August 2020

Sequencing Multiple-Spreader Crane Operations: Mathematical Formulations and Heuristic Algorithms

Shabnam Lashkari
University of Wisconsin-Milwaukee

SEQUENCING MULTIPLE-SPREADER CRANE OPERATIONS: MATHEMATICAL FORMULATIONS AND HEURISTIC ALGORITHMS

by

Shabnam Lashkari

A Dissertation Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

in Engineering

at

The University of Wisconsin-Milwaukee

August 2020

ABSTRACT

SEQUENCING MULTIPLE-SPREADER CRANE OPERATIONS: MATHEMATICAL FORMULATIONS AND HEURISTIC ALGORITHMS

by

Shabnam Lashkari

The University of Wisconsin-Milwaukee, 2020
Under the Supervision of Professor Matthew E.H. Petering

Maritime container shipping is one the oldest industries and plays a key role in transporting freight all around the world. The International Maritime Organization (IMO) reports that more than 90% of international trade across the world is carried by sea. This method of transportation is by far the most cost-efficient among rail, road, air, and water transportation.

Today most overseas shipping of finished consumer goods is done via 20-, 40-, or 45-foot long steel containers aboard deep-sea container vessels. Every day, tens of thousands of containers are moved between different countries all around the world. In addition, the amount of meat, fish, fruit, vegetables, and general foodstuffs shipped in refrigerated containers continues to increase. As the volume of freight shipped via steel shipping containers grows, it is becoming increasingly important to improve the operational efficiency of the port facilities where containerships are unloaded and loaded.

In this research, we consider several new mathematical problems inspired by the unloading of a containership. These problems are inspired by the recent development of a new kind of quay crane—a multi-spreader quay crane—that can lift more than one 40-foot container from a containership at the same time. This new crane has an extra strong steel structure that allows

heavier lifts to be performed. In contrast to traditional cranes, this new crane may deploy two or three *spreaders* simultaneously.

Multi-spreader quay cranes have the potential to significantly increase the productivity of seaport container terminals. However, due to a paucity of scheduling approaches for such cranes, this potential has not been fully realized. This motivates our research. In this dissertation, we define new mathematical problems that are inspired by the scheduling of double-spreader and triple-spreader quay cranes. These problems are called the *dual-spreader crane* and *triple-spreader crane scheduling problem* respectively.

We formulate the above problems as integer linear programs and develop fast methods for computing lower bounds on the optimal objective value in each case. In addition, we devise simulated annealing, genetic algorithm, and dynamic programming methods to produce high quality solutions for small, medium, large, and very large problem instances in a short amount of time. Experimental results show the effectiveness of our proposed methods in attacking these important logistics problems.

Chapter 1 starts with introducing container shipping history and how it has developed through the years. We then discuss how modern container shipping has dominated world trade and review some statistics to show how this industry affects the global transportation system. Finally, we discuss related academic and industrial literature.

In Chapter 2, we investigate the problem of scheduling a dual-spreader crane that can perform single container lifts and dual container lifts (in which the crane lifts two adjacent containers). This chapter presents a mathematical model of the dual-spreader crane scheduling problem (DSCSP) and describes a fast method for computing a lower bound on the optimal objective value. Then, we introduce a simulated annealing heuristic method that tries to find good

solutions to instances of the DSCSP within a short time. Finally, we describe the experimental setup and discuss the experimental results for two solution methods—standard integer programming and the simulated annealing—on a set of 120 problem instances.

Chapter 3 discusses the triple-spreader crane scheduling problem (TSCSP). A triple-spreader crane can operate in three modes: single, double, and triple. When in (single, double, triple) spreader mode, the crane can lift (1, 2, 3) adjacent containers respectively. The TSCSP is formulated as an integer linear program. Later in the chapter, a method for calculating a lower bound on the optimal objective value is introduced, a genetic algorithm that uses two different gene generating subroutines is explained in detail, and the experimental setup and the experimental results for a set of 120 problem instances are discussed.

Finally, Chapter 4 discusses final conclusions and future work.

To my loving parents and sister,
who made all of this possible,
for all of their support and encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

BASLDSL	binary array showing legal dual-spreader lifts
DP	dynamic programming
DSCSP	dual-spreader crane scheduling problem
GA	genetic algorithm
LB	lower bound
QC	quay cane
SA	simulated annealing
TSCSP	triple-spreader crane scheduling problem

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my amazing advisor, Dr. Matthew Petering, my great mentor, who recognized my potential and gave me the biggest opportunity of my life to study and do research in the field of operations research at the University of Wisconsin-Milwaukee. He has constantly helped me recognize my skills and talent and improve my weak points. He challenged me repeatedly over the course of my research and guided me through these challenges. I have always appreciated his knowledge on the subject of this study and how much he helped me explore its different aspects.

A tremendous thanks to Dr. Christine Cheng for offering her thoughtful insights and providing suggestions to improve the methodology of this study's experiments. Dr. Cheng's input increased the experiments' efficiency and improved their results. She proposed a new method to generate results, discussed the details of this new method, and helped me implement those suggestions.

I would like to express my sincere gratitude to my committee members, Dr. Hamid Seifoddini, Dr. Jaejin Jang, and Dr. Xiaohang Yue for generously offering their time, insights, support, and guidance through the preparation of this study.

I want to thank the UWM Industrial & Manufacturing Engineering (IME) Department for providing me with resources and supporting me through my studies and research. I am very grateful to be a part of the IME family and to play a small role in the development of this department. I have grown personally and professionally during my time in IME through teaching, research, networking, and studying.

I also want to thank my dear friends who supported me endlessly and provided me with valuable intuition and advice.

Chapter 1:

Introduction

1.1 The world before container shipping

For centuries, mankind sailed across the world and moved goods from one place to another. While exploring the world, they collected and shipped food, cotton, treasures and goods from lands with abundant resources and brought them back to their own countries. Although the container shipping industry belongs to the modern world, seaborn shipping and freight transportation existed for millenia but was entirely different back then. With the invention and development of new types of ships, shipping became more feasible and accessible across the world (North 1968).

In the early 1950s, when container shipping was not yet developed, the world's biggest commercial centers had docks at their hearts. The factories' warehouses were located in close proximity to the wharves for an easier raw materials delivery and to ship the final products faster (Levinson 2016). The freight was carried piece by piece on trucks or railcars and transported to the waterfront. Items of different shapes and sizes had to be unloaded separately, documented, and stored in a transit shed. Later, these mixed items, known as break bulk cargo, were moved to the dock and prepared for loading onto the ship. On the dock, workers assembled different boxes and barrels into a draft, and a driver lifted and boarded the draft using a shipboard crane. Another group of workers unhatched the boarded items, moved them to a secure place, and stored them (King 1936). The entire process of loading was tedious, time-consuming, and labor-intensive with no standardization in place.

The unloading process was not much different. Arriving ships were carrying mixed cargo, from bags of sugar to steel coils. Depending on the product type, the unloading methods were different, ranging from using winches to carrying items on the backs of workers. Even though some machinary had arrived to facilitate this process, manpower was still needed throughout the process and human injury was an inevitable part of it (University Press of Liverpool 1954). Moreover, the unloading process, just as much as the loading process, was susceptible to other risks, such as delays, pilfering, damage, loss, and blockages within ports. The mixed nature of the cargo made it challenging to prevent damage to the goods.

Into the 1950s, break bulk shipping was used to transport and ship goods over long distances. When using this method, goods were transported loose or packaged in bags, crates, casks, barrels, or other small containers that varied in terms of the material and size (Kite-Powell 2001). This method had very high labor cost; it had been estimated that the portside costs and cargo handling expenses were 37% to 75% of the total cost of transporting cargo (Levinson 2016). Due to the complexity of break bulk shipping and lack of standardization, the waiting time for ships was also extremely high and cargo ships usually were spending as much time in the port being loaded and unloaded as they did sailing the oceans (Cudahy 2006 and Talley, 2000), while dock workers had to manhandle most of the cargo into and out of tight spaces below decks.

Transportation and shipping a single type of good, such as oil, was cheaper than regular break bulk shipping, due to specialized ships and port facilities for specific products. In the maritime freight market, large oil tankers and dry-bulk carriers started operating more in synergy and attempted to use modern cargo loading and unloading facilities to operationalize the shipping processes in a larger scale. This specially designed bulk shipping had become more and more

industrialized, in contrast to break bulk shipping of more diverse goods, where the process of loading and unloading remained unchanged for decades (Broeze 2002).

The high costs of ocean shipping were a major obstacle for world trade. In 1961, ocean freight costs accounted for 10% and 12% of the value of U.S. imports and exports respectively. These costs made the international trade of some goods impractical. In 1960, the international trade proportion of the U.S. economy was smaller than in 1950, or even in the Depression era of 1930 (Levinson 2016).

In an attempt to overcome these challenges, prior to World War II, US, British, and French railway companies developed new methods for sealing goods in different shapes and boxes before transportation. During World War II, the U.S. military started using metal shipping containers to transport equipment to different sites. In 1947, the U.S. Transportation Corps developed the Transporter which was a rigid, corrugated steel container with a 9,000 lb. capacity. During the Korean War, the Transporter was used for moving equipment (Van Ham and Rijsenbrij, 2012). However, the lack of specialized equipment for loading and unloading and other social challenges, such as resistance to changes in work practices shown by unions, delayed the development of container shipping until the mid-1950s (Bernhofen et al. 2016).

While military efforts were slow in developing a more efficient shipping method, commercial attempts had far greater impact. Shipping companies within the U.S., particularly those led by a former trucking company founder, Malcolm McLean, applied a rather simple idea. In 1955, McLean recognized the inefficacy of the transportation industry, specifically the process of loading and unloading cargo on and off ships. Being an entrepreneur on the lookout for revolutionary ideas, McLean decided to move his business from a trucking company to a company

that transported goods by water. He purchased a steamship company and established the modern shipping container concept.

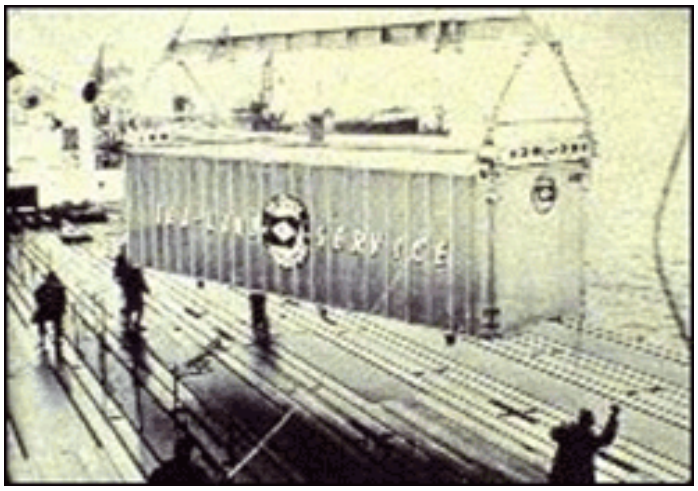

Figure 1.1. Early containers (World Shipping Council 2020a)

McLean proposed to use metal shipping containers (Figure 1.1), similar to the ones that were used by the U.S. military, in larger sizes and yet transportable by trucks or trains. Using this new idea, the loading process could take place in two locations: one location close to the manufacturer, where individual items were loaded into containers, and another at the dockside, where the containers were loaded onto ships. Unloading was similar: goods were removed from containers at the point of distribution or even sale, far removed from the docks (Levinson 2016). McLean's companies and another firm called Matson Navigation Company successfully utilized this idea in different shipping routes in the 1950s (Van Ham and Rijsenbrij, 2012). McLean and his colleagues started a revolution in transportation and world trade.

1.2 The birth of container shipping

The need for shipping standardization arose from the fact that prior to container shipping, the process of transporting goods across continents was complicated, slow, labor-intensive and generally inefficient. Ships spent more time at the dock than sailing. Furthermore, problems such as theft of goods, slow cargo transfer from the ships to the trains, and a general lack of standardized processes to load and unload cargo resulted in logistics delays (Shipping Container History).

In 1956, McLean shipped his first container ship from port Newark to Houston. This ship, carrying 58 containers as well as 15,000 tons of bulk petroleum, took 6 days to arrive at its destination. This new technology completely revolutionized the transportation industry. The containers could be stacked on top of each other, moved directly from top of the trucks or trains onto the ship deck and vice versa without the need to be unpackaged, and it protected the cargo both from being stolen and damaged (World Shipping Council, 2020a).

In 1959, the first quayside container crane called "Portainer" was employed to load containers on ships. This new piece of equipment improved loading time significantly and cut down costs, damage risks, and filching (Levine 2019).

1.3 Globalization of container shipping

In 1966 and 1967, the first transatlantic and transpacific container shipping services were launched. In 1966 the first international containership left Port Elizabeth, New Jersey for Rotterdam in The Netherlands, carrying 236 containers on board. This was indeed the beginning of a significant economic growth in many countries. The world of trade became much more connected than before as shipping goods became faster, less costly, and more secure. Ships

transported goods from Asia to Europe and America, making many stops on the way, delivering containers and loading more to deliver at the next ports in the shipping route (Levine 2019).

Modern container shipping has been around for 64 years. Nowadays, container ships transport more than 60% of the value of sea-transported goods. As the global demand for sea transportation grows, the size of vessels grows larger (UNCTAD Stat 2020b).

The United Nations Conference on Trade and Development (UNCTAD) published a report in November 2019 in the *Review of Maritime Transport 2019* showing the carrying capacity of the global merchant fleet from 1980 to 2019. According to this report, the carrying capacity of the global merchant fleet reached almost 2 billion deadweight tons (dwt: a measure of how much weight a ship can carry) in 2019. Container shipping is responsible for 13.3% of this carrying capacity in 2019 (266 million dwt). Figure 1.2 shows the containership trade capacity in seaborne trade from 1980 to 2019.

Another way to measure containership capacity is by volume. In this regard, the standard way to measure containership volume is in TEUs (twenty-Foot equivalent units). One TEU is equivalent to the volume contained in one 20-Foot-long container that measures 20′ long, 8′ wide, and 8.5′ high. Two other popular containers are 40-Foot-long and 45-Foot-long containers. One 40-Foot-long container counts as two TEUs, and one 45-Foot-long container counts as 2.25 TEUs. It is worth mentioning that in 2019 global container shipping volume reached over 800 million twenty-foot equivalent units (TEUs), which represents a 29% growth since 2012 (622 million TEUs).

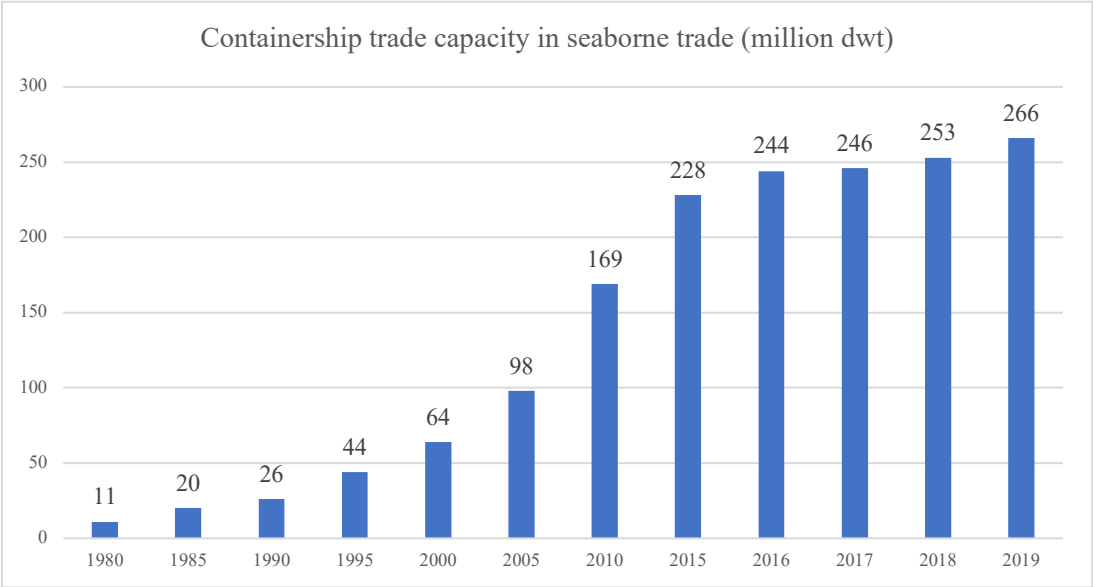

Figure 1.2. Containership trade capacity in seaborne trade (Statista).

Figure 1.3 shows the growth of world global container port traffic from 2010 to 2018 as reported by UNCTAD. Global container port traffic in 2018 is 793 million TEUs, which is a 42% growth since 2010.

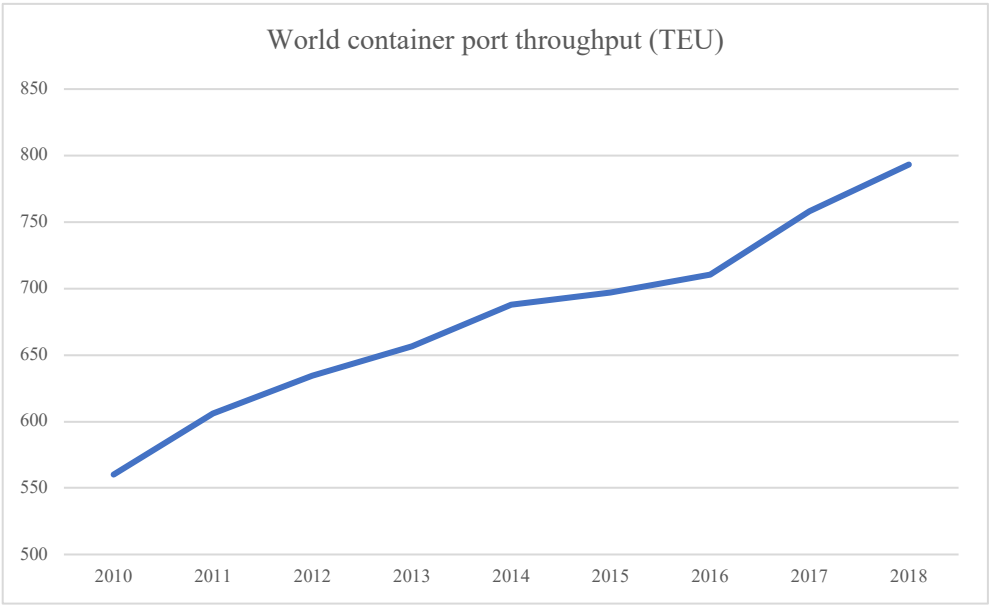

Figure 1.3. Annual world container port throughput (UNCTAD Stat 2020b).

Figure 1.4 shows the five countries that have been best connected to the global liner shipping network during the past thirteen years (UNCTAD Stat 2020a). China and the Republic of Korea have improved significantly in the past ten years compared to their competitors. According to Figure 1.2−1.4, the container shipping industry continues to grow. In a recent video report, *How a Steel Box Changed the World: A Brief History of Shipping*, the cost of shipping an average TV from China to the U.S. is only about $2 (Di Fonzo and Costas Paris, 2018). This is a small example of why container shipping is by far the most cost-effective form of freight transportation.

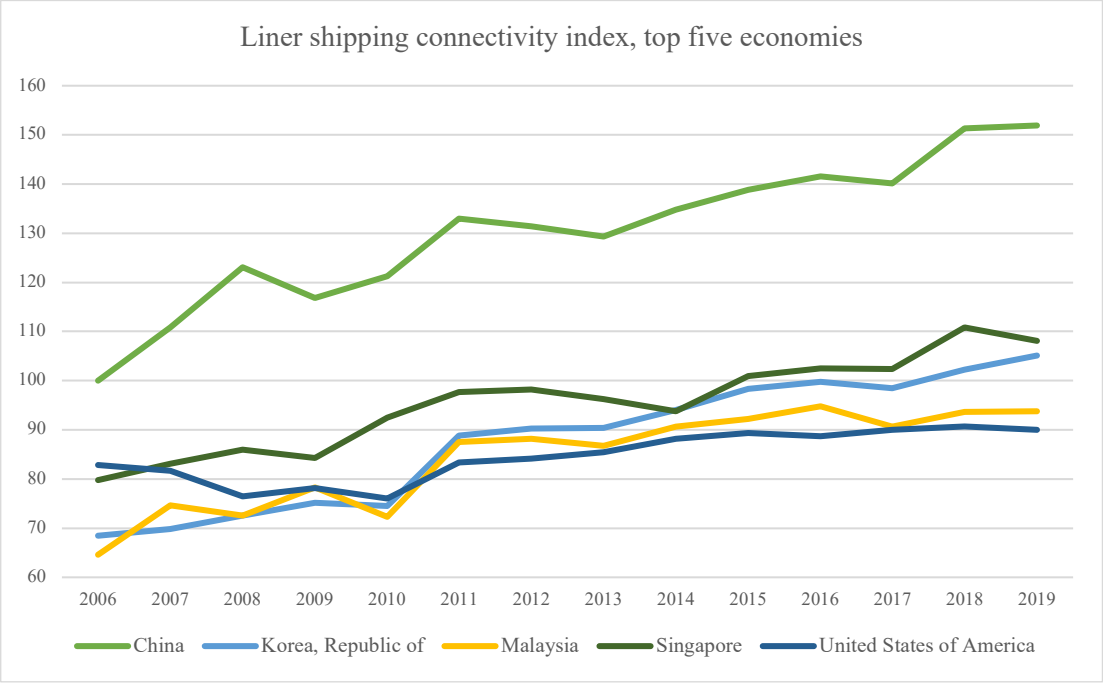

Figure 1.4. Liner shipping connectivity index (UNCTAD STAT).

1.4 Quay cranes in container terminals

Loading and unloading of container ships takes place in container terminals (ports). These facilities are magnificent lots consisting of different functionalities such as container handling,

terminal management, yard planning, traffic planning, etc., with many different kinds of equipment, including quay cranes. The quay crane (QC) is the device that transfers containers between ships and the shore. In a container terminal, after a ship docks next to one or multiple QCs, they start unloading containers based on a predetermined plan. The next step is to store the containers on the storage yard near the QCs, as shown in Figure 1.5. Later these containers are staged to either be loaded on trucks or trains to be transported on land (imported) or to be loaded onto another ship to be transported via sea (exported).

Figure 1.5. A container port (Colorado Springs Business Journal).

Much research has been done in the recent decades to improve the efficiency and productivity of these facilities. This dissertation is mainly focused on unloading containers from the tops of container ships using multiple-spreader (i.e. tandem lift) quay cranes. Multi-spreader QCs can be equipped with one, two, or three spreaders (i.e. grappling devices) simultaneously, with each spreader capable of lifting one 40-foot or two 20-foot containers. Figure 1.6 shows

containers being lifted by one, two, and three spreaders respectively. It only takes a few minutes for a multi-spreader QC to change the number of spreaders that it uses.

In this study, we investigate the problem of scheduling a multiple-spreader QC to unload containers from the top of a container ship. We define two new optimization problems, propose new mathematical models, and develop new heuristic algorithms to handle large instances of these problems.

Figure 1.6. Single, double, and triple spreader handling of containers by QCs.

1.5 Literature review

The literature relevant to this research includes all published works in academic- and industry-focused journals that discuss industrial crane systems. A thorough search of this literature considered every item with a title containing the phrase “crane,” “spreader,” or “block relocation” that was published by six academic publishers: Elsevier, Springer, INFORMS, Taylor & Francis, Wiley, and Palgrave Macmillan. Industry journals were also searched. The results of this literature review yielded several hundred articles, most of which concern the management of operations at seaport container transshipment terminals. No article with a focus on a non-seaport-related crane system was deemed relevant to this research.

Ten articles surveying the literature on seaport container terminal operations were identified, including the works by Vis and De Koster (2003), Steenken, Voß, and Stahlbock (2004),

Stahlbock and Voß (2008), Bierwirth and Meisel (2010), Angeloudis and Bell (2011), Carlo, Vis, and Roodbergen (2014a, 2014b, 2015), Gharehgozli, Roy, and de Koster (2015), and Bierwirth and Meisel (2015). Several of these articles mention multi-spreader quay cranes (QCs) as an important new technology for container terminals. However, no article discusses a published paper that proposes a method for scheduling multi-spreader cranes.

Various methods have been developed for scheduling single-spreader QCs and yard cranes (YCs) at seaport container terminals. For example, Kim and Kim (1999) develop a math model and exact solution method for routing a single yard crane (YC) at a seaport container terminal. Wu, Li, Petering, Goh, and de Souza (2015) present methods for scheduling multiple YCs that prevent YC interference and consider safety distance requirements. Regarding QCs, Imai, Chen, Nishimura, and Papadimitriou (2008) introduce a math model of the simultaneous berth and QC allocation problem and develop a genetic algorithm to find near-optimal solutions to the problem. Meisel and Bierwirth (2013) develop methods for solving the integrated berth allocation, QC allocation, and QC scheduling problem at seaport container terminals. Chen, Lee, and Cao (2011) develop methods for scheduling QCs at indented berths. Kim and Park (2004) introduce a math model for scheduling QCs at a regular berth and develop exact and heuristic methods for solving problem instances. Moccia, Cordeau, Gaudioso, and Laporte (2006), Ng and Mak (2006), and Unsal and Oguz (2013) also propose various QC scheduling methods. Tang, Zhao, and Liu (2014) consider a joint QC and truck scheduling problem.

