencing storage and retrieval requests in a container block with multiple open locations. Transportation Research Part E: Logistics and Transportation Review 125, 261–284. (Cited on pages 34, 35, 38, 40, 52, 58, and 75).

Gharehgozli, A. H., Laporte, G., Yu, Y. de Koster, R., 2015. Scheduling twin yard cranes in a container block. Transportation Science 49 (3), 686–705. (Cited on pages 33, 34, 35, 36, 38, 40, and 58).

Gharehgozli, A. H., Vernooij, F. G. Zaerpour, N., 2017. A simulation study of the performance of twin automated stacking cranes at a seaport container

terminal. European Journal of Operational Research 261 (1), 108–128. (Cited on page 36).

Gharehgozli, A. H., Yu, Y., de Koster, R. Udding, J. T., 2014. An exact method for scheduling a yard crane. European Journal of Operational Research 235 (2), 431–447. (Cited on pages 20, 23, 33, 34, 35, 37, 38, 40, 41, 52, 58, and 75).

Giallombardo, G., Moccia, L., Salani, M. Vacca, I., 2010. Modeling and solving the tactical berth allocation problem. Transportation Research Part B: Methodological 44 (2), 232–245. (Cited on page 17).

González-Neira, E. M. Montoya-Torres, J. R., 2017. A grasp meta-heuristic for the hybrid flowshop scheduling problem. Journal of Decision systems 26 (3), 294–306. (Cited on page 94).

Guan, Y. Cheung, R. K., 2004. The berth allocation problem: models and solution methods. OR Spectrum 26 (1), 75–92. (Cited on page 18).

Hai, D., Xu, J., Duan, Z. Chen, C., 2020. Effects of underground logistics system on urban freight traffic: A case study in shanghai, china. Journal of cleaner production 260, 121019. (Cited on pages 6 and 12).

Han, X., Lu, Z. Xi, L., 2010. A proactive approach for simultaneous berth and quay crane scheduling problem with stochastic arrival and handling time. European Journal of Operational Research 207 (3), 1327–1340. (Cited on pages 31 and 32).

Hansen, P. Mladenović, N., 1997. Variable neighborhood search for the p-median. Location Science 5 (4), 207–226. (Cited on page 125).

Hatami, S., Ruiz, R. Andrés-Romano, C., 2015. Heuristics and metaheuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times. International Journal of Production Economics 169, 76–88. (Cited on pages 117 and 127).

He, J., Chang, D., Mi, W. Yan, W., 2010. A hybrid parallel genetic algorithm for yard crane scheduling. Transportation Research Part E: Logistics and Transportation Review 46 (1), 136–155. (Cited on pages 34, 35, 40, 47, and 64).

He, J., Huang, Y. Yan, W., 2015a. Yard crane scheduling in a container terminal for the trade-off between efficiency and energy consumption. Advanced Engineering Informatics 29 (1), 59–75. (Cited on pages 9 and 71).

He, J., Huang, Y., Yan, W. Wang, S., 2015b. Integrated internal truck, yard crane and quay crane scheduling in a container terminal considering energy consumption. Expert Systems with Applications 42 (5), 2464–2487. (Cited on pages 32 and 71).

Hernandez, A. J., CHO, S. W. PAK, M. S., 2012. Fuel consumption within cargo operations at the port industry a simulation analysis on the case of s port company in the uk. The Asian Journal of Shipping and Logistics 28 (2), 227–254. (Cited on page 9).

Hoos, H. H. Stützle, T., 2005. Scheduling problems. In: Hoos, H. H. Stützle, T. (Eds.), Stochastic Local Search. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann, San Francisco, pp. 417–465. (Cited on page 29).

Hsu, H.-P., Tai, H.-H., Wang, C.-N. Chou, C.-C., 2021. Scheduling of collaborative operations of yard cranes and yard trucks for export containers using hybrid approaches. Advanced Engineering Informatics 48, 101292. (Cited on pages 30, 34, 35, 36, 40, and 47).

Hu, Z.-H., Sheu, J.-B. Luo, J. X., 2016. Sequencing twin automated stacking cranes in a block at automated container terminal. Transportation Research Part C: Emerging Technologies 69, 208–227. (Cited on pages 33, 34, 35, 36, and 40).

Jin, J. G., Lee, D.-H. Hu, H., 2015. Tactical berth and yard template design at container transshipment terminals: A column generation based approach. Transportation Research Part E: Logistics and Transportation Review 73, 168–184. (Cited on page 17).