Discussions of multi-spreader (i.e. tandem-lift) QCs are uncommon in the literature. Chao and Lin (2011) present a methodology that trades off the various features of advanced QCs (including multi spreader QCs) in order to choose a suitable advanced QC for any given container terminal. Xing, Yin, Quadrifoglio, and Wang (2012) and Chen, Cao, and Zhao (2014) develop

methods for scheduling automated guided vehicles (AGVs) and yard trucks (YTs), respectively, when tandem-lift QCs are used at a container terminal. Choi, Im, and Lee (2014) use a simulation methodology to develop an operating system that can increase the productivity of a container terminal where tandem-lift QCs are used. Several articles in industry journals—including those by McCarthy, Jordan, and Wright (2007) and World Cargo News (2007)—contain general discussions of tandem-lift QCs but do not present results related to the scheduling or productivity of such cranes. On the other hand, Song (2011) discusses the productivity of tandem-lift QCs during real-life experiments conducted at Pusan Newport and proposes methods for conducting double cycling operations using such cranes. Goussiatiner (2007a, 2007b) generates plausible ship stowage configurations in order to compare the productivity of unloading such ships using single spreader, dual-spreader, and triple-spreader QCs. However, no methods for scheduling multi-spreader cranes are proposed.

In Cheng et al. (2020), the authors prove that the multi-spreader crane scheduling problem (MSCSP) is NP-hard when the crane has three or more modes. In particular, the triple-spreader crane scheduling problem (TSCSP) is NP-hard. Cheng et al. (2020) also discuss, but leave open issues related to the computational complexity of the DSCSP (dual-spreader crane scheduling problem) that is investigated by Lashkari et al. (2017).

To our knowledge, Lashkari et al. (2017) is the only published work to propose methods for scheduling a multi-spreader crane. That work presents a mathematical model and simulated-annealing-based heuristic for sequencing the operations of a dual-spreader QC that is supposed to remove all containers from a container bay in minimum time. The authors develop a fast method for computing a lower bound on the optimal objective value and show that their heuristic finds feasible solutions whose objective values, on average, are within 6% of the lower bound across

four problem sizes—small, medium, large, and very large. Chapter 2 of this dissertation is nearly identical to Lashkari et al. (2017).

Chapter 3 of this dissertation extends Lashkari et al. (2017) to the case of a triple-spreader crane. In Chapter 3, we propose a mathematical model and genetic algorithm (GA) for sequencing the operations of a triple spreader QC that should remove all containers from a container bay in minimum time. We also develop a new method for computing a lower bound on the optimal objective value which differs from that presented in Chapter 2. On average, the GA produces solutions to TSCSP instances whose objective values are within 7% of the lower bound. Furthermore, the GA outperforms the simulated-annealing-based heuristic proposed in Chapter 2 on instances of the dual-spreader crane scheduling problem (DSCSP). To our knowledge, Chapter 3 is the first study to consider the scheduling of a crane that can operate with three different numbers of spreaders.

Chapter 2:
Dual-Spreader Crane Scheduling Problem

2.1 Problem description

We define the dual-spreader crane scheduling problem (DSCSP) as follows. Consider a set of identically sized containers (blocks, items) that are temporarily stored as inventory (e.g. on the deck of a ship). Due to space limitations, these containers are stacked directly on top of each other in a storage bay consisting of S stacks and T tiers. At time 0, there are E_s containers in stack s. The weight of the container in stack s, tier t is given by W_{st}. Consider the problem of sequencing the lifts made by one crane that will remove all containers from the bay. This crane can operate in two modes: single-spreader or dual-spreader mode. When in single-spreader mode, the crane may remove any single container from the top of any stack. This type of lift takes H_1 minutes. When in dual-spreader mode, the crane may simultaneously remove any two containers in the same tier from the top of any two adjacent stacks as long as the sum of their weights does not exceed w_{Limit}. This type of lift takes H_2 minutes. Furthermore, the changeover (i.e. setup) time between modes is C minutes. The crane can begin in either mode at time 0 with no initial setup cost. The goal is to sequence the individual lifts and changeovers of the crane to minimize the total time needed to remove all containers from the bay. To make the problem meaningful, we assume that $H_1 < H_2 < 2H_1$ and $\max\{W_{st}\} < w_{Limit} < 2 * \max\{W_{st}\}$.

Figure 2.1 shows an instance of the DSCSP. In this instance, $S = 8$, $T = 3$, $E_s = 3$ for all s, and the weights W_{st} of all containers in the bay are shown in the upper-left corner of the figure. In addition, we assume that $w_{Limit} = 10$, $H_1 = 1.5$, $H_2 = 1.8$, and $C = 2.1$. Note that, even for this small instance, it is not easy to decide which containers should be lifted in single-spreader mode

and which containers should be lifted in dual-spreader mode. Figure 2.1 shows a feasible crane lift sequence for this instance. This sequence consists of five dual-spreader lifts followed by four single-spreader lifts followed by five dual-spreader lifts. Two changeovers between spreader modes are required, so the total time needed to empty the bay—the *makespan*—is $10\times1.8+4\times1.5+2\times2.1=28.2$ minutes. We later show that this is not the optimal makespan for this instance.

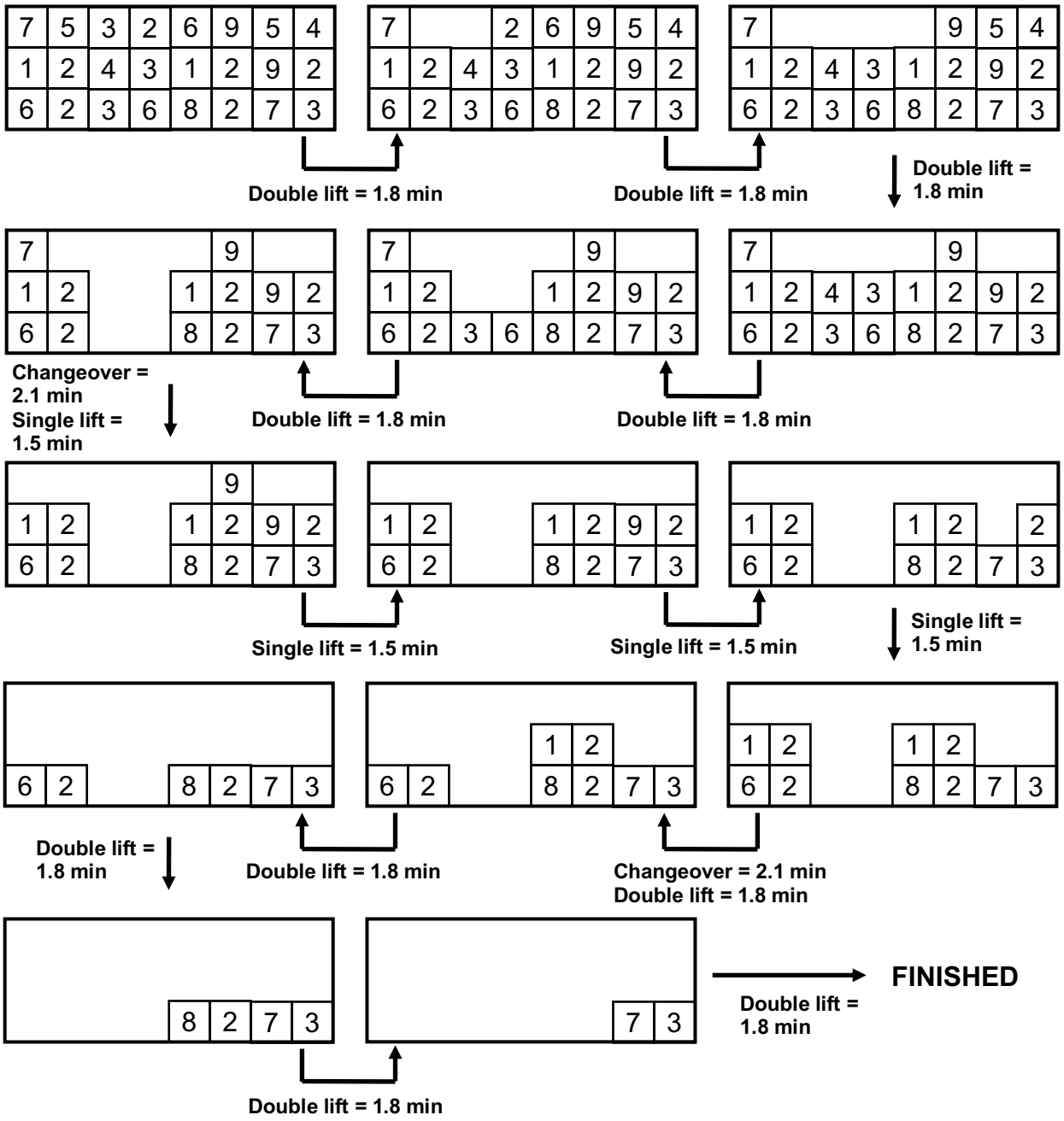

Figure 2.1. Feasible crane lift sequence with makespan 28.2 minutes for a problem instance of size 3 × 8 with $w_{\text{Limit}} = 10$.

2.2 Mathematical model

To facilitate the model development, we first convert an instance of the DSCSP into a "binary array showing legal dual-spreader lifts" (BASLDSL). Figure 2.2 depicts the conversion of the instance in Figure 2.1 to BASLDSL, where binary variables are used to indicate whether a dual-spreader lift could be performed on a pair of adjacent containers in the same tier. Without loss of generality, we use the left side of the pair to denote whether a "legal" dual-spreader lift can be performed within the given weight limit w_{Limit}. For example, the top-left '0' in BASLDSL indicates that the first and the second containers (from the left side) in the top tier cannot be dual-spreader lifted because their combined weight—12—exceeds w_{limit} = 10. Also, the '1' adjacent to the top-left '0' indicates that the second and the third containers (from the left) in the top tier can be dual-spreader lifted because their combined weight—8—does not exceed w_{Limit}. In the original problem instance, we number the tiers 1, ..., *T* from bottom to top and the stacks 1, ..., *S* from left to right. In BASLDSL, we use the terms *tier* (1, ..., *T* from bottom to top) and *column* (1, ..., *S*−1 from left to right) to refer to various locations.

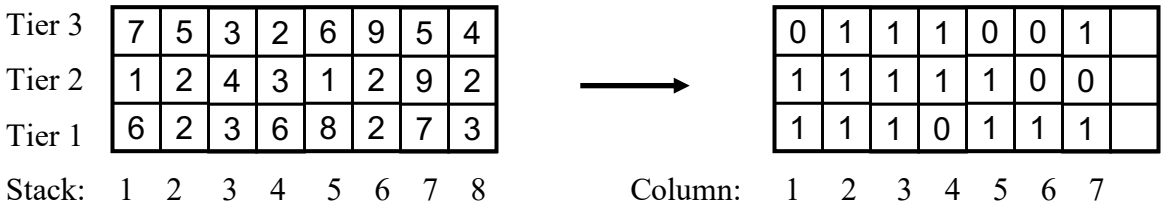

Figure 2.2. Conversion of problem instance (left) into a binary array showing legal dual spreader lifts (BASLDSL) (right), assuming w_{Limit} = 10.

2.2.1 Mathematical formulation of the DSCSP

Our mathematical model of the DSCSP, model DSCSP, discretizes time into intervals. During each time interval, at most one (single-spreader or dual-spreader) lift may occur. The duration of an interval is therefore either H_1or H_2 minutes depending on the type of operation performed. Between two consecutive intervals, at most one spreader changeover may occur (Figure 2.1).

The indices in model DSCSP are as follows:

s Stack (i.e. column) (s = 1, 2, …, S).

t Tier (t = 1, 2, …, T).

i Time interval (i = 1, 2, …, I, I+1).

The input parameters in model DSCSP are as follows:

S Number of stacks in the storage bay. $S \geq 2$ to avoid triviality.

T Number of tiers in the storage bay.

I Number of time intervals available (= $S \times T$ to be conservative).

E_s Initial number of containers in stack s (integer, ≥ 0) (s = 1, 2, ..., S).

C Changeover time between single- and dual-spreader deployment (minutes).

H_1 Handling time per lift using single spreader (minutes).

H_2 Handling time per lift using dual spreader (minutes).

L_{st} = 1 if the left side of the dual spreader can be used at stack s, tier t in the original configuration (binary) (s = 1, 2, …, S-1; t = 1, 2, …, T). This parameter equals the value of the item in column s, tier t of BASLDSL.

The decision variables in model DSCSP are as follows:

X_{si} = 1 if a single-spreader lift is performed at the top of stack s during time interval i (binary) (s = 1, 2, …, S; i = 1, 2, …, I).

Y_{si} = 1 if a dual-spreader lift is performed in which the left (right) spreader lifts the container that is on the top of stack s (s+1) during time interval i (binary) (s = 1, 2, …, S-1; i = 1, 2, …, I).

G_i = 1 if a spreader changeover is made between time intervals i−1 and i (binary) (i = 2, …, I).

F_i = 1 if all containers have been removed from the bay by the beginning of time interval i (binary) (i = 1, 2, …, I+1).

N_{si} Number of containers in stack s at the beginning of time interval i (integer, ≥ 0) (s = 1, …, S; i = 1, 2, …, I+1).

R_{ti} = 1 if containers are allowed to be removed from tier t during time interval i (binary) (t = 1, 2, …, T; i = 1, 2, …, I).

Objective function:

$$Minimize \quad \sum_{i=1}^{I}\left(\sum_{s=1}^{S} H_1 X_{si} + \sum_{s=1}^{S-1} H_2 Y_{si}\right) + \quad \sum_{i=2}^{I} G_i C \tag{1}$$

Subject to:

$$\sum_{s=1}^{S} X_{si} + \sum_{s=1}^{S-1} Y_{si} + F_i = 1 \qquad i = 1, 2, \ldots, I \tag{2}$$

$$\sum_{i=1}^{I+1} F_i \geq 1 \tag{3}$$

$$\left.\begin{aligned}
&\sum_{s=1}^{S} X_{s,i-1} + G_i - 1 \le \sum_{s=1}^{S-1} Y_{si} \\
&\sum_{s=1}^{S-1} Y_{si} + \sum_{s=1}^{S} X_{s,i-1} - 1 \le G_i \\
&G_i + \sum_{s=1}^{S-1} Y_{si} - 1 \le \sum_{s=1}^{S} X_{s,i-1}
\end{aligned}\right\} \qquad i = 2, 3, \ldots, I \tag{4}$$

$$\left.\begin{aligned}
&\sum_{s=1}^{S-1} Y_{s,i-1} + G_i - 1 \le \sum_{s=1}^{S} X_{si} \\
&\sum_{s=1}^{S} X_{si} + \sum_{s=1}^{S-1} Y_{s,i-1} - 1 \le G_i \\
&G_i + \sum_{s=1}^{S} X_{si} - 1 \le \sum_{s=1}^{S-1} Y_{s,i-1}
\end{aligned}\right\} \qquad i = 2, 3, \ldots, I \tag{5}$$

$$N_{si} \le T(1 - F_i) \qquad i = 1, 2, \ldots, I+1;\ s = 1, 2, \ldots, S \tag{6}$$

$$N_{s1} = E_s \qquad s = 1, 2, \ldots, S \tag{7}$$

$$\begin{aligned}
&N_{1,i+1} = N_{1i} - X_{1i} - Y_{1i} && i = 1, 2, \ldots, I \\
&N_{s,i+1} = N_{si} - X_{si} - Y_{si} - Y_{s-1,i} && i = 1, 2, \ldots, I\ ;\ s = 2,\ldots, S\text{-}1 \\
&N_{S,i+1} = N_{Si} - X_{Si} - Y_{S-1,i} && i = 1, 2, \ldots, I
\end{aligned} \tag{8}$$

$$\begin{aligned}
&X_{1i} + Y_{1i} \le N_{1i} && i = 1, 2, \ldots, I \\
&X_{si} + Y_{si} + Y_{s-1,i} \le N_{si} && i = 1, 2, \ldots, I;\ s = 2,\ldots, S\text{-}1 \\
&X_{Si} + Y_{S-1,i} \le N_{Si} && i = 1, 2, \ldots, I
\end{aligned} \tag{9}$$

$$\sum_{t=1}^{T} R_{ti} = 1 \qquad i = 1, 2, \ldots, I \tag{10}$$

$$-T(1 - X_{si}) \le \left(\sum_{t=1}^{T} tR_{ti}\right) - N_{si} \le T(1 - X_{si}) \qquad i = 1, 2, \ldots, I;\ s = 1, 2, \ldots, S \tag{11}$$

$$\left.\begin{aligned}
&-T(1 - Y_{si}) \le \left(\sum_{t=1}^{T} t * R_{ti}\right) - N_{si} \le T(1 - Y_{si}) && i = 1, 2, \ldots, I;\ s = 1, 2, \ldots, S\text{-}1 \\
&-T(1 - Y_{si}) \le \left(\sum_{t=1}^{T} t * R_{ti}\right) - N_{s+1,i} \le T(1 - Y_{si}) && i = 1, 2, \ldots, I;\ s = 1, 2, \ldots, S\text{-}1
\end{aligned}\right\} \tag{12}$$

$$Y_{si} + R_{ti} - 1 \le L_{st} \qquad i = 1, 2, \ldots, I;\ s = 1, 2, \ldots, S\text{-}1;\ t = 1, 2, \ldots, T \tag{13}$$

The objective function (1) minimizes the makespan, M, which is the sum of the container handling and spreader changeover times. Constraint (2) ensures that at most one lift, either by the single spreader or dual spreader, can be performed during any time interval i; if no lift is made, then the "finished" binary variable F_i should be set to 1. Constraint (3) ensures that the process of removing containers from the bay is finished by the end of the last time interval. Constraint (4) forces a changeover to happen when switching from dual-spreader to single-spreader mode. This constraint has three expressions with the following structure: $A + B - 1 \leq C$; $C + A - 1 \leq B$; and $B + C - 1 \leq A$. These expressions ensure that if any two of the binary terms A, B, and C equal 1, then the third term equals 1. Term A indicates if a dual-spreader lift is made during time interval $i-1$; B indicates if a spreader changeover is made between time intervals $i-1$ and i; and C indicates if a single-spreader lift is made during time interval i. Constraint (5) is the same as (4) except that it considers the switch from single-spreader to dual-spreader mode.

Constraint (6) indicates that all stacks need to be empty before we can set F_i to 1. Constraint (7) initializes the stack heights for the first-time interval. Constraint (8) updates the stack heights based on the lifts made during each time interval. Constraint (9) ensures that no lift is made from an empty stack. Constraint (10) ensures that containers may be removed from only one tier during each time interval. Constraints (11) and (12) enforce the physical limitation that containers can only be picked up from the top tier. Constraints (10) and (12) together ensure that the dual spreader may only lift two containers that are in the same tier. Constraint (13) ensures that dual-spreader lifts do not violate the (weight-limit-respecting) binary values in BASLDSL. The decision variable domains are included in the decision variable descriptions that precede the model.

2.2.2 Lower bound computation

The DSCSP is a challenging optimization problem. Although the NP-hardness of this problem is still an open question, the results in Section 2.4 indicate that an optimization approach based on model DSCSP struggles to find proven optimal solutions to instances of modest size. In such cases, it is good to have a lower bound on the optimal objective value in order to estimate the quality of solutions produced by heuristic methods.

Algorithm 1 shows our approach for computing a lower bound on the optimal objective value for the DSCSP. It starts from the top tier and works downwards iteratively tier-by-tier (line 2). The consideration of each tier starts at its left end (line 3). Adjacent containers are checked in pairs to see whether they can be dual-spreader lifted (line 3). If so, the number of dual-spreader lifts is increased by one and these two containers are marked as handled (lines 5 and 6); otherwise, the number of single-spreader lifts is increased by one and only the left container of the pair is marked as handled (lines 7 and 8). The counting continues from left to right in the current tier until all containers in the tier are marked as handled, i.e. until the algorithm reaches the right side of the tier (line 4). Then the next tier is considered (line 2).

Once all containers in the storage bay are marked as handled, the total number of dual-spreader lifts is checked. If this number is not large enough to justify the cost of one spreader changeover, the algorithm assumes that only single-spreader lifts are performed, and no changeovers are made (line 10). Otherwise it assumes that one changeover and the counted number of single-spreader and dual-spreader lifts are performed (line 12).

Algorithm 1: Lower bound computation.

1 Set single-spreader and dual-spreader lift counters $N_s = 0$, $N_d = 0$;
2 **for** each tier **do**
3 | Starting from the left side of the tier, check whether the next two containers can be lifted together without violating w_{Limit};
4 | **while** not reaching the right side of the tier **do**
5 | | **if** the next two unhandled containers can be dual-spreader lifted **then**
6 | | | $N_d = N_d + 1$; mark the next two containers as handled;
7 | | **else**
8 | | | $N_s = N_s + 1$; mark the next one container as handled;
9 **if** $N_d H_2 + C > 2N_d H_1$ **then**
10 | $LB = H_1 (N_s + 2N_d)$;
11 **else**
12 | $LB = N_s H_1 + N_d H_2 + C$;
13 Report the lower bound LB;

Theorem 1. Algorithm 1 computes a true lower bound on the optimal objective value for the DSCSP.

Proof. Note that one changeover is included in the lower bound computation if it is profitable; otherwise no changeover is included (lines 9–12). Thus, Algorithm 1 assumes the bare minimum number of changeovers. Consider the value of N_d after the completion of the large "for" loop in Algorithm 1 (lines 2–8). We show that this value equals the maximum total number of dual-spreader lifts (i.e. dual lifts) that can be made. This fact, combined with the stipulation $H_2 < 2H_1$, will prove the theorem.

Each dual lift is confined to a single tier. Thus, it suffices to show that the "greedy method" in Algorithm 1—which accepts all candidate dual lifts as soon as they appear during a left-to-right scan of a given tier—correctly computes the maximum number of dual lifts that can be made in any given tier.

We prove the correctness of the greedy method by induction. Let $D(n)$ be the maximum number of dual lifts that can be made on the right-most n containers in the tier at hand. Clearly, the greedy method correctly computes the maximum number of dual lifts that can be made on a

set of 1 or 2 containers in isolation; thus, it correctly computes $D(1)$ and $D(2)$ when applied to the right-most 1 or 2 containers in the tier at hand.

To complete the proof, we will show that, for $n \geq 3$, if the greedy method correctly computes $D(n-2)$ and $D(n-1)$, then it also correctly computes $D(n)$. There are two cases for the n containers at hand. In case 1, the two left-most containers cannot be feasibly dual lifted, i.e. they violate the weight limit. In this case, the greedy method skips over the left-most container; continues scanning at the second container; and computes $D(n-1)$ as the number of dual lifts made on the n containers. But this is a correct computation for $D(n)$. Indeed, the left-most container is not eligible to participate in any dual lifts, so $D(n) = D(n-1)$ in this case. In case 2, the two left-most containers can be feasibly dual lifted. In this case, the greedy method accepts the lift; skips over the two containers; continues scanning at the third container; and computes $1 + D(n-2)$ as the number of dual lifts made on the n containers. Any non-greedy method, by definition, would reject the dual lift; skip over the left-most container; continue scanning at the second container; and compute at most $D(n-1)$ as the number of dual lifts made on the n containers. However, it is a general rule that $D(n-1) \leq D(n-2) + 1$. Indeed, the addition of one container to the left-hand side of an existing row of $n-2$ containers increases the total number of dual lifts that can be performed on these containers by no more than one. This is because all dual lifts, except possibly the left-most such lift, found in the "optimal" set of dual lifts for the set of $n-1$ containers can be found in the set of $n-2$ containers. Thus, the dual lift tally computed by any non-greedy method cannot be more than that computed by the greedy method. Thus, the greedy method is "optimal" in case 2; it correctly computes $D(n)$ in this case. We have just proven the statement that begins this paragraph. Thus, the greedy method correctly computes $D(n)$ for all $n \geq 1$. ■

2.3 Heuristic approach

Experimental results, discussed in Section 2.4, indicate that a direct math programming approach based upon model DSCSP is not a satisfactory solution method for large DSCSP instances with 50 or more containers. This motivates us to develop a heuristic method that can find good solutions to large problem instances within a reasonable time. We now describe this heuristic method.

Our overall method consists of a constructive heuristic embedded within a simulated annealing (SA) metaheuristic. The constructive heuristic deterministically builds a feasible crane lift sequence based on the values of six parameters. During each iteration of the SA algorithm, the values of one or more parameters are changed to new, neighboring values, and a new feasible crane lift sequence is generated and evaluated.

Among several available metaheuristic approaches−including simulated annealing (SA), genetic algorithm (GA), tabu search (TS), and ant colony optimization (ACO)−we decided to use SA because it offered a good trade-off between ease of use and expected solution quality. SA is among the easiest metaheuristic techniques to implement, yet it usually still achieves good results. In general, SA can be used in highly non-linear problems that have a large number of constraints. SA is an extensible algorithm used to search for global optimality and it is easy to implement. The performance of SA depends on the quality of solution construction (Suman and Kumar, 2006).

We now provide a general description of the constructive heuristic, followed by a detailed description. Then we discuss the SA algorithm. The constructive heuristic is divided into two stages. In stage 1, the type of lift (single-spreader or dual-spreader) for each container is decided. In stage 2, a crane lift sequence—a list of individual lifts and spreader changeovers—is generated based on the output from stage 1.

Our general approach to stage 1 is to iteratively accept or reject the dual-spreader lift opportunities that are shown in BASLDSL (right side of Figure 2.2). Note that the rejection of some dual-lift opportunities is often necessary to guarantee feasibility. For example, at least one of the dual-lift opportunities represented by two adjacent 'ones' in the same tier in BASLDSL must be rejected; otherwise, the same container would be involved in two dual-spreader lifts—one with the container on its left, and one with the container on its right. Importantly, the dual-lift opportunities are not considered individually, but rather in batches of contiguous dual-lifts that are aligned vertically (i.e. in batches of consecutive 'ones' in the same column in BASLDSL). The consideration of such a batch often, but not always, results in all dual-lifts in the batch being accepted. To maintain feasibility, every acceptance is followed by the immediate rejection of all dual lift opportunities in the columns immediately to the right and left of the accepted batch's column in BASLDSL. The output from stage 1 is a modified, or *fixed*, version of BASLDSL in which (1) the respective values are less than or equal to those in the initial BASLDSL and (2) there are no adjacent 'ones' in the same tier.

In stage 2, we use the fixed BASLDSL to label each container in the bay with a "S" ("D") if it will be single-spreader (dual-spreader) lifted. Then we construct a feasible crane lift sequence by iteratively removing containers from the tops of the stacks in the bay if they match the current spreader being deployed. When no more lifts can be made using the current spreader, the spreader is changed. Lifting then continues using the new spreader. This process continues until no more containers remain in the bay. The makespan of the crane lift sequence is then computed.

We now describe the constructive heuristic in greater detail. Table 2.1 lists the six parameters that guide this heuristic. Parameters 1 and 6 have two possible values; Parameter 2 has

T possible values; and Parameters 3, 4, and 5 are long sequences of [0,1) real numbers. Parameters 1–5 are used in stage 1, and Parameter 6 is used in stage 2.

Table 2.1. Parameters that guide the constructive heuristic.

InitialDirection	Equals 0 (1) if heuristic initially moves down from higher to lower tiers (up from lower to higher tiers).
InitialTier	Tier at which heuristic begins (= integer from 1 to *T*).
ColumnChooser[ ]	The next value in this sequence of [0, 1) real numbers decides which column is considered next if two or more columns *c* tie for having the greatest *Depth*(*c*).
FixFullDepthYN[ ]	The next value in this sequence of [0, 1) real numbers is compared to *FullDepthChance* to decide whether or not to fix *Depth*(*c*) dual-spreader lifts in column *c*.
NumLiftsToFix[ ]	If *FixFullDepthYN* indicates that fewer than *Depth*(*c*) dual-spreader lifts in column *c* should be fixed, the next value in this sequence of [0, 1) real numbers decides how many such lifts are fixed.
InitialSpreader	Forms a feasible crane lift sequence from the values in the fixed BASLDSL. Equals "*S*" ("*D*") if the lifting begins in single-spreader (dual-spreader) mode.

Algorithm 2 shows the pseudocode for stage 1 of the constructive heuristic. In this code, *A*[*] indicates the first unused item in array *A*. In stage 1, BASLDSL is *scanned* in order to decide which potential dual-spreader lifts should and should not be performed. During this scan, the initial values in BASLDSL are gradually *fixed* to 1 or 0, where a 1 (0) means that the associated dual-spreader lift will (will not) be performed. Parameter *InitialTier* in Table 2.1 specifies the starting point for this scan, which proceeds tier by tier (line 1). Parameter *InitialDirection* specifies whether the scan initially proceeds down from *InitialTier* to lower tiers or up from *InitialTier* to higher tiers (line 1). We let *BA*[*c*, *t*] denote the value in column *c* and tier *t* in BASLDSL (line 2).

Algorithm 2. Stage 1 of the constructive heuristic.

1 Set *Phase* = 1, *Dir* = *InitialDirection*, and *Tier* = *InitialTier*;
2 Let *BA*[*c*, *t*] denote the value in column *c* and tier *t* in BASLDSL;
3 Set *Fixed*[*c*, *t*] = no for all (*c*, *t*) in BASLDSL;
4 **while** *Tier* ≥ 1 (≤*T*) when *Dir* = 0 (1) **do**
5 Set *Fixed*[*c*, *Tier*] = yes for all *c* such that *BA*[*c*, *Tier*] = 0;
6 **while** *Fixed*[*c, Tier*] = no for any *c* from 1 to *S*−1 **do**
7 For all *c*∈1,..., *S*−1 for which *Fixed*[*c*, *Tier*] = no, let *Depth*(*c*) be the number of consecutive 'ones' that appear in column *c* in BASLDSL beginning with tier *Tier* and moving down (up) if *Dir* = 0 (1). Let *CGD* be the column *c* with the greatest *Depth*(*c*). Break ties using *ColumnChooser*[*];
8 **if** *FixFullDepthYN*[*] ≤ *FullDepthChance* **then**
9 *DualLiftsFixed* = *Depth*(*CGD*);
10 **else**
11 *DualLiftsFixed* = floor(*Depth*(*CGD*)×*NumLiftsToFix*[*]);
12 **if** *DualLiftsFixed* = 0 **then**
13 Set *BA*[*CGD*, *Tier*] = 0 and *Fixed*[*CGD*, *Tier*] = yes;
14 **else**
15 Fix *DualLiftsFixed* 'ones' in column *CGD* in direction *Dir* in BASLDSL. In other words, set *BA*[*CGD*, *t*] = 1 and *Fixed*[*CGD*, *t*] = yes for all *t* from *Tier* to *Tier*-*DualLiftsFixed*+1 if *Dir* = 0 or from *Tier* to *Tier*+*DualLiftsFixed*-1 if *Dir* = 1;
16 Fix *DualLiftsFixed* 'zeros' in the columns adjacent to column *CGD* in direction *Dir* in BASLDSL;
17 Decrease (increase) *Tier* by 1 if *Dir* = 0 (1);
18 **if** *Phase* = 1 and *Dir* = 0 (1) and *Tier* = 0 (*T* + 1) and *InitialTier* < *T* (> 1) **then**
19 Set *Phase* = 2, *Dir* = 1 (0), and *Tier* = *InitialTier*;
20 Set *Fixed*[*c*, *InitialTier*] = no for all *c* from 1 to *S* − 1;
21 Report the BASLDSL with fixed values;

During the consideration of tier *Tier* in BASLDSL, the heuristic identifies the columns *c* for which *BA*[*c*, *Tier*] = 1 and is not already fixed (line 6). The value *Depth*(*c*) is then computed for each such column *c*. *Depth*(*c*) equals the number of consecutive 'ones' in column *c* that begin at tier *Tier* and proceed upwards or downwards depending on the current scanning direction *Dir* (line 7). The dual-spreader opportunities in the column *c* with the greatest *Depth*(*c*) are the first candidates for acceptance. When two or more columns *c* tie for having the greatest *Depth*(*c*), the next [0,1) real number in array *ColumnChooser*[ ] breaks the tie and selects the "column with the greatest depth" (i.e. *CGD*) (line 7). In particular, if *L* columns are tied, then the *n*th such column is selected if and only if (*n* − 1)/*L* ≤ *ColumnChooser*[*] < *n*/*L*. The next [0,1) real number in array

FixFullDepthYN[ ] is then compared to global parameter *FullDepthChance* (line 8) to decide if all *Depth*(*CGD*) dual-spreader lift opportunities in column *CGD* are accepted (line 9) or not (lines 10 and 11). If not, the next [0,1) real number in array *NumLiftsToFix*[ ] indicates what fraction of the dual-spreader lift opportunities are accepted, i.e. to what depth the 'ones' in the column are fixed. In particular, the number of 'ones' that are fixed equals floor(*Depth*(*CGD*)× *NumLiftsToFix*[*]) (line 11). If this equals 0 (line 12), the dual-lift opportunity in column *CGD*, tier *Tier* is rejected and the corresponding cell in BASLDSL is fixed to 0 (line 13). Otherwise, one or more dual-lift opportunities in column *CGD* are accepted (i.e. a sub-column of 'ones' in BASLDSL is fixed) (line 15), and the values in all cells on either side of this sub-column are fixed to 0 (line 16). The latter step ensures that two adjacent 'ones' in the same tier in BASLDSL are never both fixed. This concludes the handling of column *CGD* in the current tier. Other columns *c* are then considered one-at-a-time according to the ranking of their *Depth*(*c*) values, and the above process repeats until all values in the current tier in BASLDSL have been fixed to 1 or 0 (lines 6–16).

The procedure then continues to the next tier in the current scanning direction *Dir* (line 17). Eventually all tiers in direction *Dir* will be scanned. At this point, the second *phase* of the scanning commences: the procedure jumps back to tier *InitialTier*, un-fixes all values in that tier, and begins scanning in the direction opposite from *InitialDirection* (lines 19–20). This second phase is undertaken if and only if *InitialTier* is a middle tier (line 18). After completion of the second phase, all tiers in BASLDSL have been scanned and all cells in BASLDSL have been fixed (line 21).

Algorithm 3. The constructive heuristic.

```
1   Convert the problem instance into BASLDSL (see Figure 2.2);
2   Call Algorithm 2 to fix the values in BASLDSL;
3   Convert BASLDSL into ContainerArray, an S × T array that shows the type of container occupying
    each cell in the storage bay. "S" ("D") indicates a container that will be single-spreader (dual-
    spreader) lifted. "n" indicates that no container is present. Let CA[s, t] denote the value in stack s
    and tier t in ContainerArray;
4   Set CurrSpreader = InitialSpreader;
5   while there is at least one "S" or "D" in ContainerArray do
6   |   for t = T to 1 (decrease t by 1 each time) do
7   |   |   for s = 1 to S (increase s by 1 each time) do
8   |   |   |   if CA[s, t] = CurrSpreader = "S" and (CA[s, t+1] = "n" or does not exist) then
9   |   |   |   |   Add a single-spreader lift to the end of the crane lift sequence and let CA[s, t] =
    |   |   |   |   "n";
10  |   |   |   if CA[s, t] = CurrSpreader = "D" then
11  |   |   |   |   if CA[s, t+1] = CA[s+1, t+1] = "n" or neither value exists then
12  |   |   |   |   |   Add a dual-spreader lift to the end of the crane lift sequence and let CA[s, t]
    |   |   |   |   |   = CA[s+1, t] = "n";
13  |   |   |   |   Increase s by 1
14  |   Change CurrSpreader from "S" to "D" or vice versa;
15  if the crane lift sequence has short, unprofitable subsequences of dual-spreader lifts that are not
    worth the changeover cost then
16  |   Convert the unprofitable dual-spreader lifts into single-spreader lifts in the crane lift sequence;
17  Report the final crane lift sequence and its makespan M;
```

The overall constructive heuristic is shown in Algorithm 3. This heuristic first calls Algorithm 2 to fix the values in BASLDSL (line 2). It then converts the fixed BASLDSL into a *ContainerArray*, an $S \times T$ array that shows which containers in the storage bay will be single-spreader ("*S*") and dual-spreader ("*D*") lifted. A detailed crane lift sequence that starts using spreader *InitialSpreader* is then constructed in a straightforward manner using the values in *ContainerArray* (lines 4–14). If the end (middle) of the sequence contains short, unprofitable subsequences of dual-spreader lifts that are not worth the cost of one (two) spreader changeover(s), the unprofitable dual-spreader lifts are converted into single-spreader lifts (lines 15–16). Then the makespan of the resulting crane lift sequence is computed (line 17).

Figures 2.3 and 2.4 show how the constructive heuristic generates two different crane lift sequences for the problem instance shown in Figure 2.2 for two different sets of input parameters.

In each figure, the original problem instance and its conversion into the initial, tentative BASLDSL are shown in the upper left. The values of the six input parameters are shown in the upper-right. The left side of each figure shows how Algorithm 2 gradually fixes the values in BASLDSL using the first five parameters. Note that only the first several values in arrays *ColumnChooser*[ ], *FixFullDepthYN*[ ], and *NumLiftsToFix*[ ] are utilized; the other values are not used. The conversion of the fixed BASLDSL into *ContainerArray*; the generation of a detailed crane lift sequence; and the makespan computation are shown in the bottom right. Notice that the fixed BASLDSL, *ContainerArray*, and makespan are quite different in the two figures. Indeed, the constructive heuristic is able to generate vastly different crane lift sequences when the input parameters are changed. It turns out that the makespan shown in Figure 2.4 is optimal.

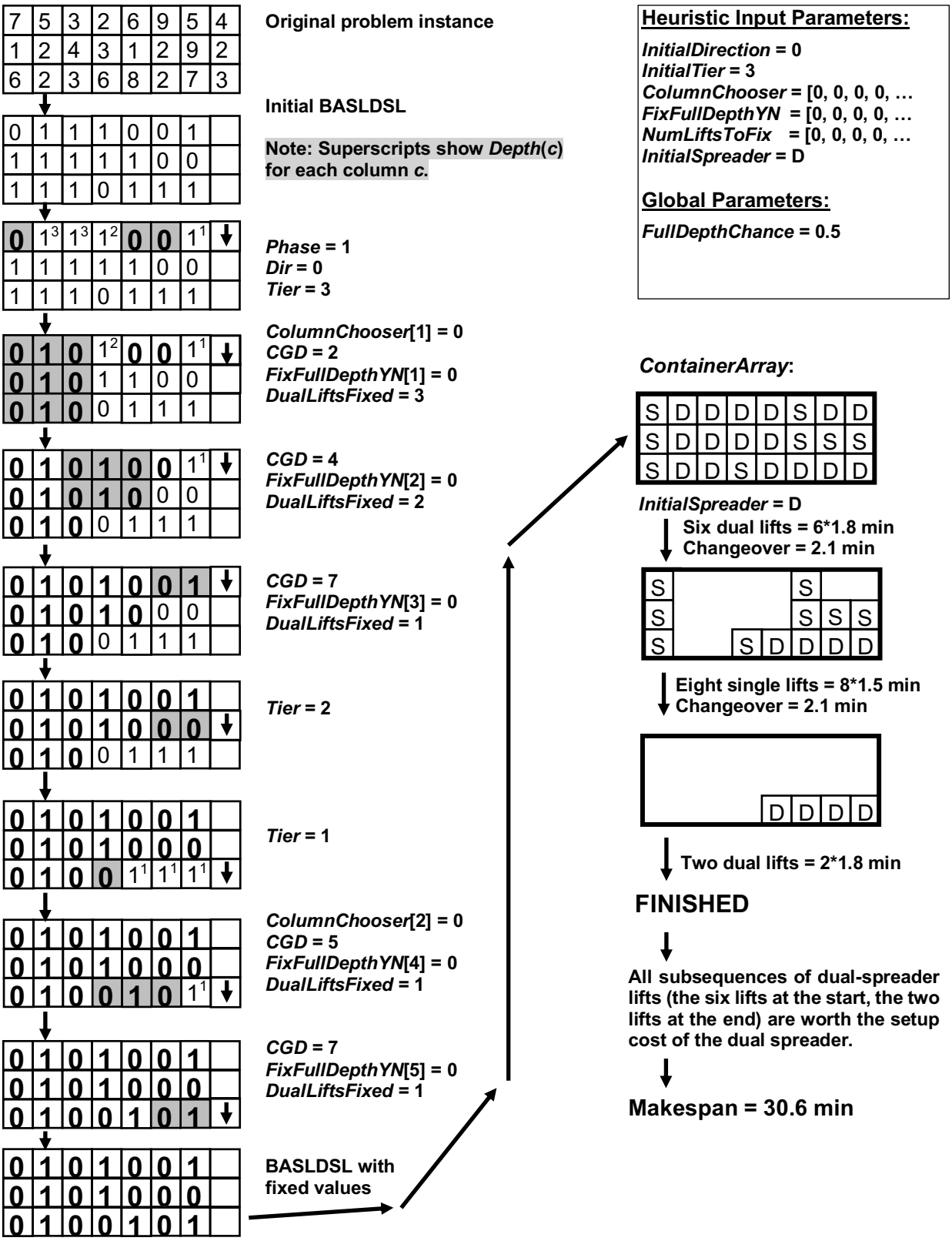

Figure 2.3. Illustration #1 of the constructive heuristic. The most recent activity in BASLDSL is highlighted. Fixed values in BASLDSL are displayed in bold.

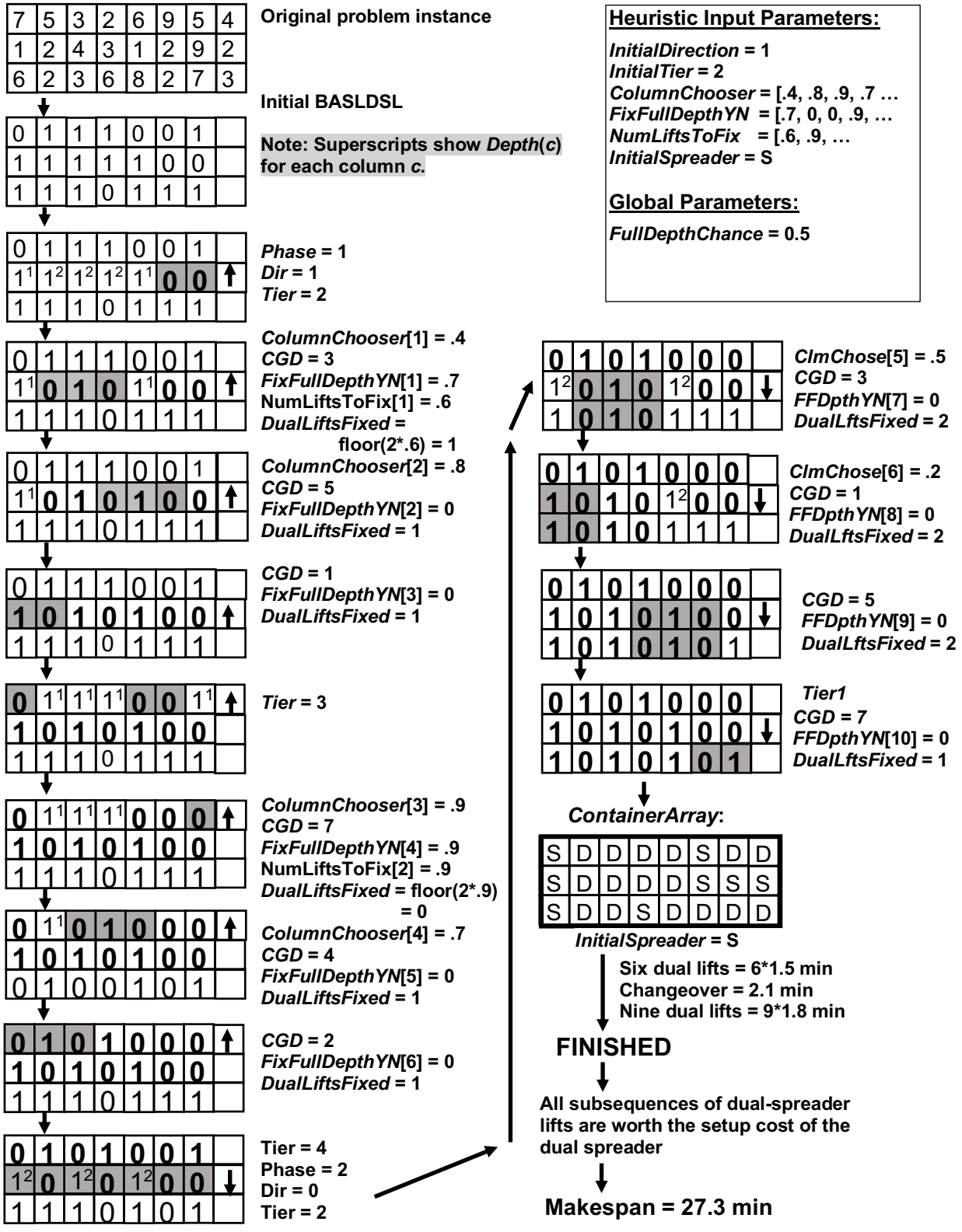

Figure 2.4. Illustration #2 of the constructive heuristic. The most recent activity in BASLDSL is highlighted. Fixed values in BASLDSL are displayed in bold.

Figure 2.5 shows how the constructive heuristic is embedded within a simulated annealing (SA) framework. The SA procedure begins by initializing the six input parameters, collectively referred to as *CurrParam*, and converting them into a feasible crane lift sequence *CurrSeq* and makespan *CurrOV* using the constructive heuristic. In each SA iteration, *CurrParam* is used to generate a new set of parameter values *NghborParam*; the constructive heuristic converts *NghborParam* into a feasible crane lift sequence; and the laws of simulated annealing decide if *NghborParam* replaces *CurrParam*.

Two kinds of neighborhood moves—small moves and large moves—are utilized. In a small move, a new, random combination of values for parameters 1 and 6—*InitialDirection* and *InitialSpreader*—is considered and the other parameters remain unchanged. In a large move, a new, random value for parameter (2, 3, 4, 5) is considered with probability (p_2, p_3, p_4, p_5) and a new, random combination of values for parameters 1 and 6 is considered ($p_2 + p_3 + p_4 + p_5 = 1$). A small move is made whenever it can produce a new set of parameter values that has not yet been explored. Otherwise, a large move is initiated. When a predefined time limit is reached, the SA procedure terminates and the best crane lift sequence that was found is displayed.

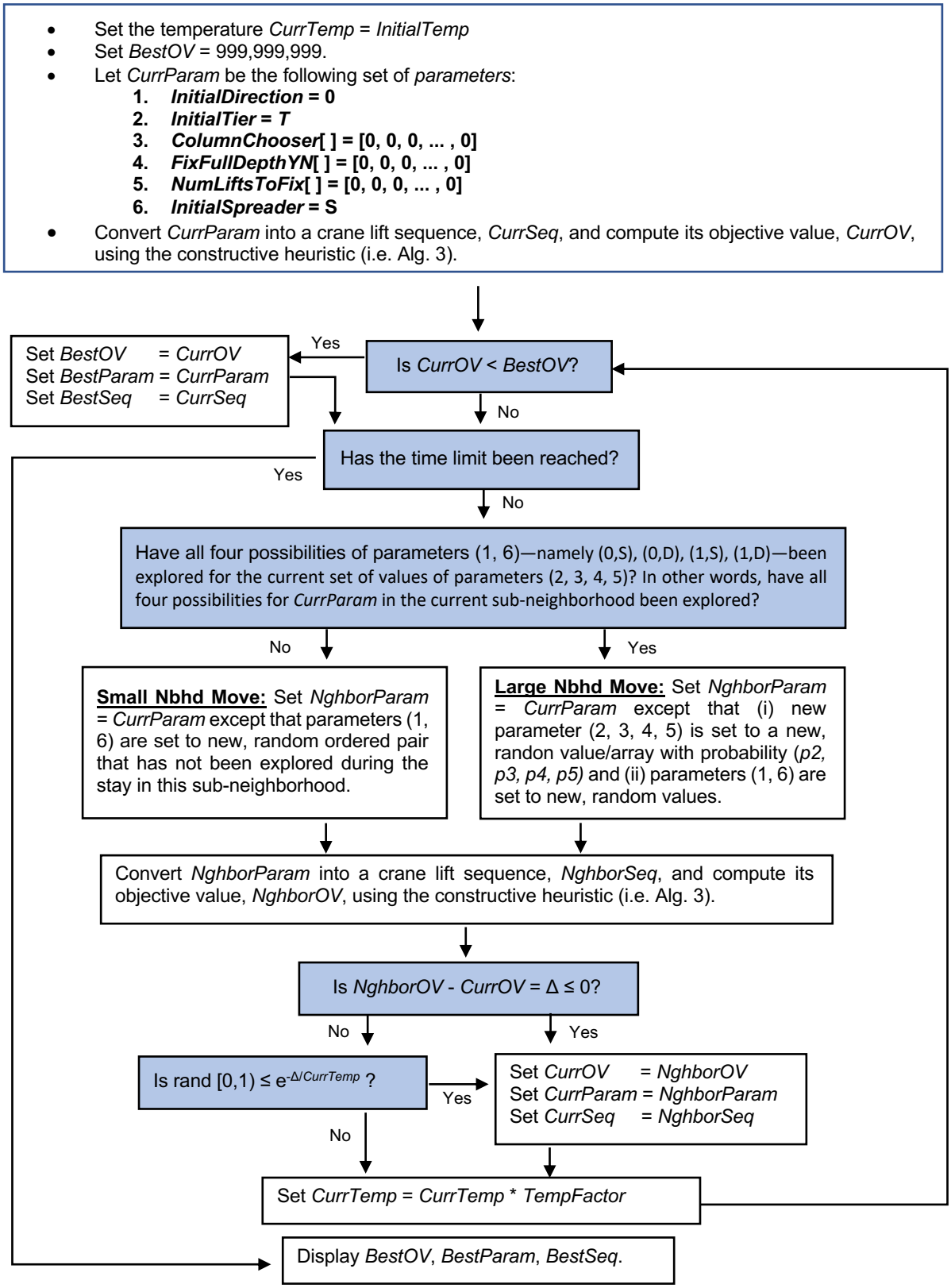

Figure 2.5. Overall logic of the simulated annealing metaheuristic.

2.4 Experimental setup, results, and discussion

The heuristic method from Section 2.3 and model DSCSP from Section 2.2 were coded into MS Visual C++ 2010 Professional. IBM ILOG Concert Technology was used to define model DSCSP within C++ and call the MILP solver IBM ILOG CPLEX 12.5 to solve instances defined in text files. To avoid running out of memory, the CPLEX "node file storage parameter" is set to 3. That is, the information for every unexplored node in the CPLEX branch-and-cut tree is stored on the hard disk and compressed. Otherwise, default CPLEX settings are used. All results are obtained using a desktop computer with the Windows 7 Enterprise 64-bit operating system, an Intel Core i7-4770 processor with eight 3.4 gigahertz cores, and 16 gigabytes of RAM.