Jung, D. H., Park, Y.-M., Lee, B. K., Kim, K. H. Ryu, K. R., 2006. A quay crane scheduling method considering interference of yard cranes in container terminals. In: Gelbukh, A. Reyes-Garcia, C. A. (Eds.), MICAI 2006: Advances in Artificial Intelligence. Springer, Berlin, Heidelberg, pp. 461–471, ISBN:978-3-540-49058-6. (Cited on page 94).

Karabulut, K. Tasgetiren, M. F., 2014. A variable iterated greedy algorithm for the traveling salesman problem with time windows. Information Sciences 279, 383–395. (Cited on page 117).

Kaveshgar, N. Huynh, N., 2015. Integrated quay crane and yard truck scheduling for unloading inbound containers. International Journal of Production Economics 159, 168–177. (Cited on pages 30, 32, and 36).

Kaveshgar, N., Huynh, N. Rahimian, S. K., 2012. An efficient genetic algorithm for solving the quay crane scheduling problem. Expert Systems with Applications 39 (18), 13108–13117. (Cited on page 32).

Kay, G., 2021. From toilet paper to coffee, here are some of the products that could soon be in short supply because of the suez canal blockage. URL https://www.businessinsider.com/toilet-paper-coffee-products-delayed-suez-canal-blockage-impact-2021-3 (Cited on page 3).

Kim, K. H. Kim, K. Y., 1999. An optimal routing algorithm for a transfer crane in port container terminals. Transportation Science 33 (1), 17–33. (Cited on pages 30, 37, and 58).

Kim, K. H. Park, Y.-M., 2004. A crane scheduling method for port container terminals. European Journal of Operational Research 156 (3), 752–768. (Cited on pages 30, 58, and 94).

Kizilay, D. Eliiyi, D. T., 2021. A comprehensive review of quay crane scheduling, yard operations and integrations thereof in container terminals. Flexible Services and Manufacturing Journal 33 (1), 1–42. (Cited on page 32).

Konecranes, 2023. Konecranes rmg: a concept, not just a crane. URL https://www.konecranes.com/sites/default/files/2021-06/RMG_Typical_Tech_Specs.pdf (Cited on pages xix, xxv, 11, and 12).

Kontoravdis, G. Bard, J. F., 1995. A grasp for the vehicle routing problem with time windows. ORSA journal on Computing 7 (1), 10–23. (Cited on page 94).

Kuhn, H. W., 1955. The hungarian method for the assignment problem. Naval research logistics quarterly 2 (1-2), 83–97. (Cited on page 38).

Laguna, M. Martí, R., 2001. A grasp for coloring sparse graphs. Computational optimization and applications 19, 165–178. (Cited on page 94).

Lalla-Ruiz, E., González-Velarde, J. L., Melián-Batista, B. Moreno-Vega, J. M., 2014. Biased random key genetic algorithm for the tactical berth allocation problem. Applied Soft Computing 22, 60–76. (Cited on page 17).

Lee, B. K. Kim, K. H., 2013. Optimizing the yard layout in container terminals. OR Spectrum 35 (2), 363–398. (Cited on page 13).

Lee, D.-H., Chen, J. H. Cao, J. X., 2011a. Quay crane scheduling for an indented berth. Engineering Optimization 43 (9), 985–998. (Cited on page 32).

Lee, D.-H., Jin, J. G. Chen, J. H., 2011b. Integrated bay allocation and yard crane scheduling problem for transshipment containers. Transportation Research Record: Journal of the Transportation Research Board 2222 (1), 63–71. (Cited on page 36).

Lee, D.-H., Wang, H. Q. Miao, L., 2008. Quay crane scheduling with non-interference constraints in port container terminals. Transportation Research Part E: Logistics and Transportation Review 44 (1), 124–135. (Cited on pages 31 and 32).

Lee, H. F. Schaefer, S. K., 1997. Sequencing methods for automated storage and retrieval systems with dedicated storage. Computers & Industrial Engineering 32 (2), 351–362. (Cited on pages 34, 35, and 47).

Lee, Y. Hsu, N.-Y., 2007. An optimization model for the container premarshalling problem. Computers & Operations Research 34 (11), 3295–3313. (Cited on pages 20 and 32).