We consider a total of 120 problem instances—30 instances for each of the problem sizes 3×8, 5×10, 10×23, and 50×50. A problem of size $T \times S$ has T tiers, S stacks, and T containers in stack s at time 0 for all s. In all instances, we assume that the container weights W_{st} take integer values from 1 to 9. We also assume that $w_{Limit} = 10$, $H_1 = 1.5$, $H_2 = 1.8$, and $C = 2.1$. Among the 30 instances for each problem size, (10, 10, 10) instances have (light, medium, heavy) container weights. In the medium instances, the weight of each container follows a discrete uniform distribution over the values {1, 2, 3, 4, 5, 6, 7, 8, 9}. In the light instances, the weight of each container has a {15%, 15%, 15%, 15%, 20%, 5%, 5%, 5%, 5%} chance of taking the value {1, 2, 3, 4, 5, 6, 7, 8, 9}. In the heavy instances, the weight of each container has a {5%, 5%, 5%, 5%, 20%, 15%, 15%, 15%, 15%} chance of taking the value {1, 2, 3, 4, 5, 6, 7, 8, 9}.

Table 2.2 shows the settings used in the heuristic algorithm for each of the four problem sizes. Preliminary experiments were performed to determine these settings. The results of these preliminary experiments, not shown here, indicated that performance was most consistent when the neighbor probabilities (p_2, p_3, p_4, p_5) equal (.25, .25, .25, .25) (Fig. 2.5). We also found that

performance improves with a higher *FullDepthChance* (line 8 of Algorithm 2, middle of Table 2.1) as the problem size increases. We hypothesize that this is because instances with more tiers can have larger batches of dual lift opportunities that are aligned vertically (i.e. longer strings of consecutive 'ones' in the same column in the initial BASLDSL). As the size of such a batch increases, it may be increasingly important to accept all dual lift opportunities in the batch to prevent "disrupting" the batch with a single rejected dual lift opportunity in its middle. Such a disruption may add two unnecessary changeovers—to and from the single spreader—to the crane lift sequence.

Table 2.2. Parameters settings for the heuristic method.

Problem size	3 × 8	5 × 10	10 × 23	50 × 50
Computational time limit (seconds)	10	60	600	600
FullDepthChance	0.5	0.7	0.8	0.99
Neighbor probabilities (p_2, p_3, p_4, p_5)	All set to (.25, .25, .25, .25)			
InitialTemp	10,000	100,000	10,000,000	100,000
TempFactor	0.9999	0.99999	0.999999	0.99999

Note that more computation time is allocated for attacking larger problems. Given this time limit, parameters *InitialTemp* and *TempFactor* are set so the SA procedure consists of three phases of roughly equal duration—(i) an initial exploration phase when almost any neighboring solution is accepted; (ii) a middle phase when the algorithm gradually transitions from being very accepting of neighbors to being very picky; and (iii) a final phase when virtually no inferior neighboring solutions are accepted. The increase (decrease) in parameters *InitialTemp* and *TempFactor* when going from problem size 3 × 8 to 10 × 23 (10 × 23 to 50 × 50) follows from the fact that the allotted computation time grows at a faster (slower) rate than the problem size for these instances.

Table 2.3 shows the results for the first set of experiments that consider the 30 small problem instances of size 3 × 8. Each individual instance is specified by a code "*T*x*S*Znn" where *T* is the number of tiers; *S* is the number of stacks; Z takes the value (L, M, H) according to the container weight scenario (light, medium, heavy); and "nn" denotes the instance number from 1 to 10. Each instance is considered in three ways—(A) using standard integer programming (IP) with no time limit ("CPLEX Alone"); (B) using the heuristic method with a 10 sec time limit; and (C) using IP with no time limit where both the lower bound and the best solution found by the heuristic are passed to the solver at the outset ("CPLEX + LB + UB"). The heuristic method generates an average of eleven million neighboring solutions within the 10 second time limit. The best objective value (i.e. makespan) found by methods A, B, and C are shown in columns M_{CP}, M_H, and M_{CP+B} respectively. Column M_{H_0} shows the makespan of the initial feasible solution used in the heuristic method, and column LB shows the lower bound.

Table 2.3. Experimental results for DSCSP instances of size 3 × 8.

Instance	CPLEX Alone		Heuristic (10 seconds)			CPLEX+LB+UB		LB	$\frac{M_H - LB}{M_H}$
	M_{CP}	Time(s)	M_H	M_{H_0}	$\frac{M_{H_0} - M_H}{M_H}$	M_{CP+B}	Time(s)		
3x8L01	27.3	17	27.3	29.4	7.69%	27.3	15	26.1	4.40%
3x8L02	26.1	12	26.1	31.8	21.84%	26.1	0	26.1	0.00%
3x8L03	26.1	6	26.1	27.3	4.60%	26.1	1	26.1	0.00%
3x8L04	29.4	211	29.4	31.8	8.16%	29.4	112	27.3	7.14%
3x8L05	21.6	1	21.6	28.2	30.56%	21.6	1	21.6	0.00%
3x8L06	27.3	80	27.3	33.9	24.18%	27.3	71	26.1	4.40%
3x8L07	24.9	4	24.9	28.2	13.25%	24.9	0	24.9	0.00%
3x8L08	30.6	573	30.6	31.8	3.92%	30.6	157	28.5	6.86%
3x8L09	26.1	7	26.1	30.6	17.24%	26.1	0	26.1	0.00%
3x8L10	28.2	424	28.2	31.5	11.70%	28.2	75	26.1	7.45%
Average	26.8	133.5	26.8	30.5	14.31%	26.8	43.2	25.9	3.02%
3x8M01	30.6	8	30.6	31.8	3.92%	30.6	42	28.5	6.86%
3x8M02	30.9	37	30.9	30.9	0.00%	30.9	25	29.7	3.88%
3x8M03	27.3	29	27.3	31.8	16.48%	27.3	53	26.1	4.40%
3x8M04	34.2	76	34.2	35.7	4.39%	34.2	55	32.1	6.14%
3x8M05	30.9	109	30.9	33	6.80%	30.9	108	28.5	7.77%
3x8M06	27.3	7	27.3	33	20.88%	27.3	0	27.3	0.00%
3x8M07	30.9	56	30.9	33	6.80%	30.9	63	29.7	3.88%
3x8M08	32.1	17	32.1	33	2.80%	32.1	24	30.9	3.74%
3x8M09	34.2	14	34.2	34.2	0.00%	34.2	31	32.1	6.14%
3x8M10	28.5	6	28.5	30.6	7.37%	28.5	33	27.3	4.21%
Average	30.7	35.9	30.7	32.7	6.94%	30.7	43.4	29.2	4.70%
3x8H01	35.7	6	35.7	36	0.84%	35.7	19	34.5	3.36%
3x8H02	32.1	27	32.1	34.2	6.54%	32.1	40	30.9	3.74%
3x8H03	35.7	9	35.7	36	0.84%	35.7	12	34.5	3.36%
3x8H04	29.7	10	29.7	31.8	7.07%	29.7	0	29.7	0.00%
3x8H05	33.3	8	33.3	35.4	6.31%	33.3	1	33.3	0.00%
3x8H06	34.5	7	34.5	35.4	2.61%	34.5	19	33.3	3.48%
3x8H07	32.1	24	32.1	32.1	0.00%	32.1	38	30.9	3.74%
3x8H08	32.1	8	32.1	35.4	10.28%	32.1	16	30.9	3.74%
3x8H09	34.2	22	34.2	34.2	0.00%	34.2	25	32.1	6.14%
3x8H10	36	1	36	36	0.00%	36	1	36	0.00%
Average	33.5	12.2	33.5	34.7	3.45%	33.5	17.1	32.6	2.76%
Ovrl Avg.	30.3	60.5	30.3	32.6	8.24%	30.3	34.6	29.2	3.49%

The results show that all three methods solve all instances to optimality. Indeed, method A solves all instances to optimality within ten minutes; method B finds these optimal solutions within ten seconds; and method C solves all instances to optimality within three minutes. Importantly, the best solutions found by the heuristic (all of which happen to be optimal) are usually within 5% of the lower bound. Also, the objective value of the best heuristic solution (30.3) is about 7% better on average than that of the initial heuristic solution (32.6). Not surprisingly, the optimal values for the light instances are typically less than those for the medium instances. The same holds for the medium instances compared to the heavy instances.

Table 2.4 shows the results for the second set of experiments that consider the 30 medium-sized instances of size 5 × 10. Here, each instance is considered in three ways—(A) using IP with a one hour time limit ("CPLEX Alone"); (B) using the heuristic method with a 60 second time limit; and (C) using IP with a one hour time limit where the lower bound and best solution found by the heuristic are passed to the solver at the outset ("CPLEX+LB+UB"). The heuristic method generates an average of 31 million neighboring solutions within the 60 sec time limit. Column "Opt?" indicates if the associated solution is proven by CPLEX to be optimal or not.

The results for methods A and B show that the heuristic method performs better than IP. Indeed, in every instance, the best solution found by the heuristic in one minute is at least as good as the best solution found by CPLEX in an hour. Also, the average objective value of the best heuristic solution (62.6) is about 1.4% better than that for CPLEX (63.5). Note that the best solutions found by the heuristic (only two of which are known to be optimal) are about 6% higher on average than the lower bound. Also, the results for method C show that CPLEX is unable, in an hour, to improve upon the best heuristic solution provided to it at the outset for any instance. Finally, we observe that the objective value of the best heuristic solution (62.6) is about 5% lower

on average than that of the initial heuristic solution (65.7). Overall, the above results indicate that our IP framework is not a suitable solution method for instances with 50 or more containers. Thus, IP is not considered in the following experiments that consider larger problem instances.

Table 2.5 shows the results for the 60 largest problem instances—30 large instances of size 10 × 23 and 30 very large instances of size 50 × 50. The results for the large (very large) instances are on the left (right). For each instance, we show the lower bound; the objective value of the best solution found by the heuristic method within a 600 second time limit; and the objective value of the initial heuristic solution. The heuristic method generates an average of 39 (1.9) million neighboring solutions within the 600 second time limit for the instances of size 10 × 23 (50 × 50).

The results show that the heuristic is finding near-optimal solutions to these instances. Indeed, the average objective value of the best heuristic solution for the instances of size 10 × 23 and 50 × 50—275.2 and 2936.8 respectively—is about 4% higher than the average lower bound—263.8 and 2828.9 respectively—for these instances. Also, note that the quality of the heuristic solution improves as containers get heavier; on average, the makespan of the best heuristic solution is roughly 5%, 4%, and 3% above the lower bound for the light, medium, and heavy instances respectively. This may be due to the fact that there are fewer opportunities for using the dual spreader—and therefore fewer choices—when containers are heavier. Overall, the heuristic method is effective in tackling a variety of instances of the DSCSP.

Table 2.4. Experimental results for DSCSP instances of size 5 × 10.

Instance	CPLEX Alone			Heuristic (10 seconds)			CPLEX+LB+UB			LB	$\frac{M_H - LB}{M_H}$
	M_{CP}	Time(s)	Opt?	M_H	M_{H_0}	$\frac{M_{H_0} - M_H}{M_H}$	M_{CP+B}	Time(s)	Opt ?		
5x10L01	62.1	3861	?	61.2	63	2.94%	61.2	3605	?	56.7	7.35%
5x10L02	55.2	3915	?	55.2	63.6	15.22%	55.2	3610	?	53.1	3.80%
5x10L03	57.3	4001	?	56.4	61.2	8.51%	56.4	3608	?	53.1	5.85%
5x10L04	56.1	3726	?	55.2	58.8	6.52%	55.2	3607	?	51.9	5.98%
5x10L05	50.7	3673	?	50.7	60	18.34%	50.7	3603	?	49.5	2.37%
5x10L06	66	3665	?	63.3	67.8	7.11%	63.3	3605	?	57.9	8.53%
5x10L07	56.1	3850	?	56.1	61.2	9.09%	56.1	3604	?	51.9	7.49%
5x10L08	56.4	4114	?	56.4	57.6	2.13%	56.4	3605	?	53.1	5.85%
5x10L09	51.6	3604	?	51.6	58.8	13.95%	51.6	3779	?	49.5	4.07%
5x10L10	51.6	3600	?	51.6	59.4	15.12%	51.6	3663	?	49.5	4.07%
Avg.	56.3	3800.9		55.8	61.1	9.89%	55.8	3628.9		52.6	5.54%
5x10M01	67.8	3602	?	64.8	66.6	2.78%	64.8	3605	?	60.3	6.94%
5x10M02	63.9	3766	?	62.4	66	5.77%	62.4	3604	?	57.9	7.21%
5x10M03	63.9	3611	?	63.6	66	3.77%	63.6	3606	?	60.3	5.19%
5x10M04	64.8	3769	?	64.8	67.2	3.70%	64.8	3606	?	61.5	5.09%
5x10M05	66	3658	?	64.8	64.8	0.00%	64.8	3607	?	61.5	5.09%
5x10M06	62.1	3669	?	57.6	59.7	3.65%	57.6	3603	?	54.3	5.73%
5x10M07	60.6	3625	?	59.7	63	5.53%	59.7	3604	?	54.3	9.05%
5x10M08	68.1	3726	?	66.9	69.9	4.48%	66.9	3607	?	61.5	8.07%
5x10M09	64.2	3652	?	62.1	63.3	1.93%	62.1	3606	?	57.9	6.76%
5x10M10	57.3	3602	?	55.5	61.8	11.35%	55.5	3604	?	51.9	6.49%
Avg.	63.9	3668		62.2	64.8	4.30%	62.2	3605.2		58.1	6.56%
5x10H01	74.4	223	yes	74.4	74.4	0.00%	74.4	114	yes	72.3	2.82%
5x10H02	69.9	3641	?	69.3	70.5	1.73%	69.3	3608	?	65.1	6.06%
5x10H03	70.2	3606	?	66.9	68.1	1.79%	66.9	3603	?	62.7	6.28%
5x10H04	69.3	3608	?	66.9	67.8	1.35%	66.9	3604	?	61.5	8.07%
5x10H05	75	41	yes	75	75	0.00%	75	104	yes	73.5	2.00%
5x10H06	72.9	3690	?	72.9	73.2	0.41%	72.9	3607	?	66.3	9.05%
5x10H07	69.6	3602	?	69.6	72	3.45%	69.6	3608	?	66.3	4.74%
5x10H08	62.4	3669	?	62.4	65.7	5.29%	62.4	3623	?	59.1	5.29%
5x10H09	69.6	3609	?	69.6	72	3.45%	69.6	3606	?	66.3	4.74%
5x10H10	70.8	3616	?	70.8	71.7	1.27%	70.8	3606	?	67.5	4.66%
Avg.	70.4	2930.5		69.8	71	1.87%	69.8	2908.3		66.1	5.37%
Ovrl Avg.	63.5	3466.5		62.6	65.7	5.35%	62.6	3380.8		58.9	5.82%

Table 2.5. Experimental results for DSCSP instances of size 10 × 23 (left) and 50 × 50 (right).

Instance	LB	M_H	M_{H_0}	$\frac{M_{H_0} - M_H}{M_H}$	$\frac{M_H - LB}{M_H}$	Instance	LB	M_H	M_{H_0}	$\frac{M_{H_0} - M_H}{M_H}$	$\frac{M_H - LB}{M_H}$
10x23L01	228.3	237.9	249.6	4.92%	4.04%	50x50L01	2496.9	2628.6	2676.6	1.83%	5.01%
10x23L02	225.9	234.6	242.4	3.32%	3.71%	50x50L02	2471.7	2605.5	2659.8	2.08%	5.14%
10x23L03	231.9	241.8	254.7	5.33%	4.09%	50x50L03	2500.5	2625.9	2661.9	1.37%	4.78%
10x23L04	228.3	239.4	253.5	5.89%	4.64%	50x50L04	2490.9	2621.1	2667.3	1.76%	4.97%
10x23L05	231.9	243.9	258.9	6.15%	4.92%	50x50L05	2492.1	2645.7	2689.2	1.64%	5.81%
10x23L06	230.7	240.6	246.3	2.37%	4.11%	50x50L06	2514.9	2644.8	2692.5	1.80%	4.91%
10x23L07	235.5	247.5	260.4	5.21%	4.85%	50x50L07	2505.3	2621.4	2658	1.40%	4.43%
10x23L08	229.5	243	249.6	2.72%	5.56%	50x50L08	2472.9	2593.2	2632.8	1.53%	4.64%
10x23L09	237.9	248.7	266.7	7.24%	4.34%	50x50L09	2494.5	2616	2668.5	2.01%	4.64%
10x23L10	237.9	247.5	259.5	4.85%	3.88%	50x50L10	2489.7	2632.8	2684.4	1.96%	5.44%
Average	231.8	242.5	254.2	4.80%	4.41%	Average	2492.9	2623.5	2669.1	1.74%	4.98%
10x23M01	264.3	278.1	281.4	1.19%	4.96%	50x50M01	2792.1	2902.5	2925	0.78%	3.80%
10x23M02	259.5	273.6	280.8	2.63%	5.15%	50x50M02	2789.7	2899.2	2919.6	0.70%	3.78%
10x23M03	263.1	274.8	283.2	3.06%	4.26%	50x50M03	2801.7	2902.5	2929.8	0.94%	3.47%
10x23M04	260.7	272.4	279.6	2.64%	4.30%	50x50M04	2834.1	2944.8	2970	0.86%	3.76%
10x23M05	269.1	279.9	284.4	1.61%	3.86%	50x50M05	2818.5	2915.7	2940.6	0.85%	3.33%
10x23M06	265.5	279.3	285.6	2.26%	4.94%	50x50M06	2810.1	2918.4	2942.1	0.81%	3.71%
10x23M07	264.3	274.8	277.2	0.87%	3.82%	50x50M07	2793.3	2915.7	2931	0.52%	4.20%
10x23M08	255.9	264.3	272.4	3.06%	3.18%	50x50M08	2780.1	2912.4	2931.9	0.67%	4.54%
10x23M09	261.9	273.6	279.6	2.19%	4.28%	50x50M09	2819.7	2940.9	2958	0.58%	4.12%
10x23M10	261.9	273.6	274.8	0.44%	4.28%	50x50M10	2772.9	2889.6	2902.2	0.44%	4.04%
Average	262.6	274.4	279.9	2.00%	4.30%	Average	2801.2	2914.2	2935	0.72%	3.88%
10x23H01	293.1	304.8	309.6	1.57%	3.84%	50x50H01	3208.5	3294	3300.3	0.19%	2.60%
10x23H02	294.3	306.9	308.1	0.39%	4.11%	50x50H02	3188.1	3268.2	3287.4	0.59%	2.45%
10x23H03	299.1	309.6	315.3	1.84%	3.39%	50x50H03	3196.5	3272.7	3279.6	0.21%	2.33%
10x23H04	296.7	309.3	311.4	0.68%	4.07%	50x50H04	3167.7	3258.6	3273	0.44%	2.79%
10x23H05	300.3	310.8	314.4	1.16%	3.38%	50x50H05	3186.9	3264.6	3277.8	0.40%	2.38%
10x23H06	295.5	308.1	308.1	0.00%	4.09%	50x50H06	3216.9	3290.4	3302.4	0.36%	2.23%
10x23H07	294.3	306.9	307.2	0.10%	4.11%	50x50H07	3204.9	3281.1	3289.2	0.25%	2.32%
10x23H08	299.1	310.8	315.6	1.54%	3.76%	50x50H08	3201.3	3286.5	3297.6	0.34%	2.59%
10x23H09	297.9	308.4	308.4	0.00%	3.40%	50x50H09	3189.3	3265.5	3277.8	0.38%	2.33%
10x23H10	299.1	309.6	315.3	1.84%	3.39%	50x50H10	3165.3	3246.3	3256.2	0.30%	2.50%
Average	296.9	308.5	311.3	0.91%	3.75%	Average	3192.5	3272.8	3284.1	0.35%	2.45%
Ovrl Avg.	263.8	275.2	281.8	2.57%	4.16%	Ovrl Avg.	2828.9	2936.8	2962.8	0.93%	3.77%

Chapter 3:

Triple-Spreader Crane Scheduling Problem

3.1 Problem description

This chapter discusses the triple-spreader crane scheduling problem (TSCSP). We define the TSCSP as follows. Consider the deck of a ship on which is placed a set of identically sized *containers* (blocks, items). Due to space limitations, these containers are placed directly on top of each other in a storage bay (i.e. bay) occupying no more than S stacks and T tiers. At time 0, there are E_s containers in stack s ($E_s \leq T$). The weight of the container in stack s, tier t is given by W_{st}. Consider the problem of sequencing the lifts made by one crane that will remove all containers from the bay. This crane can operate in three modes: single-spreader, dual-spreader, or triple-spreader mode. When in single-spreader mode, the crane may remove any single container from the top of any stack. This type of lift takes H_1 minutes. When in dual-spreader mode, the crane may simultaneously remove two containers in the same tier from the top of any two adjacent stacks as long as the sum of their weights does not exceed L_2. This type of lift takes H_2 minutes. When in triple-spreader mode, the crane may simultaneously remove three containers in the same tier from the top of any three adjacent stacks as long as the sum of their weights does not exceed L_3. This type of lift takes H_3 minutes. Furthermore, the time required to change from operating in p-spreader mode to operating in q-spreader mode—the spreader changeover time (i.e. changeover time)—is C_{pq} minutes ($1 \leq p, q \leq 3$). The crane can begin in any mode at time 0 with no initial setup cost. The goal is to sequence the individual lifts and changeovers of the crane so as to minimize the total time needed to remove all containers from the bay. To make the problem

meaningful, we assume that $S \geq 3$, $H_2 < 2H_1$, $H_3 < 3H_1$, $L_2 < 2*\max\{W_{st}\}$, and $L_3 < 3*\max\{W_{st}\}$. Figure 3.1 shows an instance of the TSCSP. In this instance, $S = 8$, $T = 3$, $E_{st} = 3$ for all s, and the weights W_{st} of all containers in the bay are shown in the upper-left corner of the figure. In addition, we assume that $L_2 = 10$, $L_3 = 12$, $H_1 = 1.5$, $H_2 = 1.8$, $H_3 = 2.2$, and $C_{pq} = 2.7$ for all p and q. Note that, even for this small instance, it is not easy to decide how the containers should be lifted. Figure 3.1 shows a feasible crane lift sequence for this instance. This sequence consists of four dual-spreader lifts followed by four single-spreader lifts and four triple-spreader lifts. Two changeovers between spreader modes take place, so the total time needed to empty the bay—the makespan—is $4\times1.8 + 4\times1.5 + 4\times2.2 + 2\times2.7 = 27.4$ minutes. We later show that this is not the optimal makespan for this instance. Note that, in Cheng et al. (2020), the authors prove that the multi-spreader crane scheduling problem (MSCSP) is NP-hard when the crane has three or more modes. In particular, the TSCSP is NP-hard.

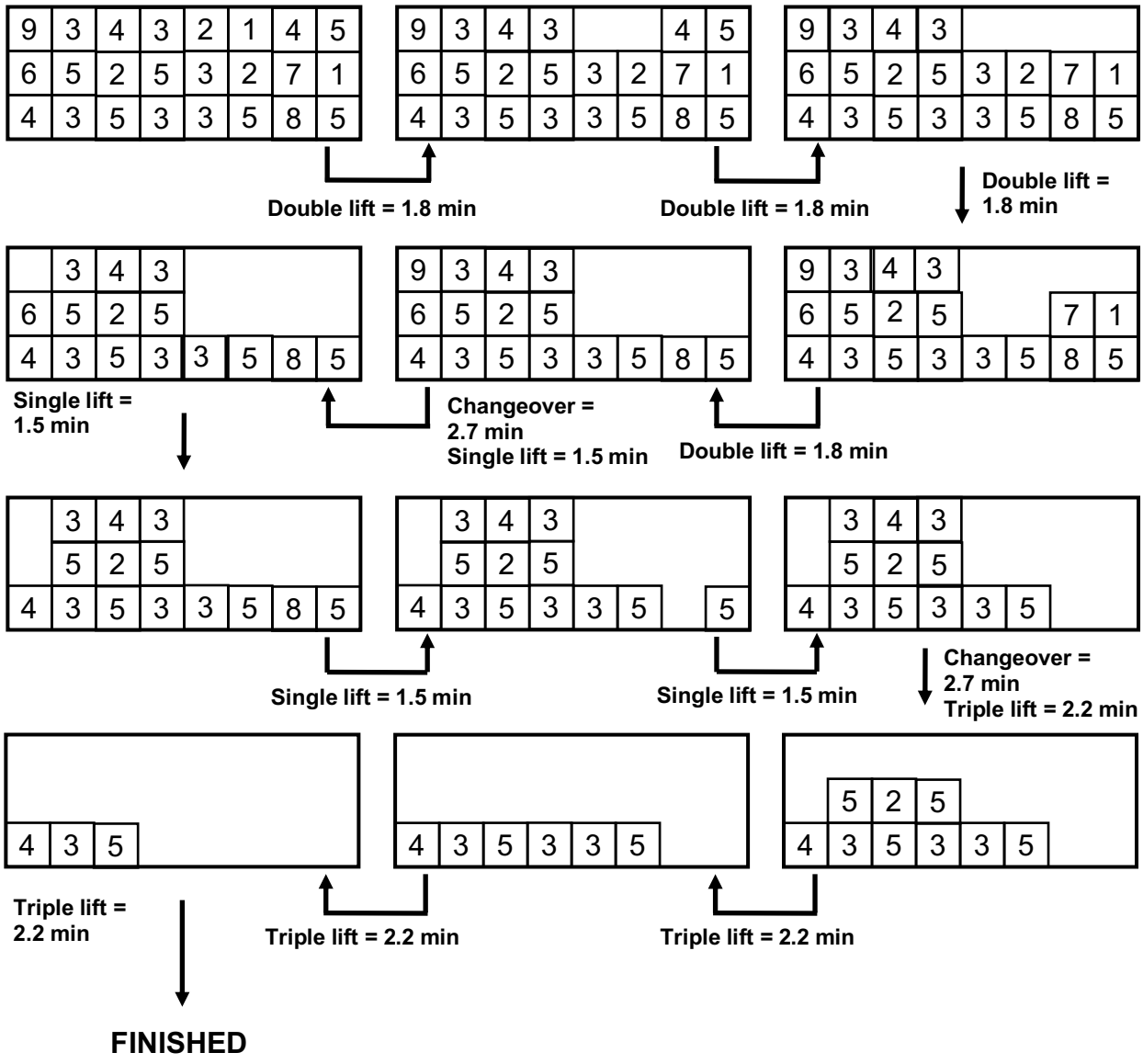

Figure 3.1. Feasible unloading sequence with makespan 27.4 minutes for a problem instance of size 3x8.