Legato, P., Trunfio, R. Meisel, F., 2012. Modeling and solving rich quay crane scheduling problems. Computers & Operations Research 39 (9), 2063–2078. (Cited on pages 19 and 31).

Lehnfeld, J. Knust, S., 2014. Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. European Journal of Operational Research 239 (2), 297–312. (Cited on pages 32 and 33).

Li, J.-Q., Pan, Q.-K. Mao, K., 2015. A hybrid fruit fly optimization algorithm for the realistic hybrid flowshop rescheduling problem in steelmaking systems. IEEE Transactions on Automation Science and Engineering 13 (2), 932–949. (Cited on page 117).

Li, W., Goh, M., Wu, Y., Petering, M. *et al.*, 2012. A continuous time model for multiple yard crane scheduling with last minute job arrivals. International Journal of Production Economics 136 (2), 332–343. (Cited on pages 34, 35, and 36).

Li, W., Wu, Y., Petering, M., Goh, M. Souza, R. d., 2009. Discrete time model and algorithms for container yard crane scheduling. European Journal of Operational Research 198 (1), 165–172. (Cited on pages 12, 15, 20, 34, 35, and 36).

Liang, C., Huang, Y. Yang, Y., 2009. A quay crane dynamic scheduling problem by hybrid evolutionary algorithm for berth allocation planning. Computers & Industrial Engineering 56 (3), 1021–1028. (Cited on page 31).

Liang, C.-J., Chen, M., Gen, M. Jo, J., 2014. A multi-objective genetic algorithm for yard crane scheduling problem with multiple work lines. Journal of Intelligent Manufacturing 25 (5), 1013–1024. (Cited on pages 34, 35, 40, and 47).

Liebherr, 2023. Rubber tyre gantry cranes.
URL https://www.liebherr.com/en/aus/products/maritime-cranes/port-equipment/rubber-tyre-stacking-cranes/rubber-tyre-gantry-cranes.html (Cited on pages xix, xxv, 11, and 12).

Lim, A., Rodrigues, B., Xiao, F. Zhu, Y., 2004. Crane scheduling with spatial constraints. Naval Research Logistics 51 (3), 386–406. (Cited on page 71).

Liu, C.-I., Jula, H., Vukadinovic, K. Ioannou, P., 2004. Automated guided vehicle system for two container yard layouts. Transportation Research Part C: Emerging Technologies 12 (5), 349–368. (Cited on page 13).

Ma, S., Li, H., Zhu, N. Fu, C., 2021. Stochastic programming approach for unidirectional quay crane scheduling problem with uncertainty. Journal of Scheduling 24, 137–174. (Cited on page 32).

Mahdi, W., 1999. The dispersal of austronesian boat forms in the indian ocean. Archaeology & language III, Artefacts, Languages and Texts 34, 144–208. (Cited on page 2).

Marinakis, Y., Migdalas, A. Pardalos, P. M., 2005. Expanding neighborhood grasp for the traveling salesman problem. Computational Optimization and Applications 32, 231–257. (Cited on page 94).

Meisel, F. Bierwirth, C., 2013. A framework for integrated berth allocation and crane operations planning in seaport container terminals. Transportation Science 47 (2), 131–147. (Cited on page 31).

Missaoui, A. Ruiz, R., 2022. A parameter-less iterated greedy method for the hybrid flowshop scheduling problem with setup times and due date windows. European Journal of Operational Research 303 (1), 99–113. (Cited on page 117).

Mladenović, N., Labbé, M. Hansen, P., 2003. Solving the p-center problem with tabu search and variable neighborhood search. Networks 42 (1), 48–64. (Cited on page 125).

Moccia, L., Cordeau, J.-F., Gaudioso, M. Laporte, G., 2006. A branch-and-cut algorithm for the quay crane scheduling problem in a container terminal. Naval Research Logistics (NRL) 53 (1), 45–59. (Cited on page 32).

Montgomery, D. C., 2017. Design and analysis of experiments, tenth ed. Wiley, ISBN:9781119113478.
URL https://books.google.es/books?id=Py7bDgAAQBAJ (Cited on pages 26, 81, 100, 135, 137, 148, 167, and 170).

Montoya-Torres, J. R., Aponte, A. Rosas, P., 2011. Applying grasp to solve the multi-item three-echelon uncapacitated facility location problem. Journal of the Operational Research Society 62 (2), 397–406. (Cited on page 94).