3.2 Mathematical model

We now present a math model of the TSCSP. To facilitate model development, we first convert the problem instance into two binary matrices: one indicating the legal dual-spreader lifts ($L2$), and one indicating the legal triple-spreader lifts ($L3$). Figure 3.2 depicts the conversion of the problem instance from Figure 3.1, where binary variables indicate whether a dual-spreader

(triple-spreader) lift could be performed on a pair (trio) of adjacent containers in the same tier. Without loss of generality, we use the left-most container in the set of adjacent containers to denote whether a legal dual-spreader or a triple-spreader lift can be performed within the given weight limit L_2 or L_3. For example, the top-left '0' in Figure 3.2(a) indicates that the first and the second containers (from the left side) in the top tier cannot be dual-spreader lifted because their combined weight—12—exceeds L_2 = 10. Also, the '1' to the right of the top-left '0' indicates that the second and the third containers (from the left) in the top tier can be dual-spreader lifted because their combined weight—7—does not exceed L_2. Similarly, in Figure 3.2(b), the top-left '0' indicates that the first, second, and third containers (from the left) in the top tier cannot be triple-spreader lifted because their combined weight—16—exceeds L_3 = 12. Also, the '1' adjacent to the top-left '0' indicates that the second, third, and fourth containers (from the left) in the top tier can be triple-spreader lifted because their combined weight—10—does not exceed L_3. In the original problem instance and the binary matrices, we number the tiers 1, …, *T* from bottom to top and the stacks 1, …, *S* from left to right.

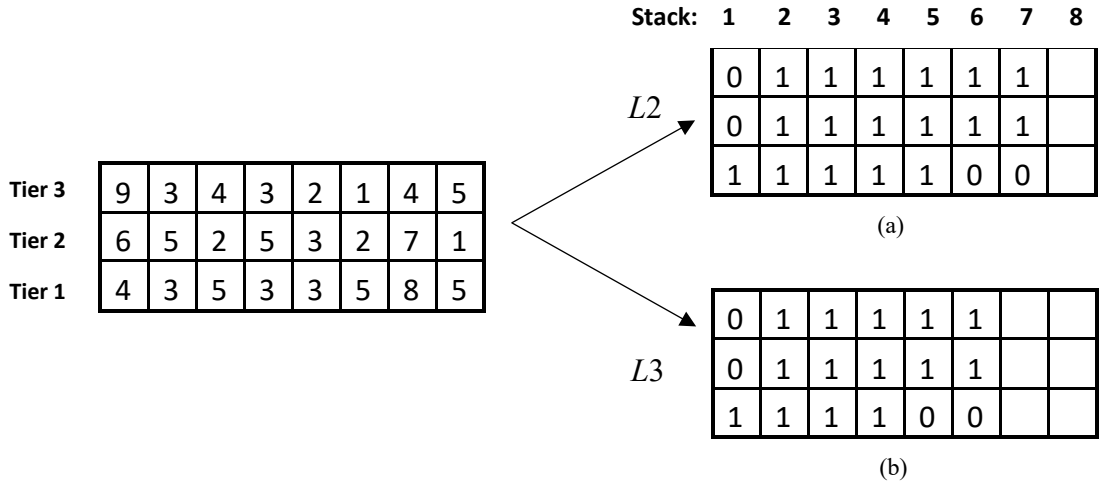

Figure 3.2. Conversion of problem instance (left) into binary array showing (a) legal dual spreader lifts (upper-right) and (b) legal triple spreader lifts (lower-right) assuming L_2 = 10 and L_3 = 12.

Our mathematical model, model TSCSP, discretizes time into intervals. During each time interval, at most one (single-spreader, dual-spreader, or triple-spreader) lift may occur. The duration of an interval is therefore H_1, H_2, or H_3 minutes depending on the type of operation performed. Between two consecutive intervals, at most one changeover may occur.

Table 3.1 shows the indices, input parameters, and decision variables in model TSCSP. Input parameters *S*, *T*, E_s, C_{pq} and H_p were discussed in Section 3.1. Input parameters $L2_{st}$ and $L3_{st}$ refer to values in the aforementioned binary matrices $L2$ and $L3$ respectively (Figure 3.2). Parameter *I*—the number of time intervals available for lifting containers—is conservatively set to $S \times T$: the number of time intervals needed if the storage bay begins full and the crane only operates in single-spreader mode.

Decision variables X_{si}, Y_{si}, and Z_{si} indicate the type of lift conducted during each time interval and above which stack(s) the lift is performed. Variable R_{ti} indicates the tier (if any) from which containers are lifted during each time interval. Variable G_{pqi} indicates if a changeover occurs between two consecutive time intervals. Variable N_{si} tracks the height of the container stacks at the beginning of each time interval, and F_i indicates whether the lifting is finished (= 1) or not (= 0) by the beginning of time interval *i*.

Table 3.1. Indices, input parameters, and decision variables in model TSCSP

Indices	
s	Stack (i.e. column) (s = 1, 2, …, S).
t	Tier (t = 1, 2, …, T).
i	Time interval (i = 1, 2, …, I, I+1).
p, q	Spreader type (p, q = 1, 2, or 3) (1 = single spreader; 2 = dual spreader; 3 = triple spreader).
Input parameters	
S	Number of stacks in the storage bay. $S \geq 3$ to avoid triviality.
T	Number of tiers in the storage bay.
I	Number of time intervals available (= $S \times T$ to be conservative).
E_s	Initial number of containers in stack s (integer, ≥ 0) (s = 1, 2, ..., S).
C_{pq}	Changeover time from spreader type p to spreader type q (minutes) (real, ≥ 0) (p, q = 1, 2, or 3).
H_p	Handling time per lift using spreader type p (minutes) (real, ≥ 0) (p = 1, 2, or 3).
$L2_{st}$	= 1 if the two containers occupying stacks s and s+1 in tier t can be lifted together using the dual spreader without violating the weight limit (binary) (s = 1, 2, …, S-1; t = 1, 2, …, T).
$L3_{st}$	= 1 if the three containers occupying stacks s, s+1, and s+2 in tier t can be lifted together using the triple spreader without violating the weight limit (binary) (s = 1, 2, …, S-2; t = 1, 2, …, T).
Decision variables	
X_{si}	= 1 if a single-spreader lift is performed at the top of stack s during time interval i (binary) (s = 1, 2, …, S; i = 1, 2, …, I).
Y_{si}	= 1 if a dual-spreader lift is performed in which the left (right) spreader lifts the container that is on the top of stack s (s+1) during time interval i (binary) (s = 1, 2, …, S-1; i = 1, 2, …, I).
Z_{si}	= 1 if a triple-spreader lift is performed in which the (left, center, right) spreader lifts the container that is on the top of stack (s, s+1, s+2) during time interval i (binary) (s = 1, 2, …, S-2; i = 1, 2, …, I).
G_{pqi}	= 1 if a changeover from spreader type p to spreader type q is made between time intervals i-1 and i (binary) (p, q = 1, 2, or 3; i = 2, 3, …, I).
F_i	= 1 if all containers have been removed from the bay by the beginning of time interval i (binary) (i = 1, 2, …, I+1).
N_{si}	Number of containers in stack s at the beginning of time interval i (integer, ≥ 0) (s = 1, 2, …, S; i = 1, 2, …, I+1).
R_{ti}	= 1 if any container(s) is removed from tier t during time interval i (binary) (t = 1, 2, …, T; i = 1, 2, …, I).

Objective function:

$$Minimize \quad \sum_{i=1}^{I}\left(\sum_{s=1}^{S} H_1 X_{si} + \sum_{s=1}^{S-1} H_2 Y_{si} + \sum_{s=1}^{S-2} H_3 Z_{si}\right) + \sum_{(p,q=1\ to\ 3: p\neq q)} \sum_{i=2}^{I} C_{pq} G_{pqi} \tag{1}$$

Subject to:

$$\sum_{s=1}^{S} X_{si} + \sum_{s=1}^{S-1} Y_{si} + \sum_{s=1}^{S-2} Z_{si} + F_i = 1 \qquad i = 1, 2, \ldots, I \tag{2}$$

$$\sum_{i=1}^{I+1} F_i \geq 1 \tag{3}$$

$$\left.\begin{aligned} \sum_{s=1}^{S} X_{s,i-1} + G_{12i} - 1 &\leq \sum_{s=1}^{S-1} Y_{si} \\ \sum_{s=1}^{S-1} Y_{si} + \sum_{s=1}^{S} X_{s,i-1} - 1 &\leq G_{12i} \\ G_{12i} + \sum_{s=1}^{S-1} Y_{si} - 1 &\leq \sum_{s=1}^{S} X_{s,i-1} \end{aligned}\right\} \qquad i = 2, 3, \ldots, I \tag{4a}$$

$$\left.\begin{aligned} \sum_{s=1}^{S-1} Y_{s,i-1} + G_{21i} - 1 &\leq \sum_{s=1}^{S} X_{si} \\ \sum_{s=1}^{S} X_{si} + \sum_{s=1}^{S-1} Y_{s,i-1} - 1 &\leq G_{21i} \\ G_{21i} + \sum_{s=1}^{S} X_{si} - 1 &\leq \sum_{s=1}^{S-1} Y_{s,i-1} \end{aligned}\right\} \qquad i = 2, 3, \ldots, I \tag{4b}$$

$$\left.\begin{aligned} \sum_{s=1}^{S} X_{s,i-1} + G_{13i} - 1 &\leq \sum_{s=1}^{S-2} Z_{si} \\ \sum_{s=1}^{S-2} Z_{si} + \sum_{s=1}^{S} X_{s,i-1} - 1 &\leq G_{13i} \\ G_{13i} + \sum_{s=1}^{S-2} Z_{si} - 1 &\leq \sum_{s=1}^{S} X_{s,i-1} \end{aligned}\right\} \qquad i = 2, 3, \ldots, I \tag{5a}$$

$$\left.\begin{aligned} \sum_{s=1}^{S-2} Z_{s,i-1} + G_{31i} - 1 &\leq \sum_{s=1}^{S} X_{si} \\ \sum_{s=1}^{S} X_{si} + \sum_{s=1}^{S-2} Z_{s,i-1} - 1 &\leq G_{31i} \\ G_{31i} + \sum_{s=1}^{S} X_{si} - 1 &\leq \sum_{s=1}^{S-2} Z_{s,i-1} \end{aligned}\right\} \qquad i = 2, 3, \ldots, I \tag{5b}$$

$$\left.\begin{aligned}
&\sum_{s=1}^{S-1} Y_{s,i-1} + G_{23i} - 1 \le \sum_{s=1}^{S-2} Z_{si} \\
&\sum_{s=1}^{S-2} Z_{si} + \sum_{s=1}^{S-1} Y_{s,i-1} - 1 \le G_{23i} \\
&G_{23i} + \sum_{s=1}^{S-2} Z_{si} - 1 \le \sum_{s=1}^{S-1} Y_{s,i-1}
\end{aligned}\right\} \qquad i = 2, 3, \ldots, I \qquad (6a)$$

$$\left.\begin{aligned}
&\sum_{s=1}^{S-2} Z_{s,i-1} + G_{32i} - 1 \le \sum_{s=1}^{S-1} Y_{si} \\
&\sum_{s=1}^{S-1} Y_{si} + \sum_{s=1}^{S-2} Z_{s,i-1} - 1 \le G_{32i} \\
&G_{32i} + \sum_{s=1}^{S-1} Y_{si} - 1 \le \sum_{s=1}^{S-2} Z_{s,i-1}
\end{aligned}\right\} \qquad i = 2, 3, \ldots, I \qquad (6b)$$

$$\sum_{(p,q=1\ to\ 3: p\ne q)} G_{pqi} \le 1 \qquad i = 2, 3, \ldots, I \qquad (7)$$

$$N_{si} \le T(1 - F_i) \qquad i = 1, 2, \ldots, I+1;\ s = 1, 2, \ldots, S \qquad (8)$$

$$N_{s1} = E_s \qquad s = 1, 2, \ldots, S \qquad (9)$$

$$N_{1,i+1} = N_{1i} - X_{1i} - Y_{1i} - Z_{1i} \qquad i = 1, 2, \ldots, I \qquad (10a)$$

$$N_{2,i+1} = N_{2i} - X_{2i} - Y_{1i} - Y_{2i} - Z_{1i} \qquad i = 1, 2, \ldots, I \text{ (only applies if } S = 3) \qquad (10b)$$

$$N_{2,i+1} = N_{2i} - X_{2i} - Y_{1i} - Y_{2i} - Z_{1i} - Z_{2i} \qquad i = 1, 2, \ldots, I \text{ (only applies if } S \ge 4) \qquad (10c)$$

$$N_{s,i+1} = N_{si} - X_{si} - Y_{s-1,i} - Y_{si} - Z_{s-2,i} - Z_{s-1,i} - Z_{si} \qquad i = 1, 2, \ldots, I;\ s = 3, 4, \ldots, S\text{-}2 \qquad (10d)$$

$$N_{S-1,i+1} = N_{S-1,i} - X_{S-1,i} - Y_{S-2,i} - Y_{S-1,i} - Z_{S-3,i} - Z_{S-2,i} \qquad i = 1, 2, \ldots, I \text{ (only applies if } S \ge 4) \qquad (10e)$$

$$N_{S,i+1} = N_{Si} - X_{Si} - Y_{S-1,i} - Z_{S-2,i} \qquad i = 1, 2, \ldots, I \qquad (10f)$$

$$X_{1i} + Y_{1i} + Z_{1i} \le N_{1i} \qquad i = 1, 2, \ldots, I \qquad (11a)$$

$$X_{2i} + Y_{1i} + Y_{2i} + Z_{1i} \le N_{2i} \qquad i = 1, 2, \ldots, I \text{ (only applies if } S = 3) \qquad (11b)$$

$$X_{2i} + Y_{1i} + Y_{2i} + Z_{1i} + Z_{2i} \le N_{2i} \qquad i = 1, 2, \ldots, I \text{ (only applies if } S \ge 4) \qquad (11c)$$

$$X_{si} + Y_{s-1,i} + Y_{si} + Z_{s-2,i} + Z_{s-1,i} + Z_{si} \le N_{si} \qquad i = 1, 2, \ldots, I;\ s = 3, 4, \ldots, S\text{-}2 \qquad (11d)$$

$$X_{S-1,i} + Y_{S-2,i} + Y_{S-1,i} + Z_{S-3,i} + Z_{S-2,i} \le N_{S-1,i} \qquad i = 1, 2, \ldots, I \text{ (only applies if } S \ge 4) \qquad (11e)$$

$$X_{Si} + Y_{S-1,i} + Z_{S-2,i} \le N_{Si} \qquad i = 1, 2, \ldots, I \qquad (11f)$$

$$\sum_{t=1}^{T} R_{ti} = 1 - F_i \qquad i = 1, 2, \ldots, I \tag{12}$$

$$-T(1-X_{si}) \le \left(\sum_{t=1}^{T} t * R_{ti}\right) - N_{si} \le T(1-X_{si}) \qquad i = 1, 2, \ldots, I;\ s = 1, 2, \ldots, S \tag{13a}$$

$$\left.\begin{aligned} -T(1-Y_{si}) &\le \left(\sum_{t=1}^{T} t * R_{ti}\right) - N_{si} \le T(1-Y_{si}) \\ -T(1-Y_{si}) &\le \left(\sum_{t=1}^{T} t * R_{ti}\right) - N_{s+1,i} \le T(1-Y_{si}) \end{aligned}\right\} \begin{aligned} &i = 1, 2, \ldots, I;\ s = 1, 2, \ldots, S\text{-}1 \\ &i = 1, 2, \ldots, I;\ s = 1, 2, \ldots, S\text{-}1 \end{aligned} \tag{13b}$$

$$\left.\begin{aligned} -T(1-Z_{si}) &\le \left(\sum_{t=1}^{T} t * R_{ti}\right) - N_{si} \le T(1-Z_{si}) \\ -T(1-Z_{si}) &\le \left(\sum_{t=1}^{T} t * R_{ti}\right) - N_{s+1,i} \le T(1-Z_{si}) \\ -T(1-Z_{si}) &\le \left(\sum_{t=1}^{T} t * R_{ti}\right) - N_{s+2,i} \le T(1-Z_{si}) \end{aligned}\right\} \begin{aligned} &i = 1, 2, \ldots, I;\ s = 1, 2, \ldots, S\text{-}2 \\ &i = 1, 2, \ldots, I;\ s = 1, 2, \ldots, S\text{-}2 \\ &i = 1, 2, \ldots, I;\ s = 1, 2, \ldots, S\text{-}2 \end{aligned} \tag{13c}$$

$$Y_{si} + R_{ti} - 1 \le L2_{st} \qquad i = 1, 2, \ldots, I;\ s = 1, 2, \ldots, S\text{-}1;\ t = 1, 2, \ldots, T \tag{14a}$$

$$Z_{si} + R_{ti} - 1 \le L3_{st} \qquad i = 1, 2, \ldots, I;\ s = 1, 2, \ldots, S\text{-}2;\ t = 1, 2, \ldots, T \tag{14b}$$

$$\sum_{s=1}^{S} X_{si} + \sum_{s=1}^{S-1} Y_{si} + \sum_{s=1}^{S-2} Z_{si} \ge \sum_{s=1}^{S} X_{s,i+1} + \sum_{s=1}^{S-1} Y_{s,i+1} + \sum_{s=1}^{S-2} Z_{s,i+1} \qquad i = 1, 2, \ldots, I\text{-}1 \tag{15}$$

The objective function (1) minimizes the makespan, *M*, which is the sum of the container handling and spreader changeover times. Constraint (2) ensures that at most one lift is performed during each time interval *i*; if no lift is made, then the "finished" binary variable F_i should equal 1. Constraint (3) ensures that the process of removing containers from the bay is finished by the end of the last time interval. Constraint (4a) forces a changeover to happen when switching from single-spreader to dual-spreader mode. This constraint has three expressions with the following structure: A + B − 1 ≤ C; C + A − 1 ≤ B; and B + C − 1 ≤ A. These expressions ensure that if any two of the binary terms A, B, and C equal 1, then the third term equals 1. Term A indicates if a

single-spreader lift is made during time interval $i-1$; B indicates if a changeover from single-spreader to dual-spreader mode is made between time intervals $i-1$ and i; and C indicates if a dual-spreader lift is made during time interval i. Constraint (4b) is the same as (4a) except that it considers the switch from dual-spreader to single-spreader mode. Constraints (5a-5b) work just like (4a-4b) but instead consider the switch between single-spreader and triple-spreader mode. Constraints (6a-6b) work just like (4a-4b) but instead consider the switch between dual-spreader and triple-spreader mode. Constraint (7) ensures that at most one changeover is performed between any two time intervals.

Constraint (8) sets $F_i = 0$ when any $N_{si} \neq 0$, i.e. when at least one stack is non-empty. Constraint (9) initializes the stack heights at the start of the first time interval. Constraints (10a-10f) update the stack heights based on the lifts made during each time interval. Constraints (11a-11f) ensure that no lift is made from an empty stack. Constraint (12) ensures that containers are not removed from any tier during any time interval after the lifting is finished. Also, while the lifting is not finished, containers are removed from exactly one tier during each time interval. Constraint (13a) enforces the physical limitation that a single-spreader lift can only be made from the top tier of a stack. This "sandwich" constraint is of the form $Left \leq Middle \leq Right$. If $X_{si} = 0$, *Left* is very negative and *Right* is very positive, so there is no meaningful constraint on *Middle*. If $X_{si} = 1$, $Left = Right = 0$, so *Middle* must equal 0, i.e. the tier from which containers are removed must equal the height of the stack from which containers are removed (N_{si}). Constraint (13b), similar to (13a), enforces the physical limitation that a dual-spreader lift can only be made from the top tier of two adjacent stacks. This constraint also requires the heights of these two adjacent stacks to be equal. Constraint (13c), similar to (13b), enforces the physical limitation that (i) a triple-spreader lift can only be made from the top tier of three adjacent stacks and (ii) the heights

of these three adjacent stacks must be equal. Constraints (14a-14b) ensure that dual- and triple-spreader lifts do not violate the weight-limit-respecting binary values in matrices $L2$ and $L3$ respectively. Finally, constraint (15) ensures that no lifts are made during time interval i+1 if no lifts are made during interval i. In other words, the crane can never transition from an idle state to a non-idle state. This constraint is redundant but helps the solver find an optimal solution more quickly.

3.3 Genetic algorithm (GA)

We developed a genetic algorithm for attacking large instances of the TSCSP. A genetic algorithm (GA) is one the most commonly used metaheuristic techniques. Like other metaheuristics, it sacrifices optimality for quicker and more efficient results. GAs are easy to implement and have lots of potential to be applied to different problem types and optimization problems with nonlinear objective functions and/or constraints (Ezugwu et al. 2020). GAs can be faster than other algorithms if the implementation is done correctly and efficiently. In the TSCSP, the chromosome representation we use is a great match for GA. Mutation and crossover operations can be done without the concern of infeasibility. However, GAs have some limitations such as getting stuck in a local optimum (Fernández 2018). Since GAs don't have a termination criterion, we use a time limit to terminate our GA.

In the GA we developed for the TSCSP, each feasible solution is represented by a chromosome consisting of a sequence of T genes ($g_1, g_2, g_3, \ldots, g_T$)—where g_t is the gene for tier t. The gene for tier t specifies how each container in tier t is handled. In other words, g_t specifies which containers are single-, dual-, and triple-spreader lifted in tier t. The fitness (i.e. objective value, makespan) of a chromosome is computed by (a) forming a feasible crane lift sequence that

agrees with its genes and (b) evaluating the makespan of this sequence. The setup and procedure of our GA is summarized in Tables 3.2, 3.3, and 3.4 and Figures 3.3, 3.4, and 3.5. Table 3.2 and Figures 3.3 and 3.4 show how the genes and chromosomes are constructed. Tables 3.3 and 3.4 show the two methods we use for evaluating chromosome fitness. Figure 3.5 shows the overall GA procedure. A detailed discussion of the GA now follows.

3.3.1 Tier options

Significant preliminary work is performed before the GA procedure commences. This work consists of creating a predefined set of lifting options (i.e. *tier options*, options, genes) for each tier of the instance at hand. A lifting option for tier t specifies which containers are single-, dual-, and triple-spreader lifted in tier t. The options for tier t, for each t from 1 to T, are created by solving variations of the *tier t sub-problem*, i.e. the single-tier TSCSP formed by considering only the containers in tier t in the original TSCSP and ignoring spreader changeover costs.

Table 3.2 shows our math model—model TSCSP-Sub—of the tier t sub-problem. The objective function (16) minimizes total container handling time. Constraints (17-22) ensure that (a) exactly one lift is made in stack s if there is a container in stack s and (b) no lift is made in stack s if there is no container in stack s. Constraints (23-24) ensure that the dual- and triple-spreader lifts do not violate the binary values in matrices $L2$ and $L3$ respectively. An optimal solution to model TSCSP-Sub can be obtained within a few seconds using standard integer programming software when $S \leq 50$.

Table 3.2. Indices, input parameters, and decision variables in math model TSCSP-Sub.

Indices	
s	Stack (s = 1, 2, …, S).
p	Spreader type (p = 1, 2, or 3) (1 = single spreader; 2 = dual spreader; 3 = triple spreader).
Input parameters	
t	Tier under consideration.
S	Number of stacks in the storage bay.
E_s	= 1 if there is a container in stack s and tier t in the original TSCSP instance (binary) (s = 1, 2, ..., S).
H_p	Handling time per lift using spreader type p (minutes) (real, ≥ 0) (p = 1, 2, or 3).
$L2_{st}$	= 1 if the two containers occupying stacks s and s+1 in tier t in the original TSCSP instance can be lifted together in dual-spreader mode without violating the weight limit (binary) (s = 1, 2, …, S-1).
$L3_{st}$	= 1 if the three containers occupying stacks s, s+1, and s+2 in tier t in the original TSCSP instance can be lifted together in triple-spreader mode without violating the weight limit (binary) (s = 1, 2, …, S-2).
Decision Variables	
X_s	= 1 if a single-spreader lift is performed in stack s (binary) (s = 1, 2, …, S).
Y_s	= 1 if a dual-spreader lift is performed in which the left (right) spreader lifts the container in stack s (s+1) (binary) (s = 1, 2, …, S-1).
Z_s	= 1 if a triple-spreader lift is performed in which the (left, center, right) spreader lifts the container in stack (s, s+1, s+2) (binary) (s = 1, 2, …, S-2).

Objective:

Minimize: $\sum_{s=1}^{S} H_1 X_s + \sum_{s=1}^{S-1} H_2 Y_s + \sum_{s=1}^{S-2} H_3 Z_s$ (16)

Constraints:

$$X_1 + Y_1 + Z_1 = E_1 \quad (17)$$

$$X_2 + Y_1 + Y_2 + Z_1 = E_2 \quad (only\ applies\ if\ S = 3) \quad (18)$$

$$X_2 + Y_1 + Y_2 + Z_1 + Z_2 = E_2 \quad (only\ applies\ if\ S \geq 4) \quad (19)$$

$$X_s + Y_{s\_1} + Y_s + Z_{s\_2} + Z_{s\_1} + Z_s = E_s \quad for\ s = 3, 4, \ldots, S-2. \tag{20}$$

$$X_{S\_1} + Y_{S\_2} + Y_{S\_1} + Z_{S\_3} + Z_{S\_2} = E_{S\_1} \quad (only\ applies\ if\ S \geq 4) \tag{21}$$

$$X_S + Y_{S\_1} + Z_{S\_2} = E_S \tag{22}$$

$$Y_s \leq L2_{St} \quad for\ s = 1, 2, \ldots, S-1 \tag{23}$$

$$Z_s \leq L3_{St} \quad for\ s = 1, 2, \ldots, S-2 \tag{24}$$

The lifting options we create are divided into four *categories*: options in which (1) at least one dual-spreader and at least one triple-spreader lift is made; (2) at least one dual-spreader lift but no triple spreader lift is made; (3) at least one triple-spreader lift but no dual-spreader lift is made; and (4) no dual spreader or triple-spreader lifts are made in that tier. The options in category (1, 2, 3, 4) are created by solving math program TSCSP-Sub (Table 3) with the additional constraints (b+d, b+c, a+d, a+c) shown in equation (25) below:

$$\sum_{s=1}^{S-1} Y_s = 0 \ \text{(a)}; \quad \sum_{s=1}^{S-1} Y_s \geq 1 \ \text{(b)}; \quad \sum_{s=1}^{S-2} Z_s = 0 \ \text{(c)}; \quad \sum_{s=1}^{S-2} Z_s \geq 1 \ \text{(d)} \tag{25}.$$

For each combination of category and tier, we identify the *Max#OptionsPerCategoryPerTier* best lifting options—having the lowest makespans—by repeatedly solving math program TSCSP-Sub and adding the constraint "sum of (the X_s, Y_s, and Z_s variables that equaled one in the previous optimal solution) ≤ (the number of X_s, Y_s, and Z_s variables that equaled one in the previous optimal solution) – 1" to the model after each new optimal solution is found. Spreader changeover costs are ignored when the makespan is computed. Fewer than *Max#OptionsPerCategoryPerTier* options are constructed for a given tier and category if the number of feasible solutions is less than this value.

The left half of Figure 3.3 shows a partial list of the feasible tier option for each tier for the problem instance from Figures 3.1 and 3.2. The options for each tier *t* have been categorized and numbered from 1 to *numOptions(t)* where *numOptions(t)* is the total number of options created for tier *t*. Note that a total of (24, 44, and 44) options have been created for tier (1, 2, 3). In tier 1, (6, 12, 5, 1) options have been created for category (1, 2, 3, 4). In tier 2, (15, 20, 8, 1) options have been created for category (1, 2, 3, 4). The number of options created for each category in tier 3 equals that in tier 2 because $L2$ and $L3$ have identical values in these two tiers (Figure 3.2). Within each category, the tier options are ordered from best to worst. For example, tier 1 option 7 has a lower makespan than tier 1 option 8, and tier 1 option 19 has a lower makespan than tier 1 option 20. Note that the number of feasible options in category 1, 2, or 3 for a given tier could range from 0 (if no feasible options exist) to millions (if *S* is large). However, there is always exactly one feasible option in category 4 for each tier.

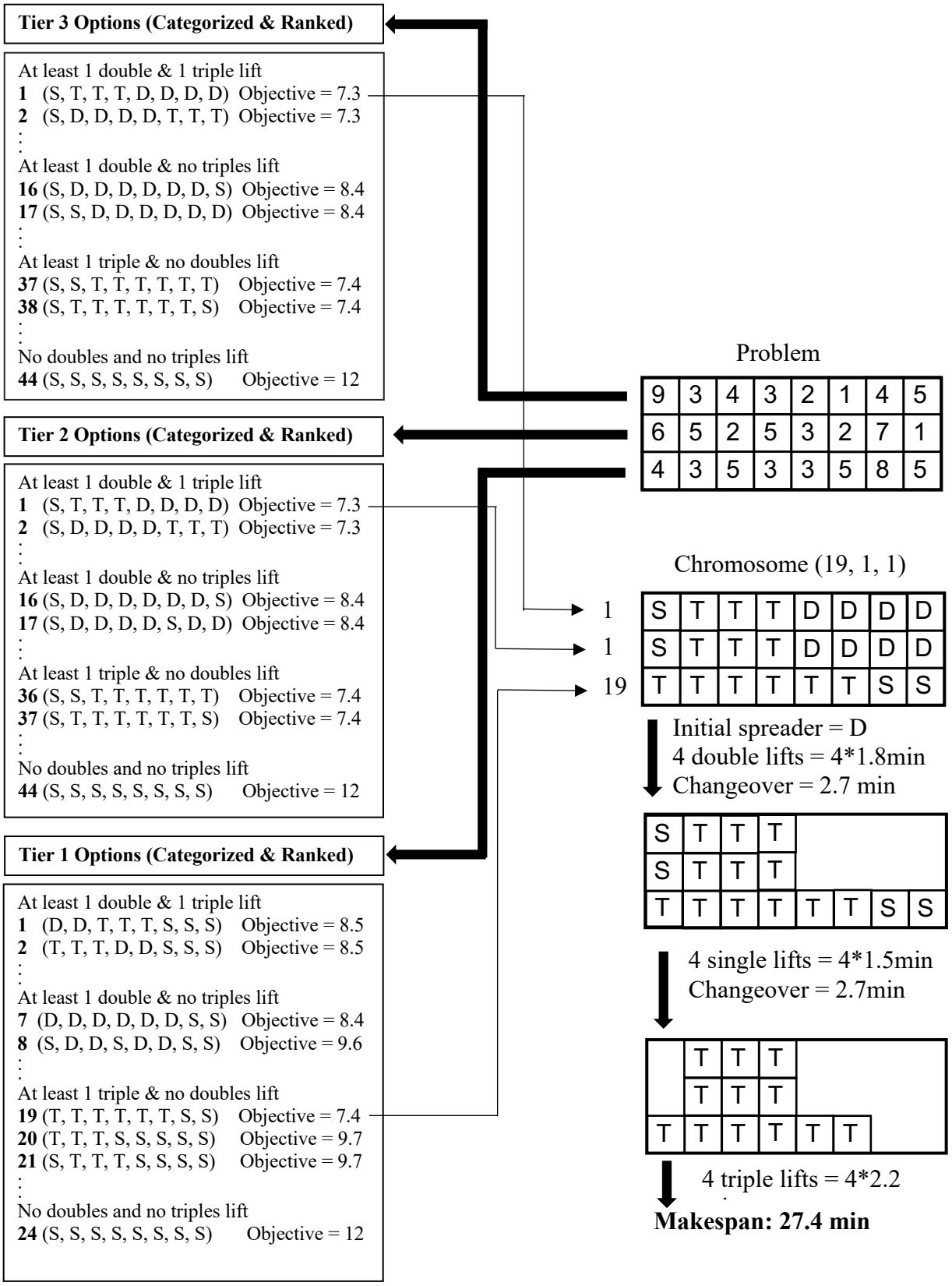

Figure 3.3. Example 1 of GA chromosome formation and objective value computation.

3.3.2 Chromosome composition and fitness computation

A chromosome ($g_1, g_2, g_3, \ldots, g_T$) consists of a sequence of T tier options or *genes*—one for each tier—where g_t is the tier option number used for tier t. The middle-right portion of Figure 3.3 shows chromosome (19, 1, 1) for the instance at hand. Here, option 19 is used for tier 1; option 1 is used for tier 2; and option 1 is used for tier 3. This is a *greedy* chromosome because it combines the best (i.e. least cost) options for the individual tiers (ignoring changeover costs). Note that this would be an optimal solution if all changeover costs were zero. The middle-right portion of Figure 3.4 shows chromosome (21, 37, 37) for the same instance. Here, option 21 is used for tier 1; option 37 is used for tier 2; and option 37 is used for tier 3.

Chromosome fitness is computed by (a) finding a feasible crane lift sequence that agrees with the lift type of each container—single-spreader, dual-spreader, or triple-spreader—specified by the chromosome and then (b) evaluating the makespan of this sequence. To eliminate dominated solutions, we require in each feasible crane lift sequence that the use of a given spreader mode continues until no more containers matching that spreader mode can be feasibly lifted from the top of any stack.

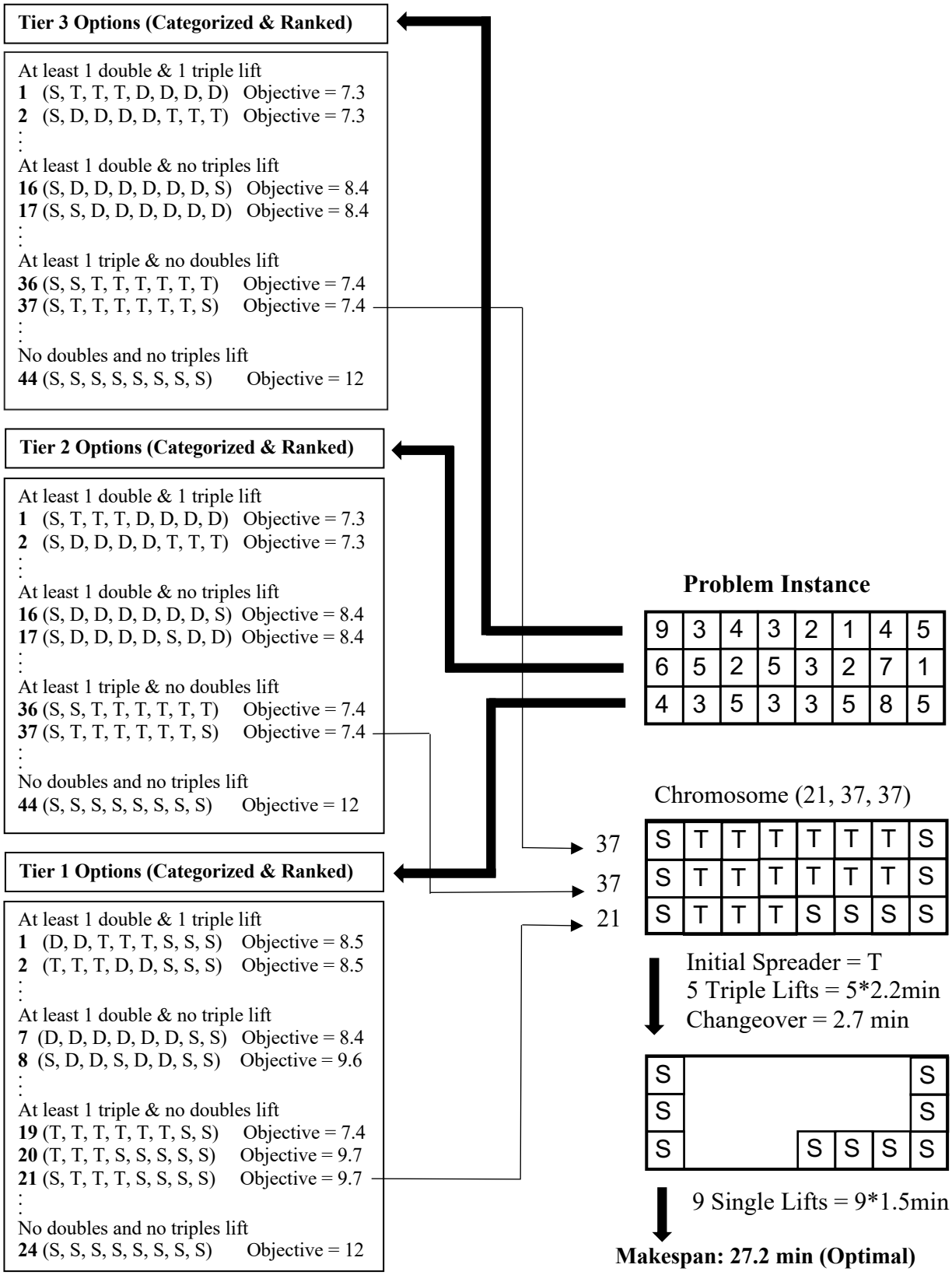

Figure 3.4. Example 2 of GA chromosome formation and objective value computation.

Note that the makespan equals (i) total handling time plus (ii) total spreader changeover time. Item (i) is already given by the chromosome, so the challenge is to compute (ii). Note that a crane lift sequence for a given chromosome can be summarized by a spreader mode sequence M_1-M_2-M_3- … where M_j is the spreader mode used during phase j of the lifting. The value of M_j is (S, D, T) when the crane operates in (single-spreader, dual-spreader, triple-spreader) mode respectively. For example, the crane lift sequence shown beneath chromosome (19, 1, 1) in Figure 3.3 is summarized by the spreader mode sequence D-S-T. According to this sequence, the crane first operates in dual-spreader mode, then in single-spreader mode, and then in triple-spreader mode. In each mode, the crane lifts as many containers as possible from the tops of the stacks without violating the chromosome values and the need to keep containers that are part of the same multi-spreader lift together. If C_{pq}= 2.7 for all p and q, then sequence D-S-T is the most efficient spreader mode sequence for this chromosome because it empties the bay after only three phases (i.e. two changeovers). No other spreader mode sequence is this efficient. On the other hand, T-S-T-D-T-S is an inefficient spreader mode sequence for this chromosome. This sequence has six phases (i.e. five changeovers): two triple lifts are performed, then two single lifts, then one triple lift, then four dual lifts, then one triple lift, and then two single lifts.

The above discussion shows that, for each chromosome, care must be taken to find a good spreader mode sequence in which the total changeover time is minimized to the extent possible. Keeping this in mind, we developed two methods—a branch-and-bound method and greedy method—to compute the total changeover time associated with a given chromosome.

Table 3.3 shows the logic in the branch-and-bound method. This method identifies an optimal spreader mode sequence upon termination. The method works by building up spreader mode *subsequences* one phase at a time. It begins with three subsequences, each with a single

phase: S, D, and T (line 5). Only subsequences that are feasible—having at least one lift performed during each phase of the subsequence—are extended to form longer subsequences. Each feasible subsequence is branched, i.e. extended in two ways, corresponding to the two spreader modes that can theoretically follow the mode which ends the subsequence. Eventually, one subsequence will be finished, i.e. will result in an empty bay, and the total changeover time for this subsequence is stored in the variable *minChangeoverCost* (line 25). This value establishes an upper bound which is used to eliminate any unfinished subsequence *s* from consideration whose associated total changeover costs is already greater than *minChangeoverCost*. Every time a new subsequence is finished, *minChangeoverCost* is updated if necessary (lines 23-25). The process continues until there are no more unfinished subsequences *s* such that costs is less than *minChangeoverCost*. Upon termination, *minChangeoverCost* is the optimal total changeover time. Lines 9-22 in Table 3.3 show the procedure for computing the total changeover time of each spreader mode subsequence and identifying which subsequences are feasible and/or finished. Lines 26-41 show the procedure for branching each feasible, unfinished, *N*-phase subsequence into two (*N*+1)-phase subsequences. The fitness value of a chromosome equals (i) the total handling time plus (ii) the final value of *minChangeoverCost*.

Table 3.3. Branch-and-bound method for computing the total spreader changeover cost of a chromosome.

Convert chromosome ($g_1, g_2, \ldots, g_T$) into *ContainerArray*, an $S \times T$ array that indicates the spreader mode used to lift each container ("1" indicates single-spreader mode, "2" indicates dual-spreader mode, "3" indicates triple-spreader mode.
Let S = 3 = number of spreader mode sequences currently being investigated.
Let N = 1 = length of each sequence currently being investigated (i.e. the number of spreader phases in each subsequence).
Let M_{sn} = spreader mode used (1, 2, or 3) during n^{th} phase of subsequence s (for all s = 1 to S and all n = 1 to N).
Initially M_{11} = 1, M_{21} = 2 and M_{31} = 3.
Let *minChangeoverCost* = ∞
Let *done* = true
while *done* = false **do**
    for s = 1 to S **do**
        Restore *ContainerArray* to its original form
        Let $finished_s$ = false, $feasible_s$ = true, and $cost_s$ = 0.
        Let n = 1
        while $n \leq N$ and $feasible_s$ = true **do**
            if $n \geq 2$ **then**
                $cost_s = cost_s + C_{(M_{s,n-1}),(M_{sn})}$
            Lift as many containers as possible from top of the $ContainerArray$ using spreader M_{sn}, ensuring that containers that are supposed to be part of the same multi-spreader lift are kept together.
            Let $lift_{sn}$ = the number of lifts made.
            if $lift_{sn}$ = 0 **then**
                $feasible_s$ = false
            else if there are no more containers in *ContainerArray* **then**
                $finished_s$ = true
            $n = n$+1
    for s = 1 to S **do**
        if $finished_s$ = true and $cost_s < minChangeoverCost$ **then**
            $minChangeoverCost = cost_s$
    done = true
    Let p = 1
    for s = 1 to S **do**
        if $finished_s$ = false and $feasible_s$ = true and $cost_s < minChangeoverCost$ **then**
            done = false
            for n = 1 to N **do**
                $Q_{pn} = Q_{p+1,n} = M_{sn}$
            if M_{sN} = 1 **then** $Q_{p,N+1}$ = 2 and $Q_{p+1,N+1}$ = 3
            else if M_{sN} = 2 **then** $Q_{p,N+1}$ = 1 and $Q_{p+1,N+1}$ = 3
            else if M_{sN} = 3 **then** $Q_{p,N+1}$ = 1 and $Q_{p+1,N+1}$ = 2
            $p = p + 2$
    $N = N$+1
    $S = p$-1
    for s = 1 to S **do**
        for n = 1 to N **do**
            $M_{sn} = Q_{sn}$
output *minChangeoverCost*

Table 3.4 shows the logic in the greedy method. This method constructs a spreader mode sequence one phase at a time. The spreader mode used in the next phase is the mode that can lift the greatest number of containers from the bay. This method is not guaranteed to find the minimum total changeover time but uses much less computation time than the branch-and-bound method. In this method, the fitness value of a chromosome equals (i) the total handling time plus (ii) the final value of *changeoverCost* (lines 3, 35).

Table 3.4. Greedy method for computing the total spreader changeover cost of a chromosome.

```
1  Convert chromosome (g1, g2, ..., gT) into ContainerArray, an S × T array that indicates the spreader
     mode used to lift each container ("1" indicates single-spreader mode, "2" indicates dual-spreader mode,
     "3" indicates triple-spreader mode.
2  Let currMode ∈{1,2,3} be the spreader mode that can lift the greatest number of containers from the
     top of ContainerArray. Ties are broken arbitrarily.
3  Let changeoverCost = 0
4  while there's at least one "1", "2" or "3" in ContainerArray (i.e. while at least one container is still
       present)
5      Lift as many unblocked containers as possible from the top of ContainerArray using spreader mode
         currMode
6      if there are no more containers in ContainerArray
7          Go to line 35
8      if currMode = 1
9          Let N_D = number of containers that can now be lifted from the top of ContainerArray using
             spreader mode 2
10         Let N_T = number of containers that can now be lifted from the top of ContainerArray using
             spreader mode 3
11         if N_D ≥ N_T then
12             currMode = 2
13             changeoverCost = changeoverCost + C12
14         else
15             currMode = 3
16             changeoverCost = changeoverCost + C13
17     else if currMode = 2
18         Let N_S = number of containers that can now be lifted from the top of ContainerArray using
             spreader mode 1
19         Let N_T = number of containers that can now be lifted from the top of ContainerArray using
             spreader mode 3
20         if N_S ≥ N_T then
21             currMode = 1
22             changeoverCost = changeoverCost + C21
23         else
24             currMode = 3
25             changeoverCost = changeoverCost + C23
26     else if currMode = 3
27         Let N_S = number of containers that can now be lifted from the top of ContainerArray using
             spreader mode 1
28         Let N_D = number of containers that can now be lifted from the top of ContainerArray using
             spreader mode 2
29         if N_S ≥ N_D then
30             currMode = 1
31             changeoverCost = changeoverCost + C31
32         else
33             currMode = 2
34             changeoverCost = changeoverCost + C32
35 output changeoverCost
```

The bottom-right portion of Figures 3.3 and 3.4 show the fitness evaluation of chromosomes (19, 1, 1) and (21, 37, 37) respectively. In Figure 3.3, the branch-and-bound method is used; this method yields a spreader mode sequence of D-S-T which produces a total makespan of 27.4 minutes. The greedy method applied to this chromosome would have resulted in a spreader mode sequence of D-T-S-T and a makespan of 30.1 minutes. In Figure 3.4, both fitness evaluation methods result in a makespan of 27.2 minutes. Note that the chromosome in Figure 3.4 has a better makespan than the one in Figure 3.3 even though its tier options are each inferior to those in Figure 3.3. Chromosome (21, 37, 37) achieves the optimal makespan for this instance: 27.2 minutes.

3.3.3 GA procedure

Figure 3.5 shows the overall GA procedure. The first step is to use the method from Section 3.3.1 to generate the options (i.e. genes) for each tier t from 1 to T. Each option in tier t is numbered from 1 to $numOptions(t)$ where $numOptions(t)$ is the total number of options created for tier t (see left side of Figures 3.3 and 3.4). The second step is to build N chromosomes in the first generation. In each such chromosome, the gene for tier t is a random integer from 1 to $numOptions(t)$.

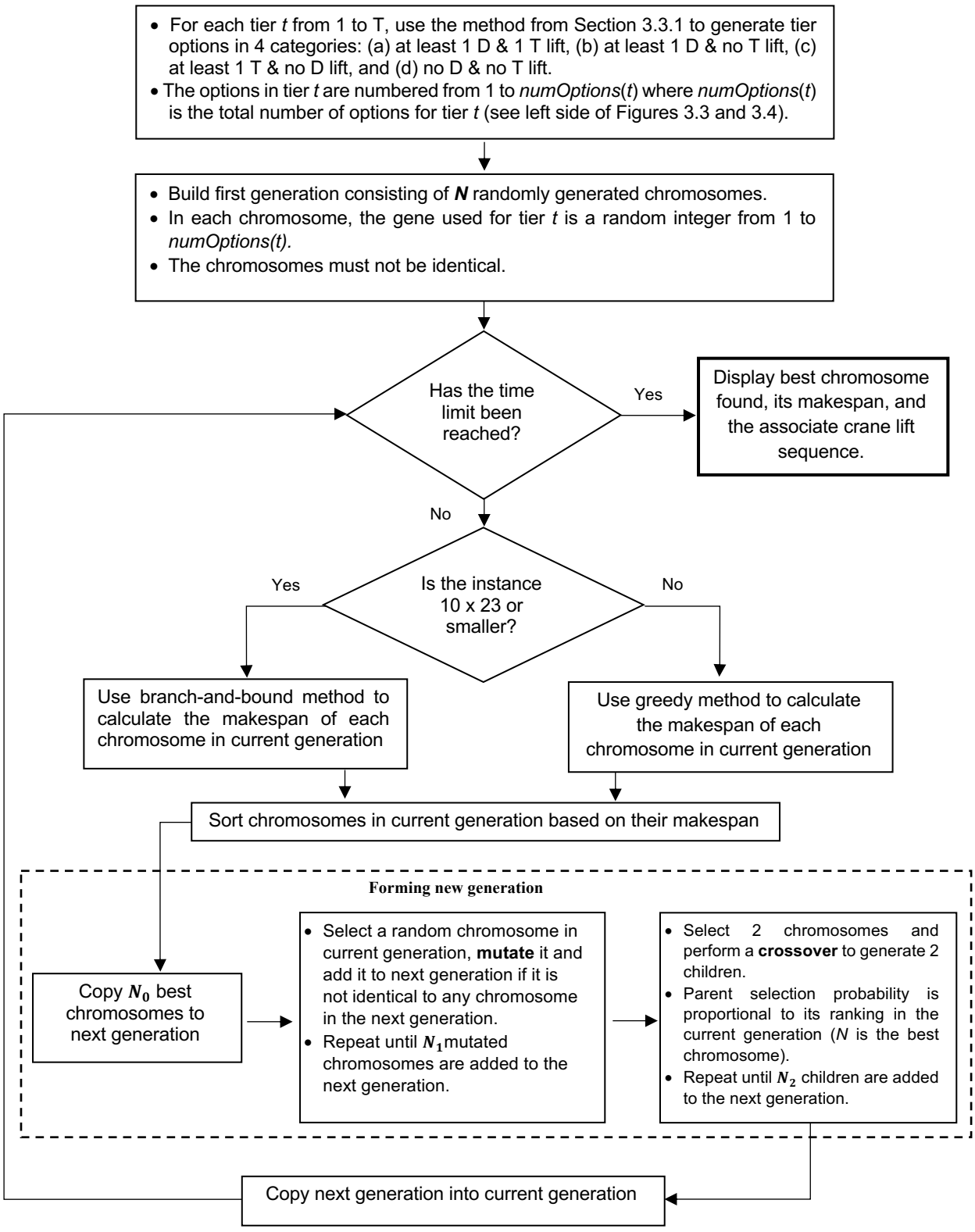

Figure 3.5. Overall logic of genetic algorithm.