Moura, A. Oliveira, J. F., 2009. An integrated approach to the vehicle routing and container loading problems. OR Spectrum 31 (4), 775–800. (Cited on page 19).

Narasimhan, A. Palekar, U. S., 2002. Analysis and algorithms for the transtainer routing problem in container port operations. Transportation Science 36 (1), 63–78. (Cited on page 37).

Ng, W., 2005. Crane scheduling in container yards with inter-crane interference. European Journal of operational research 164 (1), 64–78. (Cited on pages 34 and 47).

Ng, W. Mak, K., 2005a. An effective heuristic for scheduling a yard crane to handle jobs with different ready times. Engineering optimization 37 (8), 867–877. (Cited on pages 12, 15, 20, 34, 37, 47, 58, and 64).

Ng, W. C. Mak, K. L., 2005b. Yard crane scheduling in port container terminals. Applied Mathematical Modelling 29 (3), 263–276. (Cited on pages 20, 34, 36, 37, 47, 58, and 64).

Ng, W. C. Mak, K. L., 2006. Quay crane scheduling in container terminals. Engineering Optimization 38 (6), 723–737. (Cited on page 31).

Nucamendi-Guillén, S., Angel-Bello, F., Martínez-Salazar, I. Cordero-Franco, A. E., 2018. The cumulative capacitated vehicle routing problem: New

formulations and iterated greedy algorithms. Expert Systems with Applications 113, 315–327. (Cited on page 117).

Pan, Q.-K., Ruiz, R. Alfaro-Fernández, P., 2017. Iterated search methods for earliness and tardiness minimization in hybrid flowshops with due windows. Computers & Operations Research 80, 50–60. (Cited on page 117).

Pan, Q.-K., Tasgetiren, M. F. Liang, Y.-C., 2008. A discrete differential evolution algorithm for the permutation flowshop scheduling problem. Computers and Industrial Engineering 55 (4), 795–816. (Cited on page 117).

Park, Y.-M. Kim, K. H., 2003. A scheduling method for berth and quay cranes. OR spectrum 25 (1), 1–23. (Cited on page 31).

Parreño Torres, C., 2020. Improving container terminal efficiency: New models and algorithms for premarshalling and stowage problems. Ph.D. thesis, Universitat de València, Valencia.
URL https://roderic.uv.es/rest/api/core/bitstreams/d0362e05-ed2c-4378-bfa2-395e89826016/content (Cited on page 6).

Pawlik, T., Stemmler, L., Baird, A. J. Helch, M., 2011. The value of container terminal investment to ocean carrier strategy. Maritime Economics & Logistics 13 (3), 319–341. (Cited on page 17).

Rajkumar, M., Asokan, P., Anilkumar, N. Page, T., 2011. A grasp algorithm for flexible job-shop scheduling problem with limited resource constraints. International Journal of Production Research 49 (8), 2409–2423. (Cited on page 94).

Resende, M. Ribeiro, C., 2016. Optimization by GRASP: Greedy Randomized Adaptive Search Procedures. Springer New York, ISBN:9781493965281.
URL https://books.google.es/books?id=B7LpjwEACAAJ (Cited on page 94).

Riahi, V., Newton, M. H., Su, K. Sattar, A., 2019. Constraint guided accelerated search for mixed blocking permutation flowshop scheduling. Computers & Operations Research 102, 102–120. (Cited on page 126).

Ribas, I., Companys, R. Tort-Martorell, X., 2011. An iterated greedy algorithm for the flowshop scheduling problem with blocking. Omega 39 (3), 293–301. (Cited on page 117).

Rodrigues, F. Agra, A., 2022. Berth allocation and quay crane assignment/scheduling problem under uncertainty: A survey. European Journal of Operational Research 303 (2), 501–524. (Cited on pages 18 and 20).

Rodriguez, F. J., Lozano, M., Blum, C. Garcia-Martinez, C., 2013. An iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem. Computers & operations research 40 (7), 1829–1841. (Cited on page 117).

Ruiz, R., Pan, Q.-K. Naderi, B., 2019. Iterated greedy methods for the distributed permutation flowshop scheduling problem. Omega 83, 213–222. (Cited on page 117).

Ruiz, R. Stützle, T., 2007. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European journal of Operational Rsearch 177 (3), 2033–2049. (Cited on pages xxvii, 24, 116, 117, 118, 119, 120, 121, 127, 128, 129, 130, 131, and 170).