The iterative portion of the GA commences immediately after the first generation of chromosomes is formed. In step A of each iteration, the makespan (i.e. fitness) of each chromosome in the current generation is evaluated. This is done using the greedy (branch-and-bound) method from Section 3.3.2 if the initial storage bay has more (no more) than 230 containers. In step B, the chromosomes are sorted based on their makespans.

In step C, the N_0 best chromosomes in the current generation are copied into the next generation. In step D, N_1 mutated chromosomes are created and added to the next generation. Each mutated chromosome is formed by selecting a random chromosome in the current generation and then, for each t from 1 to T, changing its tier t gene to a new, random tier t gene with probability *mProb*. In step E, parent chromosomes from the current generation are mated, and a total of N_2 (= $N - N_0 - N_1$) children are added to the next generation. In each crossover operation, two parent chromosomes ($g1_1$, $g1_2$, $g1_3$, …, $g1_T$) and ($g2_1$, $g2_2$, $g2_3$, …, $g2_T$) are mated—forming two children—by performing a crossover operation at a random position p ($1 \leq p \leq T$-1) in the parent chromosomes. Random variable p follows a discrete uniform distribution with minimum value 1 and maximum value T-1. The resulting children are ($g1_1$, …, $g1_p$, $g2_{(p+1)}$, …, $g2_T$) and ($g2_1$, …, $g2_p$, $g1_{(p+1)}$, …, $g1_T$). Each parent's selection probability is proportional to its fitness ranking in current generation where the chromosome in the current generation with the lowest (highest) makespan (ties are broken randomly) has ranking N (1). Unless it is impossible to do so (i.e. unless there are fewer than N unique chromosomes), we require that every chromosome in each generation be unique. When a predefined time limit is reached, the GA procedure terminates and the best chromosome that has been found—and its corresponding makespan and crane lift sequence—is displayed.

3.4 Lower bound computation

We compute a lower bound on the TSCSP's optimal objective value as follows. First, we use math model TSCSP-Sub (Table 3.2) to find the best option in each category for each tier t from 1 to T (Section 3.3.1). The best option in category (1, 2, 3, 4) is created by solving math program TSCSP-Sub (Table 3.2) with the additional constraints (b+d, b+c, a+d, a+c) shown in (25). Let Sub_{tc} be the optimal objective value for problem TSCSP-Sub when tier t category c is considered. We assume this is $+\infty$ if the problem is infeasible. Also, let $N1_{tc}$ be the total number of single-spreader lifts made in the optimal solution to problem TSCSP-Sub when tier t category c is considered. This variable is undefined if the problem is infeasible.

We use Sub_{tc} and $N1_{tc}$ to compute four lower bounds—LB_1, LB_2, LB_3, LB_4—where LB_y is a lower bound on the best makespan among the *type y* feasible solutions (i.e. crane lift sequences). A type 1 feasible solution has at least one dual-spreader and at least one triple-spreader lift. A type 2 (3) feasible solution has at least one dual-spreader (triple-spreader) lift but no triple-spreader (dual-spreader) lifts. A type 4 feasible solution has no dual-spreader and no triple-spreader lifts. These solution types encompass all feasible solutions, so $LB = min\{LB_1, LB_2, LB_3, LB_4\}$ is a lower bound on the optimal value of the TSCSP.

The values LB_1, LB_2, LB_3, LB_4 are computed as follows:

$$LB_1 = \sum_{t=1}^{T}\left(\min_c \{Sub_{tc}\}\right) + \min\{C_{12}+C_{23},\ C_{13}+C_{32},\ C_{21}+C_{13},\ C_{23}+C_{31},\ C_{31}+C_{12},\ C_{32}+C_{21}\}$$

if $N1_{tc} > 0$ for any (t,c) $(t = 1\ldots T,\ c = 1\ldots 4)$ in which $Sub_{tc} = \min\{Sub_{t1}, Sub_{t2}, Sub_{t3}, Sub_{t4}\}$ (26a)

$$= \sum_{t=1}^{T}\left(\min_c \{Sub_{tc}\}\right) + \min\{C_{23},\ C_{32}\}$$

if $N1_{tc} = 0$ for all (t,c) $(t = 1\ldots T,\ c = 1\ldots 4)$ in which $Sub_{tc} = \min\{Sub_{t1}, Sub_{t2}, Sub_{t3}, Sub_{t4}\}$ (26b)

$$LB_2 = \sum_{t=1}^{T}\left(\min\{Sub_{t2}, Sub_{t4}\}\right) + \min\{C_{12}, C_{21}\} \quad \text{if } N1_{t2} > 0 \text{ for any } t \text{ from 1 to } T \tag{27a}$$

$$= \sum_{t=1}^{T} Sub_{t2} \quad \text{if } N1_{t2} = 0 \text{ for all } t \text{ from 1 to } T \tag{27b}$$

$$LB_3 = \sum_{t=1}^{T}\left(\min\{Sub_{t3}, Sub_{t4}\}\right) + \min\{C_{13}, C_{31}\} \quad \text{if } N1_{t3} > 0 \text{ for any } t \text{ from 1 to } T \tag{28a}$$

$$= \sum_{t=1}^{T} Sub_{t3} \quad \text{if } N1_{t3} = 0 \text{ for all } t \text{ from 1 to } T \tag{28b}$$

$$LB_4 = \sum_{t=1}^{T} Sub_{t4} \tag{29}$$

Equation (26a) refers to the case in which all three spreader modes are used. Equation (26b) refers to the case in which dual-spreader and triple-spreader modes are used but single-spreader mode is not used. Equation (27a) refers to the case in which single-spreader and dual-spreader modes are used but triple-spreader mode is not used. Equation (27b) refers to the case in which only dual-spreader mode is used. Equation (28a) refers to the case in which single-spreader and triple-spreader modes are used but dual-spreader mode is not used. Equation (28b) refers to the case in which only triple-spreader mode is used. Equation (29) refers to the case in which only single-spreader mode is used. The above equations assume (i) the bare minimum number of changeovers—2, 1, or 0—needed to transition between the spreader modes that are used and (ii) the most efficient spreader mode sequence among the theoretical alternatives. For the instance shown in Figures 3.1, 3.2, 3.3, and 3.4 (LB_1, LB_2, LB_3, LB_4) = (27.4, 27.9, 24.9, 36), so LB = 24.9.

3.5 Experimental setup, results, and discussion

The lower bound procedure from Section 3.4, genetic algorithm (GA) from Section 3.3, and model TSCSP from Section 3.2 were coded into MS Visual C++ 2010 Professional. IBM

ILOG Concert Technology was used to define model TSCSP within C++ and call the MILP solver IBM ILOG CPLEX 12.5 to solve instances defined in text files. To avoid running out of memory, the CPLEX "node file storage parameter" was set to 3. Otherwise, default CPLEX settings were used. All results were obtained on a desktop computer with the Windows 7 Enterprise 64-bit operating system, an Intel Core i7-4770 processor with eight 3.4 gigahertz cores, and 16 GB of RAM.

We perform three experiments. In Experiment 1, we test the math model and GA on 120 instances of the TSCSP. In Experiment 2, we compare the performance of the GA to that of the simulated annealing-heuristic introduced in Chapter 2 on the 120 instances of the dual-spreader crane scheduling problem (DSCSP) considered in Chapter 2. In experiment 3, we use dynamic programming to create tier options for the GA and then re-test the GA on the 120 instances of the TSCSP considered in experiment 1.

3.5.1 Experiment 1

In this experiment we consider a total of 120 problem instances—30 instances for each of the problem sizes 3×8, 5×10, 10×23, and 50×50. Problem size T×S has T tiers, S stacks, and T containers in stack s at time 0 for all s. In all instances, we assume that the container weights W_{st} take integer values from 1 to 9. We also assume that the weight limits are L_2 = 10 for dual-spreader lifts and L_3 = 12 for triple-spreader lifts. We also assume that H_1 = 1.5, H_2 = 1.8, H_3 = 2.2 and C_{pq} = 2.7 for all p and q. Among the 30 instances for each problem size, (10, 10, 10) instances have (light, medium, heavy) container weights. In the medium instances, the weight of each container follows a discrete uniform distribution over the values {1, 2, 3, 4, 5, 6, 7, 8, 9}. In the light instances, the weight of each container has a {15%, 15%, 15%, 15%, 20%, 5%, 5%, 5%,

5%} chance of taking the value {1, 2, 3, 4, 5, 6, 7, 8, 9}. In the heavy instances, the weight of each container has a {5%, 5%, 5%, 5%, 20%, 15%, 15%, 15%, 15%} chance of taking the value {1, 2, 3, 4, 5, 6, 7, 8, 9}.

Table 3.5 shows the GA parameter settings used in this experiment. These settings were chosen based on preliminary experiments whose results are not shown here. Note that more computation time is allocated for attacking larger problems. A nontrivial portion of this time is spent solving instances of math program TSCSP-Sub (Table 3.2) in order to create the tier options, particularly for problem instances of size 10×23 and 50×50. Each generation has 50 chromosomes: 5 copied from the previous generation, 15 formed by mutation, and 30 formed by the crossover operation. The gene mutation probability, *mProb*, is set to 0.2. The maximum number of options per category per tier, *Max#OptionsPerCategoryPerTier*, is 100 for every problem size. This value limits the time spent creating the tier options but still allows a variety of excellent tier options to be considered by the GA. Makespan is computed via the greedy method for instances of size 50×50; otherwise the branch-and-bound method is used (Section 3.3.2).

Table 3.5. GA parameter settings in Experiment 1.

Problem Size	3 x 8	5 x 10	10 x 23	50 x 50
Computational time limit (seconds)	30	120	600	600
$\boldsymbol{N}$	50	50	50	50
$\boldsymbol{N_0}$	5	5	5	5
$\boldsymbol{N_1}$	15	15	15	15
$\boldsymbol{N_2}$	30	30	30	30
Gene mutation probability (*mProb*)	0.2	0.2	0.2	0.2
Max#OptionsPerCategoryPerTier	100	100	100	100
Evaluation of spreader changeover cost	Branch+Bound	Branch+Bound	Branch+Bound	Greedy

Table 3.6 shows the results for the 30 small problem instances of size 3×8. Each individual instance is specified by a code "TxSZ$_{nn}$" where T is the number of tiers; S is the number of stacks;

Z takes the value (L, M, H) according to the container weight scenario (light, medium, heavy); and "nn" denotes the instance number from 1 to 10. Instance "3x8L03" is depicted in Figures 3.1 and 3.2. Each instance is considered using (A) CPLEX's default integer programming (IP) solver with no time limit and (B) the GA with a 30 sec time limit. The GA creates and evaluates an average of 133,000 generations (6.7 million chromosomes) within the time limit. The best objective value (i.e. makespan) found by methods A and B are shown in columns M_{CP} and M_{GA}. Column M_0 shows the makespan of a greedy chromosome which combines the best options for the individual tiers (ties are broken arbitrarily). Column *LB* shows the lower bound.

The results show that both methods solve all instances to optimality. Indeed, method A solves all instances to optimality within eleven minutes, and method B finds these optimal solutions within 30 seconds. Importantly, the best solutions found by the GA (all of which happen to be optimal) are usually within 10% of the lower bound. Also, the average makespan of the best solution found by the GA (31.1) is about 7% better than that of the greedy solution (33.5). Not surprisingly, the optimal makespans for the light (medium) instances are typically less than those for the medium (heavy) instances.

Table 3.6. Experiment 1 results for TSCSP instances of size 3 × 8.

Instance	CPLEX		Heuristic (30 seconds)			LB	$\frac{M_{GA} - LB}{LB}$
	M_{CP}	Time (s)	M_{GA}	M_0	$\frac{M_0 - M_{GA}}{M_0}$		
3x8L01	28.6	549	28.6	31.2	8.33%	27.9	2.51%
3x8L02	30.3	635	30.3	33.9	10.62%	27.9	8.60%
3x8L03	27.2	71	27.2	27.4	0.73%	24.9	9.24%
3x8L04	23.3	22	23.3	24.0	2.92%	21.3	9.39%
3x8L05	27.2	169	27.2	32.7	16.82%	26.7	1.87%
3x8L06	30.3	365	30.3	35.0	13.43%	29.1	4.12%
3x8L07	25.5	25	25.5	28.8	11.46%	25.5	0.00%
3x8L08	25.5	102	25.5	28.8	11.46 %	25.5	0.00%
3x8L09	24.9	49	24.9	27.8	10.43%	24.9	0.00%
3x8L10	30.3	344	30.3	33.9	10.62%	28.5	6.32%
Average	27.3	233.1	27.3	30.4	9.68%	26.2	4.20%
3x8M01	29.1	47	29.1	29.6	1.69%	29.1	0.00%
3x8M02	33.9	9	33.9	35.4	7.63%	32.7	3.67%
3x8M03	29.5	18	29.5	30.9	4.53%	29.5	0.00%
3x8M04	32.2	40	32.2	34.8	15.23%	29.5	9.15%
3x8M05	35.1	238	35.1	35.7	15.13%	30.3	15.84%
3x8M06	25.5	19	25.5	28.8	11.46%	25.5	0.00%
3x8M07	33.9	176	33.9	35.9	8.91%	32.7	3.67%
3x8M08	29.1	51	29.1	32.3	9.91%	29.1	0.00%
3x8M09	35.4	140	35.4	38.6	15.28%	32.7	8.26%
3x8M10	33.0	581	33.0	36.2	18.51%	29.5	11.86%
Average	31.7	131.9	31.7	33.8	10.83%	30.1	5.25%
3x8H01	35.4	42	35.4	35.4	0.00%	32.7	8.26%
3x8H02	35.1	30	35.1	39.7	11.59%	32.7	7.34%
3x8H03	30.3	51	30.3	31.8	4.72%	29.1	4.12%
3x8H04	35.1	373	35.1	38.5	8.83%	31.5	11.43%
3x8H05	32.7	412	32.7	35.8	8.66%	31.5	3.81%
3x8H06	33.9	57	33.9	35.7	5.04%	30.3	11.88%
3x8H07	35.4	16	35.4	35.4	0.00%	32.7	8.26%
3x8H08	36.0	84	36.0	40.9	11.98%	33.9	6.19%
3x8H09	36.0	9	36.0	36.6	1.64%	33.9	6.19%
3x8H10	32.7	7	32.7	34.2	4.39%	31.5	3.81%
Average	34.3	108.1	34.3	36.4	5.68%	32.0	7.13%
Overall	31.1	157.7	31.1	33.5	8.73%	29.4	5.53%

Table 3.7 shows the results for the 30 instances of size 5×10. Here, each instance is considered using (A) CPLEX with a one-hour time limit and (B) the GA with a 120 second time limit. The GA creates and evaluates an average of 332,000 generations (16.6 million chromosomes) within the time limit. The results for methods A and B show that the GA performs better than CPLEX. Indeed, in every instance, the best solution found by the GA in two minutes is at least as good as the best solution found by CPLEX in an hour. Also, the average makespan of the best GA solution (62.2) is about 2.4% better than that for CPLEX (63.7). Note that the best solutions found by the GA are about 7% higher on average than the lower bound. Finally, we observe that the average makespan of the best GA solution (62.2) is about 7% lower than that of the greedy solution (67.1). These results indicate that standard IP is not a suitable solution method for instances with 50 or more containers. Thus, IP is not used in the following experiments that consider larger problem instances.

Table 3.8 shows the results for the large problem instances of size 10×23. For each instance, we show the makespan of the best solution found by the GA within 600 seconds; the makespan of a greedy solution; and the lower bound. The GA creates and evaluates an average of 34,000 generations (1.7 million chromosomes) within the time limit. The results show that the GA finds near-optimal solutions to these instances. Indeed, the average makespan of the best GA solution (270.2) is about 8% higher than the average lower bound (249.4). Note that the quality of the GA solution improves as containers get heavier; on average, the makespan of the best GA solution is roughly 11%, 10%, and 6% above the lower bound for the light, medium, and heavy instances respectively. This may be due to the fact that there are fewer opportunities for operating in dual-spreader and triple-spreader mode—and therefore fewer choices—when containers are heavier.

Table 3.7. Experiment 1 results for TSCSP instances of size 5 × 10.

Instance	CPLEX		Heuristic (120 seconds)			LB	$\frac{M_{GA} - LB}{LB}$
	M_{CP}	Time (s)	M_{GA}	M_0	$\frac{M_0 - M_{GA}}{M_0}$		
5x10L01	50.7	3734	49.0	58.5	16.24%	47.7	2.73%
5x10L02	56.5	3637	55.4	63.5	12.76%	50.0	10.80%
5x10L03	50.5	3656	49.1	53.2	7.71%	47.8	2.72%
5x10L04	57.3	3603	50.9	55.8	8.78%	47.7	6.71%
5x10L05	48.7	3603	48.7	54.1	9.98%	48.7	0.00%
5x10L06	58.5	3612	58.5	64.2	8.88%	53.4	9.55%
5x10L07	58.8	3620	56.5	62.4	9.46%	48.9	15.54%
5x10L08	57.6	3612	54.7	58.7	6.81%	53.3	2.63%
5x10L09	46.5	3614	46.5	51.9	10.40%	46.5	0.00%
5x10L10	50.5	3600	49.2	57.1	13.84%	47.8	2.93%
Average	53.6	3629.1	51.8	57.9	10.49%	49.2	5.36%
5x10M01	69.6	3616	68.7	74.6	7.91%	63.3	8.53%
5x10M02	62.0	3621	58.8	65.3	9.95%	54.5	7.89%
5x10M03	66.0	3600	64.2	68.5	6.28%	60.4	6.29%
5x10M04	67.5	3601	64.8	67.3	3.71%	59.2	9.46%
5x10M05	65.1	3661	59.7	67.0	10.90%	53.5	11.59%
5x10M06	68.6	3600	68.6	71.0	3.38%	62.9	9.06%
5x10M07	62.1	3603	62.1	68.6	9.48%	57.3	8.38%
5x10M08	69.3	3602	67.2	74.9	10.28%	60.9	10.34%
5x10M09	66.0	3606	66.0	69.6	5.17%	61.5	7.32%
5x10M10	61.2	3601	61.2	70.4	13.07%	56.9	7.56%
Average	65.7	3611.1	64.1	69.7	8.01%	59.0	8.64%
5x10H01	70.5	3601	69.6	75.5	7.81%	66.9	4.04%
5x10H02	72.9	3601	72.9	73.9	1.35%	66.9	8.97%
5x10H03	71.7	3647	71.7	75.4	4.91%	65.7	9.13%
5x10H04	72.0	3601	69.6	71.1	2.11%	65.7	5.94%
5x10H05	72.0	3606	68.4	70.0	2.29%	65.7	4.11%
5x10H06	72.9	3616	72.9	76.3	4.46%	69.3	5.19%
5x10H07	74.4	3618	74.4	75.9	1.98%	70.5	5.53%
5x10H08	68.4	3727	66.9	70.3	4.84%	63.3	5.69%
5x10H09	75.0	3600	75.0	77.2	2.85%	72.9	2.88%
5x10H10	67.2	3612	66.0	72.3	8.71%	61.5	7.32%
Average	71.7	3622.9	70.7	73.8	4.13%	66.8	5.88%
Overall	63.7	3621.0	62.2	67.1	7.54%	58.4	6.63%

Table 3.8. Experiment 1 results for TSCSP instances of size 10 × 23.

Instance	Heuristic (600 seconds)			LB	$\frac{M_{GA} - LB}{LB}$
	M_{GA}	M_0	$\frac{M_0 - M_{GA}}{M_0}$		
10x23L01	237.4	243.9	2.67%	216.9	9.45%
10x23L02	229.1	234.2	2.18%	204.5	12.03%
10x23L03	216.6	220.9	1.95%	196.6	10.17%
10x23L04	236.5	238.2	0.71%	213.9	10.57%
10x23L05	228.5	236.5	3.38%	206.8	10.49%
10x23L06	219.8	225.0	2.31%	198.0	11.01%
10x23L07	226.1	230.1	1.74%	203.1	11.32%
10x23L08	223.1	230.2	3.08%	203.2	9.79%
10x23L09	225.7	232.8	3.05%	205.8	9.67%
10x23L10	232.7	234.7	0.85%	207.7	12.04%
Average	227.5	232.7	2.19%	205.6	10.65%
10x23M01	264.5	270.3	2.15%	237.9	11.18%
10x23M02	265.7	270.3	1.70%	240.6	10.43%
10x23M03	268.5	276.5	2.89%	244.1	10.00%
10x23M04	279.0	286.4	2.58%	254.0	9.84%
10x23M05	268.3	272.0	1.36%	242.3	10.73%
10x23M06	271.4	279.5	2.90%	249.8	8.65%
10x23M07	276.4	283.4	2.47%	253.7	8.95%
10x23M08	274.5	283.3	3.11%	253.6	8.24%
10x23M09	274.2	278.0	1.37%	248.3	10.43%
10x23M10	274.0	278.9	1.76%	254.6	7.62%
Average	271.6	277.9	2.23%	247.9	9.61%
10x23H01	310.8	317.0	1.96%	294.9	5.39%
10x23H02	312.2	318.0	1.82%	299.1	4.38%
10x23H03	306.7	311.2	1.45%	284.2	7.92%
10x23H04	314.1	318.7	1.44%	300.9	4.39%
10x23H05	307.1	313.5	2.04%	291.9	5.21%
10x23H06	317.4	323.6	1.92%	299.3	6.05%
10x23H07	311.8	316.4	1.45%	292.1	6.74%
10x23H08	314.1	321.5	2.30%	297.2	5.69%
10x23H09	307.9	315.4	2.38%	291.1	5.77%
10x23H10	310.8	314.5	1.18%	295.6	5.14%
Average	311.3	317.0	1.79%	294.6	5.67%
Overall	270.2	276.9	2.05%	249.4	8.64%

Table 3.9 shows the results for the very large problem instances of size 50×50. This table has the same structure as Table 3.8. Here, the GA time limit is also 600 seconds. The GA creates and evaluates an average of 8200 generations (410,000 chromosomes) within the time limit. The results show that the GA finds near-optimal solutions to these instances. Indeed, the average makespan of the best GA solution (2810.8) is about 6.6% higher than the average lower bound (2637.2). On average, the makespan of the best GA solution is roughly 9%, 7%, and 5% above the lower bound for the light, medium, and heavy instances respectively. Overall, the GA appears to be an effective method for attacking the TSCSP.

3.5.2 Experiment 2

We now compare the performance of the GA to that of the simulated annealing algorithm (i.e. method "H") proposed in Chapter 2 (i.e. by Lashkari et al. 2017) on the 120 instances of the DSCSP considered in Chapter 2. Similar to the TSCSP instances, the DSCSP instances are broken into twelve categories corresponding to four problem sizes—3×8, 5×10, 10×23, 50×50—and three container weight scenarios—light, medium, and heavy—with ten instances in each category. Text files defining all DSCSP instances can be found in the supplementary material accompanying Lashkari et al. (2017). As in Chapter 2, we assume that H_1= 1.5, H_2 = 1.8, C_{12} = C_{21} = 2.1, and the weight limit L_2 for dual spreader lifts is 10.

Table 3.10 shows the GA settings used in this experiment. The computation times allocated to methods GA and H are identical and match the computation times used by Lashkari et al. (2017). As in Experiment 1, each generation has 50 chromosomes: 5 copied, 15 mutated, and 30 formed by crossover. Also, *mProb* is set to 0.2.

Table 3.9. Experiment 1 results for TSCSP instances of size 50 × 50.

Instance	Heuristic (600 seconds)			LB	$\frac{M_{GA} - LB}{LB}$
	M_{GA}	M_0	$\frac{M_0 - M_{GA}}{M_0}$		
50x50L01	2333.4	2363.0	1.25%	2160.5	8.00%
50x50L02	2322.6	2361.8	1.66%	2145.8	8.24%
50x50L03	2361.2	2390.8	1.24%	2177.5	8.44%
50x50L04	2418.1	2443.7	1.05%	2222.3	8.81%
50x50L05	2332.8	2373.8	1.73%	2149.7	8.52%
50x50L06	2391.7	2427.4	1.47%	2211.4	8.15%
50x50L07	2345.1	2390.6	1.90%	2158.4	8.65%
50x50L08	2370.5	2419.6	2.03%	2182.0	8.64%
50x50L09	2392.7	2421.2	1.18%	2205.2	8.50%
50x50L10	2358.5	2393.2	1.45%	2163.7	9.00%
Average	2362.7	2398.5	1.50%	2177.6	8.50%
50x50M01	2801.4	2812.7	0.40%	2621.0	6.88%
50x50M02	2779.1	2800.6	0.77%	2595.4	7.08%
50x50M03	2779.3	2808.7	1.05%	2598.1	6.97%
50x50M04	2797.4	2814.6	0.61%	2620.2	6.76%
50x50M05	2837.5	2845.0	0.26%	2650.7	7.05%
50x50M06	2855.9	2873.3	0.61%	2670.8	6.93%
50x50M07	2838.7	2864.8	0.91%	2651.5	7.06%
50x50M08	2844.0	2872.0	0.97%	2669.5	6.54%
50x50M09	2829.4	2864.5	1.23%	2645.8	6.94%
50x50M10	2797.5	2822.9	0.90%	2628.5	6.43%
Average	2816.0	2837.9	0.77%	2635.1	6.86%
50x50H01	3250.8	3288.5	1.15%	3096.8	4.97%
50x50H02	3238.5	3282.7	1.35%	3085.6	4.96%
50x50H03	3275.3	3324.8	1.49%	3127.7	4.72%
50x50H04	3237.4	3276.4	1.19%	3073.9	5.32%
50x50H05	3269.8	3310.2	1.22%	3113.1	5.03%
50x50H06	3264.7	3304.2	1.20%	3109.8	4.98%
50x50H07	3239.3	3293.2	1.64%	3085.3	4.99%
50x50H08	3275.8	3322.2	1.40%	3125.1	4.82%
50x50H09	3242.1	3275.9	1.03%	3092.3	4.84%
50x50H10	3243.4	3279.4	1.10%	3079.6	5.32%
Average	3253.7	3295.8	1.28%	3098.9	4.99%
Overall	2810.8	2844.1	1.18%	2637.2	6.78%

The last two rows of Table 3.10 require an explanation. In the DSCSP there are no triple-spreader lifts. Thus, there are only two categories of tier options—(1) those with at least one dual-spreader lift and (2) those with no dual-spreader lifts. The value *Max#OptionsPerCategoryPerTier* in Table 3.10 refers to the maximum number of options in category 1 that are generated per tier. (The number of options in category 2 that are generated per tier is always 1.) The value of this parameter—which was set based on preliminary experiments—allows a variety of good tier options to be created at the outset and leaves enough time for the GA to find good combinations of these options (i.e. good chromosomes) within the predetermined overall time limit. This parameter has a larger value here than in Experiment 1 because there are only two tier option categories to consider. Note that *Max#OptionsPerCategoryPerTier* for problem sizes 3×8 and 5×10 is large enough to include all unique, feasible tier options. The term "Basic" in the final row of Table 3.10 refers to a method that selects the better of the two possible spreader mode sequences: S-D-S-D-S-… and D-S-D-S-D-….