Ruiz, R. Stützle, T., 2008. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. European Journal of Operational Research 187 (3), 1143–1159. (Cited on page 117).

Saini, S., Roy, D. de Koster, R., 2017. A stochastic model for the throughput analysis of passing dual yard cranes. Computers & Operations Research 87, 40–51. (Cited on page 36).

Sammarra, M., Cordeau, J.-F., Laporte, G. Monaco, M. F., 2007. A tabu search heuristic for the quay crane scheduling problem. Journal of Scheduling 10, 327–336. (Cited on page 32).

Schmidt, G. Wilhelm, W. E., 2000. Strategic, tactical and operational decisions in multi-national logistics networks: a review and discussion of modelling issues. International Journal of Production Research 38 (7), 1501–1523. (Cited on page 17).

Shamika, N. S., Jan, H., Regina, A., Gonzalo, A. *et al.*, Dec 2021. Review of maritime transport 2021. Tech. rep., United Nations Conference on Trade and Development (UNCTAD), Geneva.
URL `https://unctad.org/system/files/official-document/rmt2021_en_0.pdf` (Cited on pages xix, 4, and 5).

Shamika, N. S., Jan, H., Regina, A., Gonzalo, A. *et al.*, Nov 2022. Review of maritime transport 2022. navigating stormy waters. Tech. rep., United

Nations Conference on Trade and Development (UNCTAD), Geneva, ISBN:978-92-1-113073-7.
URL https://unctad.org/system/files/official-document/rmt2022_en.pdf (Cited on pages xix, 4, and 6).

Shamika, N. S., Jan, H., Regina, A., Gonzalo, A. *et al.*, Nov 2023. Review of maritime transport 2023. towards a green and just transition. Tech. rep., United Nations Conference on Trade and Development (UNCTAD), Geneva, ISBN:978-92-1-002886-8.
URL https://unctad.org/system/files/official-document/rmt2023_en.pdf (Cited on pages xix, 4, 5, and 21).

Speer, U. Fischer, K., 2017. Scheduling of different automated yard crane systems at container terminals. Transportation Science 51 (1), 305–324. (Cited on pages 36, 52, and 77).

Stahlbock, R. Voß, S., 2008a. Operations research at container terminals: a literature update. OR spectrum 30, 1–52. (Cited on pages 12 and 17).

Stahlbock, R. Voß, S., 2008b. Vehicle routing problems and container terminal operations–an update of research. The Vehicle Routing Problem: Latest Advances and New Challenges, 551–589. (Cited on page 19).

Steenken, D., Voß, S. Stahlbock, R., 2004. Container terminal operation and operations research-a classification and literature review. OR Spectrum 26 (1), 3–49. (Cited on page 13).

Stein, S. K., 2017. The Sea in World History: Exploration, Travel, and Trade. Vol. 1. ABC-CLIO.
URL https://books.google.es/books?hl=en&lr=&id=VanOEAAAQBAJ (Cited on page 2).

Talley, W. K., 2000. Ocean container shipping: impacts of a technological improvement. Journal of Economic Issues 34 (4), 933–948. (Cited on page 6).

Tanaka, S. Takii, K., 2015. A faster branch-and-bound algorithm for the block relocation problem. IEEE Transactions on Automation Science and Engineering 13 (1), 181–190. (Cited on pages 21 and 33).

Tavakkoli-Moghaddam, R., Makui, A., Salahi, S., Bazzazi, M. Taheri, F., 2009. An efficient algorithm for solving a new mathematical model for a quay crane scheduling problem in container ports. Computers & Industrial Engineering 56 (1), 241–248. (Cited on page 32).

UNCTADStat, 2022. Container port throughput, annual table summary. URL https://unctadstat.unctad.org/ (Cited on pages xix, 8, and 21).

UNDESA, 2015. Resolution adopted by the general assembly on 25 september 2015. URL https://documents-dds-ny.un.org/doc/UNDOC/GEN/N15/291/89/PDF/N1529189.pdf?OpenElement (Cited on pages 8 and 9).

Vallada, E., Belenguer, J. M., Villa, F. Alvarez-Valdes, R., 2023. Models and algorithms for a yard crane scheduling problem in container ports. European Journal of Operational Research 309 (2), 910–924. (Cited on pages 23, 34, 35, 37, 39, 41, 52, 54, 59, 63, 65, 75, 79, 83, 91, 101, 104, 105, 109, 110, 114, 118, 140, 141, and 146).

Vidović, M., Radivojević, G. Raković, B., 2011. Vehicle routing in containers pickup up and delivery processes. Procedia-Social and Behavioral Sciences 20, 335–343. (Cited on page 19).

Vis, I. F. Carlo, H. J., 2010. Sequencing two cooperating automated stacking cranes in a container terminal. Transportation Science 44 (2), 169–182. (Cited on pages 34, 35, and 36).

Vis, I. F. Roodbergen, K. J., 2009. Scheduling of container storage and retrieval. Operations Research 57 (2), 456–467. (Cited on pages 20, 23, 30, 34, 35, 37, 40, 41, and 58).

Wang, H., Villa, F., Vallada, E. Ruiz, R., 2025. Solving the yard crane scheduling problem with dynamic assignment of input/output points. Computers & Operations Research 173, 106853. (Cited on pages 54 and 123).

Wiese, J., Suhl, L. Kliewer, N., 2010. Mathematical models and solution methods for optimal container terminal yard layouts. OR Spectrum 32 (3), 427–452. (Cited on page 12).

Wu, L. Ma, W., 2017. Quay crane scheduling with draft and trim constraints. Transportation Research Part E: Logistics and Transportation Review 97, 38–68. (Cited on page 32).

Wu, Y., Li, W., Petering, M. E. H., Goh, M. Souza, R. d., 2015. Scheduling multiple yard cranes with crane interference and safety distance requirement. Transportation Science 49 (4), 990–1005. (Cited on pages 34 and 35).

Yang, C. H., Choi, Y. S. Ha, T. Y., 2004. Simulation-based performance evaluation of transport vehicles at automated container terminals. OR spectrum 26 (2), 149–170. (Cited on page 35).

Yang, X.-M. Jiang, X.-J., 2020. Yard crane scheduling in the ground trolley-based automated container terminal. Asia-Pacific Journal of Operational Research 37 (2), 2050007. (Cited on pages 34, 35, 40, and 47).

Yang, Z., Wang, G. Chu, F., 2013. An effective grasp and tabu search for the 0–1 quadratic knapsack problem. Computers & Operations Research 40 (5), 1176–1185. (Cited on page 94).

Yepes-Borrero, J. C., Perea, F., Villa, F. Vallada, E., 2023. Flowshop with additional resources during setups: Mathematical models and a grasp algorithm. Computers & Operations Research 154, 106192. (Cited on page 94).

Yepes-Borrero, J. C., Villa, F., Perea, F. Caballero-Villalobos, J. P., 2020. Grasp algorithm for the unrelated parallel machine scheduling problem with setup times and additional resources. Expert Systems with Applications 141, 112959. (Cited on page 94).

Zandieh, M. Adibi, M., 2010. Dynamic job shop scheduling using variable neighbourhood search. International Journal of Production Research 48 (8), 2449–2458. (Cited on page 125).

Zeng, Q., Yang, Z. Hu, X., 2011. Disruption recovery model for berth and quay crane scheduling in container terminals. Engineering Optimization 43 (9), 967–983. (Cited on page 32).

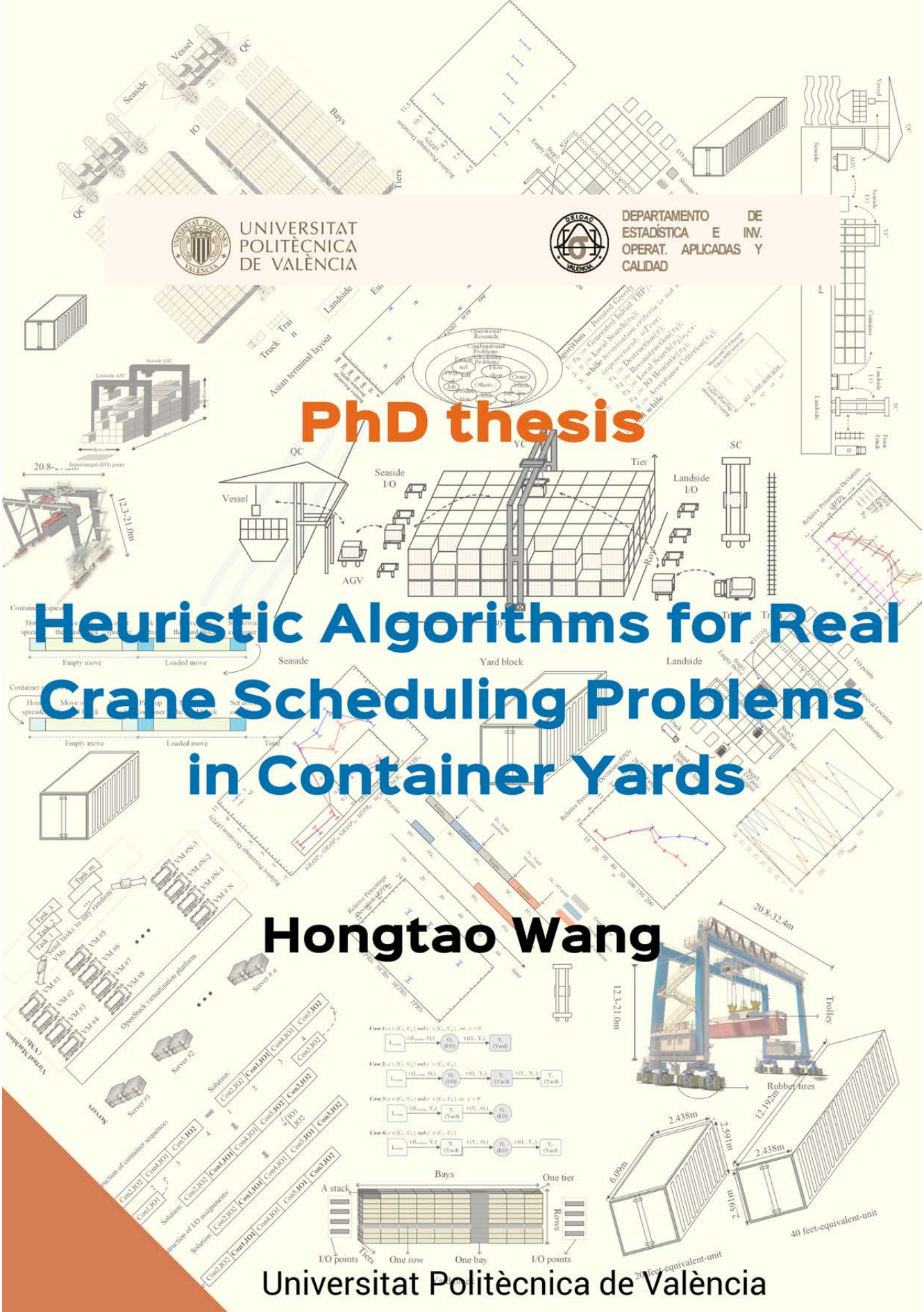
Zhang, Z.-Q., Xu, Y.-X., Qian, B., Hu, R. *et al.*, 2024. An enhanced estimation of distribution algorithm with problem-specific knowledge for distributed no-wait flowshop group scheduling problems. Swarm and Evolutionary Computation 87, 101559. (Cited on page 126).

Zhen, L., Lee, L. H., Chew, E. P., Chang, D.-F. Xu, Z.-X., 2011. A comparative study on two types of automated container terminal systems. IEEE Transactions on Automation Science and Engineering 9 (1), 56–69. (Cited on page 35).

Zheng, F., Man, X., Chu, F., Liu, M. Chu, C., 2018. Two yard crane scheduling with dynamic processing time and interference. IEEE Transactions on Intelligent Transportation Systems 19 (12), 3775–3784. (Cited on pages 33, 34, 35, 36, and 40).

Zheng, F., Man, X., Chu, F., Liu, M. Chu, C., 2019. A two-stage stochastic programming for single yard crane scheduling with uncertain release times of retrieval tasks. International Journal of Production Research 57 (13), 4132–4147. (Cited on pages 34, 35, 37, 38, and 39).

Zhou, C., Lee, B. K. Li, H., 2020. Integrated optimization on yard crane scheduling and vehicle positioning at container yards. Transportation Research Part E: Logistics and Transportation Review 138, 101966. (Cited on pages 30, 34, 35, 36, and 47).



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE ESTADÍSTICA E INV. OPERAT. APLICADAS Y CALIDAD

PhD thesis

Heuristic Algorithms for Real Crane Scheduling Problems in Container Yards

Hongtao Wang

Universitat Politècnica de València