Table 3.10. GA parameter settings in Experiment 2.

Problem Size	3 x 8	5 x 10	10 x 23	50 x 50
Computational time limit (seconds)	10	60	600	600
$\boldsymbol{N}$	50	50	50	50
$\boldsymbol{N_0}$	5	5	5	5
$\boldsymbol{N_1}$	15	15	15	15
$\boldsymbol{N_2}$	30	30	30	30
Gene mutation probability *(mProb)*	0.2	0.2	0.2	0.2
Max#TierOptionsPerCategoryPerTier	200	200	200	200
Evaluation of spreader changeover cost	Basic	Basic	Basic	Basic

Table 3.11 shows the overall results from this experiment. Here, results are aggregated according to problem category. The twelve problem categories are shown in column 1, and the allowed computation time for each instance is shown in column 2. The results for method H and the lower bound on the optimal objective value—which are copied from Chapter 2—are shown in

columns 3 and 5 respectively. The results for the GA are shown in column 4. Note that the GA performs at least as well as method H for every problem category. On average, the solutions found by the GA are 0.48% better than method H. The solutions found by the GA are 4.07% above the lower bound on average. These results demonstrate the GA's effectiveness in addressing the DSCSP.

Table 3.11. Comparing SA and GA performance on the 120 DSCSP instances (average for each instance category shown).

Instance Category	Heuristic Runtime (s)	M_H	M_{GA}	LB	$\frac{M_H - M_{GA}}{M_H}$	$\frac{M_{GA} - LB}{LB}$
3x8L	10	26.8	26.8	25.9	0.00%	3.47%
3x8M	10	30.7	30.7	29.2	0.00%	5.14%
3x8H	10	33.5	33.5	32.6	0.00%	2.76%
Average	10	30.3	30.3	29.2	0.00%	3.79%
5x10L	60	55.8	55.8	52.6	0.00%	6.08%
5x10M	60	62.2	62.2	58.1	0.00%	7.06%
5x10H	60	69.8	69.8	66.1	0.00%	5.60%
Average	60	62.6	62.6	58.9	0.00%	6.25%
10x23L	600	242.5	239.9	231.8	1.07%	3.49%
10x23M	600	274.4	274.2	262.6	0.07%	4.42%
10x23H	600	308.5	308.5	296.9	0.00%	3.91%
Average	600	275.2	274.2	263.8	0.38%	3.94%
50x50L	600	2623.5	2547.0	2492.9	2.92%	2.17%
50x50M	600	2914.2	2871.6	2801.2	1.46%	2.51%
50x50H	600	3272.8	3263.5	3192.5	0.28%	2.22%
Average	600	2936.8	2894.0	2828.9	1.55%	2.30%
Overall	-	-	-	-	0.48%	4.07%

3.5.3 Experiment 3

In Section 3.3.1 we introduced an algorithm to create tier options. This preliminary step generates genes that are basically feasible ways to lift the containers in a tier.

Every chromosome consists of one gene per tier, and each chromosome is fed to the GA for evaluation and makespan computation.

In experiment 1, CPLEX generates tier options in four categories to provide a variety of genes in terms of combining single, dual, and triple lifts. For each category and each tier, the 100 best tier options were generated and indexed from the best lift time to the worst. This is particularly important because the first chromosome in the first generation always consists of the best gene for each tier.

For small problem instances, CPLEX performed efficiently and provided tier options quickly. Indeed, for small-size problems (3×8 and 5×10), the majority of computation time was spent forming new GA generations and assessing the fitness of chromosomes. However, for larger problem instances (10×23 and 50×50), the majority of computation time was spent generating tier options prior to the main GA routine. This led to relatively few GA generations being created, and it limited the chances of finding solutions with small makespans.

To rectify this problem, we introduce dynamic programming (DP). Dynamic programming breaks down a complex optimization problem into smaller incremental steps and tries to solve each step using the best solutions that were identified in previous steps (Allison, 2020). In this experiment, we deploy dynamic programming to find the best tier options for the TSCSP. We expect a DP-based method to generate tier options more quickly than an integer programming based-method, leaving additional computation time for the main part of the GA routine.

Without loss of generality, we focus on tier t and assume that the options for tier t need to be generated. Tier t consists of S stacks. A feasible tier option for tier t consists of a series of single (S), dual (D), and triple (T) lifts that agree with the rules of legal binary

dual/triple spreader lifts. At every step, dynamic programming evaluates legal spreader options based on the binary information contained in matrices $L2$ and $L3$ (Figure 3.2), calculates the partial tier option lift time, and ranks the solutions.

In the TSCSP, we have at most three comparisons when deciding on the type of spreader for stack s. If neither the dual nor triple lifts are possible, the container on tier t stack s would be lifted by a single spreader (S). If a dual lift is legal but a triple lift is not, the algorithm compares a single lift to a dual lift and chooses the smaller lift time. A dual lift is legal when the corresponding value matrix $L2 = 1$. Another scenario is when a triple lift is legal (the $L3 = 1$) and a dual lift is not. In this case, a comparison between the time it takes to lift the container in stack s with a single spreader and the time it takes to lift the containers in stacks s-2, s-1 and s with a triple-spreader takes place. The final scenario is where all three spreaders can be used to lift the container in stack s. As mentioned above, the dynamic programming algorithm looks at previous steps to make the best decision for the current state.

Let $OPT[s]$ be the minimum handling time for the first s containers in tier t (spreader changeover time is ignored). In our basic DP, we use the formula below to calculate the best objective value at each step.

$$\begin{aligned} OPT[s] = Minimum\ \{&OPT[s-1] + singleSpreaderOptionForDP,\\ &OPT[s-2] + dualSpreaderOptionForDP,\\ &OPT[s-3] + tripleSpreaderOptionForDP\} \end{aligned}$$

Where:

$$singleSpreaderOptionForDP = H_1$$

$$dualSpreaderOptionForDP = \begin{cases} H_2 & if \quad L2_{s-1,t} = 1 \\ \infty & \text{otherwise} \end{cases} \qquad (t = 1, \ldots, T)$$

$$tripleSpreaderOptionForDP = \begin{cases} H_3 & if \quad L3_{s-2,t} = 1 \\ \infty & \text{otherwise} \end{cases} \qquad (t = 1, \ldots, T)$$

This above recursion formula finds the optimal tier option for tier *t*. However, in our advanced DP algorithm, which is a combination of enumeration and dynamic programming, we are interested in generating the *Max#OptionsPerCategoryPerTier* (= 100 as in experiment 1) best tier options for each tier. Let *OPT*[*m*][*s*] be the handling time (spreader changeover time is ignored) for the m^{th} best (unique) option (*m* = 1, …, *Max#OptionsPerCategoryPerTier)* for handling the first *s* containers in tier *t*. The values of *OPT*[*m*][*s*] for *m* = 1 to *Max#OptionsPerCategoryPerTier* can be derived from the values of *OPT*[*m*][*s*-1], *OPT*[*m*][*s*-2], and *OPT*[*m*][*s*-3] from *m* = 1 to *Max#OptionsPerCategoryPerTier* using a recursive procedure similar to that shown above.

Table 3.12, 3.13, and 3.14 show the algorithms we developed to generate tier options in 3 categories: at least one dual and at least one triple lift, at least one dual and no triple lift, and at least one triple and no dual. In these tables, $L'2_{st}$ is equal to 1 if the two containers occupying stacks *s* and *s*-1 in tier *t* can be lifted together using the dual spreader without violating the weight limit (binary) (*s* = 1, 2, …, *S*-1; *t* = 1, 2, …, *T*). $L'3_{st}$ is equal to 1 if the three containers occupying stacks *s*, *s*-1, and *s*-2 in tier *t* can be lifted together using the triple spreader without violating the weight limit (binary) (*s* = 1, 2, …, *S*-2; *t* = 1, 2, …, *T*). *SpreaderVector*[*m*][*s*] stores the sequence of lifts (S, D, T) that are made from left to right for the m^{th} best option for handling the first *s* containers in tier *t.* For example, if we are generating tier options with at least one triple lift and no dual lifts, then *SpreaderVector*[1][8] = (T, T, S, S) for tier 1 in the example shown in the Figures 3.3 and 3.4. In other words, the best way to lift the first 8 containers in tier 1 in the example shown in Figures 3.3 and 3.4 is to lift the first 3 containers using the triple spreader, the next 3 containers using the triple spreader, and the last 2 containers each with a single spreader. This is tier option 19 (for tier 1), and the corresponding lifting time for this, *OPT*[1][8] is 7.4 minutes.

Table 3.12. DP algorithm for generating options for tier *t* with at least one dual and at least one triple lift.

Set $OPT[1][1] = H_1$ and Let $SpreaderVector[1][1]$ = "S"
Set $OPT[m][1] = +\infty$ for $m = 2, \ldots, Max\#OptionsPerCategoryPerTier$
if $L'2_{2t} = 1$ **then**
    $OPT[1][2] = H_2$ and let $SpreaderVector[1][2]$ = "D"
    $OPT[2][2] = 2H_1$ and let $SpreaderVector[2][2]$ = "SS"
else $OPT[1][2] = 2H_1$ and let $SpreaderVector[1][2]$ = "SS"
Set the rest of $OPT[m][2] = +\infty$ for $m = 2$ or 3, …, $Max\#OptionsPerCategoryPerTier$
for $s = 3$ to S **do**
    Set $n_1 = n_2 = n_3 = 1$
    for $m = 1$ to $Max\#OptionsPerCategoryPerTier$ **do**
        if $L'3_{st} = 1$ **then** $tripleSpreaderOptionForDP = H_3$ **else** $tripleSpreaderOptionForDP = \infty$
        if $L'2_{st} = 1$ **then** $dualSpreaderOptionForDP = H_2$ **else** $dualSpreaderOptionForDP = \infty$
        $singleSpreaderOptionForDP = H_1$
        $OPT[m][s] = Minimum\ \{OPT[n_1][s\text{-}1] + singleSpreaderOptionForDP,$
                $OPT[n_2][s\text{-}2] + dualSpreaderOptionForDP,$
                $OPT[n_3][s\text{-}3] + tripleSpreaderOptionForDP\}$
        if the minimum comes from using $tripleSpreaderOptionForDP$ **then**
            $n_3 = n_3 + 1$ & let $SpreaderVector[m][s]$ equal $SpreaderVector[n_3][s\text{-}3]$ with a "T" added to the end of it.
        else if the minimum comes from using $dualSpreaderOptionForDP$ **then**
            $n_2 = n_2 + 1$ & let $SpreaderVector[m][s]$ equal $SpreaderVector[n_2][s\text{-}2]$ with a "D" added to the end of it.
        else if the minimum comes from using $singleSpreaderOptionForDP$ **then**
            $n_1 = n_1 + 1$ & let $SpreaderVector[m][s]$ equal $SpreaderVector[n_1][s\text{-}1]$ with an "S" added to the end of it.
        else if no more options for comparison remain among previous stacks **then**
            Set $OPT[n][s] = +\infty$ for all n from m to $Max\#OptionsPerCategoryPerTier$
for $m = 1$ to $Max\#OptionsPerCategoryPerTier$ **do**
    if $SpreaderVector[m][S]$ doesn't have at least one "D" and at least one "T" in it **then**
        remove $SpreaderVector[m][S]$ and $OPT[m][S]$
output all $OPT[m][S]$ and corresponding $SpreaderVector[m][S]$ values that haven't been removed.

Table 3.13. DP algorithm for generating options for tier t with at least one dual and no triple lift.

Set $OPT[1][1] = H_1$ and let $SpreaderVector[1][1]$ = "S"
Set $OPT[m][1] = +\infty$ for m = 2, …, $Max\#OptionsPerCategoryPerTier$
if $L'2_{2t} = 1$ **then**
    $OPT[1][2] = H_2$ and let $SpreaderVector[1][2]$ = "D"
    $OPT[2][2] = 2H_1$ and let $SpreaderVector[2][2]$ = "SS"
else $OPT[1][2] = 2H_1$ and let $SpreaderVector[1][2]$ = "SS"
Set the rest of $OPT[m][2] = +\infty$ for m = 2 or 3, …, $Max\#OptionsPerCategoryPerTier$
for s = 3 to S **do**
    Set $n_1 = n_2 = 1$
    for m = 1 to $Max\#OptionsPerCategoryPerTier$ **do**
        if $L'2_{st} = 1$ **then** $dualSpreaderOptionForDP = H_2$ **else** $dualSpreaderOptionForDP = \infty$
        $singleSpreaderOptionForDP = H_1$
        $OPT[m][s] = Minimum\ \{OPT[n_1][s\text{-}1] + singleSpreaderOptionForDP,$
                                $OPT[n_2][s\text{-}2] + dualSpreaderOptionForDP\}$
        if the minimum comes from using $dualSpreaderOptionForDP$ **then**
            $n_2 = n_2 + 1$ & let $SpreaderVector[m][s]$ equal $SpreaderVector[n_2][s\text{-}2]$ with a "D" added to the end of it.
        else if the minimum comes from using $singleSpreaderOptionForDP$ **then**
            $n_1 = n_1 + 1$ & let $SpreaderVector[m][s]$ equal $SpreaderVector[n_1][s\text{-}1]$ with an "S" added to the end of it.
        else if no more options for comparison remain among previous stacks **then**
            Set $OPT[n][s] = +\infty$ for all n from m to $Max\#OptionsPerCategoryPerTier$
for m = 1 to $Max\#OptionsPerCategoryPerTier$ **do**
    if $SpreaderVector[m][S]$ doesn't have at least one "D" in it **then**
        remove $SpreaderVector[m][S]$ and $OPT[m][S]$
output all $OPT[m][S]$ and corresponding $SpreaderVector[m][S]$ values that haven't been removed

Table 3.15 shows the experimental results for using GA to attack the TSCSP instances of size 3×8 when using dynamic programming to generate tier options. The table has the same format as Table 3.6 except that in this table M_{DP} shows the best objective value found by the DP-supported GA within the time limit. On average 220,000 generations (11 million chromosomes) were generated within the time limit which is 1.8 times more than in experiment 1. Similar to Table 3.6, the CPLEX-based method (column M_{CP}) solves all instances to optimality within 11 minutes, and the GA (using DP) finds these optimal solutions within 30 seconds. All results in Table 3.15 match those for the IP-based GA in Table 3.6. The best solutions found by the DP-based GA are usually within 10% of the lower bound. Also, the average makespan of the best solution found by the GA (using DP) is about 7% better than that of the greedy solution.

Table 3.14. DP algorithm for generating options for tier t with at least one triple and no dual lift.

Set $OPT[1][1] = H_1$ and let $SpreaderVector[1][1]$ = "S"
Set $OPT[m][1] = +\infty$ for $m = 2, \ldots, Max\#OptionsPerCategoryPerTier$
Set $OPT[1][2] = 2H_1$ and let $SpreaderVector[1][2]$ = "SS"
Set the rest of $OPT[m][2] = +\infty$ for $m = 2, \ldots, Max\#OptionsPerCategoryPerTier$
for $s = 3$ to S **do**
    Set $n_1 = n_3 = 1$
    for $m = 1$ to $Max\#OptionsPerCategoryPerTier$ **do**
        if $L'3_{st} = 1$ **then** $tripleSpreaderOptionForDP = H_3$ **else** $tripleSpreaderOptionForDP = \infty$
        $singleSpreaderOptionForDP = H_1$
        $OPT[m][s] = Minimum\ \{OPT[n_1][s\text{-}1] + singleSpreaderOptionForDP,$
                $OPT[n_3][s\text{-}3] + tripleSpreaderOptionForDP\}$
        if the minimum comes from using $tripleSpreaderOptionForDP$ **then**
            $n_3 = n_3 + 1$ & let $SpreaderVector[m][s]$ equal $SpreaderVector[n_3][s\text{-}3]$ with a "T" added to the end of it.
        else if the minimum comes from using $singleSpreaderOptionForDP$ **then**
            $n_1 = n_1 + 1$ & let $SpreaderVector[m][s]$ equal $SpreaderVector[n_1][s\text{-}1]$ with an "S" added to the end of it.
        else if no more options for comparison remain among previous stacks **then**
            Set $OPT[m][s] = +\infty$ for all n from m to $Max\#OptionsPerCategoryPerTier$
for $m = 1$ to $Max\#OptionsPerCategoryPerTier$ **do**
    if $SpreaderVector[m][S]$ doesn't have at least one "T" in it **then**
        remove $SpreaderVector[m][S]$ and $OPT[m][S]$
output all $OPT[m][S]$ and corresponding $SpreaderVector[m][S]$ values that haven't been removed

Table 3.16 shows the results for 5×10 instances. This table compares the CPLEX method with a one-hour time limit and the DP-supported GA with a 120 second time limit. The GA creates and evaluates an average of 360,000 generations (18 million chromosomes) within the time limit. The results for the DP-supported GA are very similar to the results in Table 3.7. Although the GA with DP-based tier generation creates 1.14 times more chromosomes compared to the method in Section 3.3.1, the results are identical to those in table 3.7 except for one instance in which DP-based GA finds a better makespan compared to IP-based GA. The best solutions found by the GA are about 7% higher on average than the lower bound. Finally, we observe that the average makespan of the best GA solution is about 7% lower than that of the greedy solution.

Table 3.15. Experiment 3 results for TSCSP instances of size 3 × 8.

Instance	CPLEX		GA (using DP) (30 seconds)			LB	$\frac{M_{DP} - LB}{LB}$
	M_{CP}	Time (s)	M_{DP}	M_0	$\frac{M_0 - M_{DP}}{M_0}$		
3x8L01	28.6	549	28.6	31.2	8.33%	27.9	2.51%
3x8L02	30.3	635	30.3	33.9	10.62%	27.9	8.60%
3x8L03	27.2	71	27.2	27.4	0.73%	24.9	9.24%
3x8L04	23.3	22	23.3	24.0	2.92%	21.3	9.39%
3x8L05	27.2	169	27.2	32.7	16.82%	26.7	1.87%
3x8L06	30.3	365	30.3	35.0	13.43%	29.1	4.12%
3x8L07	25.5	25	25.5	28.8	11.46%	25.5	0.00%
3x8L08	25.5	102	25.5	28.8	11.46 %	25.5	0.00%
3x8L09	24.9	49	24.9	27.8	10.43%	24.9	0.00%
3x8L10	30.3	344	30.3	33.9	10.62%	28.5	6.32%
Average	27.3	233.1	27.3	30.4	9.68%	26.2	4.20%
3x8M01	29.1	47	29.1	29.6	1.69%	29.1	0.00%
3x8M02	33.9	9	33.9	35.4	7.63%	32.7	3.67%
3x8M03	29.5	18	29.5	30.9	4.53%	29.5	0.00%
3x8M04	32.2	40	32.2	34.8	15.23%	29.5	9.15%
3x8M05	35.1	238	35.1	35.7	15.13%	30.3	15.84%
3x8M06	25.5	19	25.5	28.8	11.46%	25.5	0.00%
3x8M07	33.9	176	33.9	35.9	8.91%	32.7	3.67%
3x8M08	29.1	51	29.1	32.3	9.91%	29.1	0.00%
3x8M09	35.4	140	35.4	38.6	15.28%	32.7	8.26%
3x8M10	33.0	581	33.0	36.2	18.51%	29.5	11.86%
Average	31.7	131.9	31.7	33.8	10.83%	30.1	5.25%
3x8H01	35.4	42	35.4	35.4	0.00%	32.7	8.26%
3x8H02	35.1	30	35.1	39.7	11.59%	32.7	7.34%
3x8H03	30.3	51	30.3	31.8	4.72%	29.1	4.12%
3x8H04	35.1	373	35.1	38.5	8.83%	31.5	11.43%
3x8H05	32.7	412	32.7	35.8	8.66%	31.5	3.81%
3x8H06	33.9	57	33.9	35.7	5.04%	30.3	11.88%
3x8H07	35.4	16	35.4	35.4	0.00%	32.7	8.26%
3x8H08	36.0	84	36.0	40.9	11.98%	33.9	6.19%
3x8H09	36.0	9	36.0	36.6	1.64%	33.9	6.19%
3x8H10	32.7	7	32.7	34.2	4.39%	31.5	3.81%
Average	34.3	108.1	34.3	36.4	5.68%	32.0	7.13%
Overall	31.1	157.7	31.1	33.5	8.73%	29.4	5.53%

Table 3.16. Experiment 3 results for TSCSP instances of size 5 × 10 (* indicates a better result than in experiment 1).

Instance	CPLEX		GA (using DP) (30 seconds)			LB	$\frac{M_{DP}-LB}{LB}$
	M_{CP}	Time (s)	M_{DP}	M_0	$\frac{M_0-M_{DP}}{M_0}$		
5x10L01	50.7	3734	49.0	58.5	16.24%	47.7	2.73%
5x10L02	56.5	3637	55.4	63.5	12.76%	50.0	10.80%
5x10L03	50.5	3656	49.1	53.2	7.71%	47.8	2.72%
5x10L04	57.3	3603	50.9	55.8	8.78%	47.7	6.71%
5x10L05	48.7	3603	48.7	54.1	9.98%	48.7	0.00%
5x10L06	58.8	3612	58.5	64.2	8.88%	53.4	9.55%
5x10L07	57.6	3620	56.3*	62.4	9.78%	48.9	15.13%
5x10L08	54.4	3612	54.7	58.7	6.81%	53.3	2.63%
5x10L09	46.5	3614	46.5	51.9	10.40%	46.5	0.00%
5x10L10	50.5	3600	49.2	57.1	13.84%	47.8	2.93%
Average	53.6	3629.1	51.8	57.9	10.49%	49.2	5.35%
5x10M01	69.6	3616	68.7	74.6	7.91%	63.3	8.53%
5x10M02	62.0	3621	58.8	65.3	9.95%	54.5	7.89%
5x10M03	66.0	3600	64.2	68.5	6.28%	60.4	6.29%
5x10M04	67.5	3601	64.8	67.3	3.71%	59.2	9.46%
5x10M05	65.1	3661	59.7	67.0	10.90%	53.5	11.59%
5x10M06	68.6	3600	68.6	71.0	3.38%	62.9	9.06%
5x10M07	62.1	3603	62.1	68.6	9.48%	57.3	8.38%
5x10M08	69.3	3602	67.2	74.9	10.28%	60.9	10.34%
5x10M09	66.0	3606	66.0	69.6	5.17%	61.5	7.32%
5x10M10	61.2	3601	61.2	70.4	13.07%	56.9	7.56%
Average	65.7	3611.1	64.1	69.7	8.01%	59.0	8.64%
5x10H01	70.5	3601	69.6	75.5	7.81%	66.9	4.04%
5x10H02	72.9	3601	72.9	73.9	1.35%	66.9	8.97%
5x10H03	71.7	3647	71.7	75.4	4.91%	65.7	9.13%
5x10H04	72.0	3601	69.6	71.1	2.11%	65.7	5.94%
5x10H05	72.0	3606	68.4	70.0	2.29%	65.7	4.11%
5x10H06	72.9	3616	72.9	76.3	4.46%	69.3	5.19%
5x10H07	74.4	3618	74.4	75.9	1.98%	70.5	5.53%
5x10H08	68.4	3727	66.9	70.3	4.84%	63.3	5.69%
5x10H09	75.0	3600	75.0	77.2	2.85%	72.9	2.88%
5x10H10	67.2	3612	66.0	72.3	8.71%	61.5	7.32%
Average	71.7	3622.9	70.7	73.8	4.13%	66.8	5.88%
Overall	63.7	3621.0	62.2	67.1	7.54%	58.4	6.63%

Table 3.17 shows the results for the large problem instances of size 10×23. The GA creates and evaluates an average of 60,000 generations (3 million chromosomes) within the time limit, which is 1.76 times more than in Table 3.8. The results show that the DP-based GA is not only creating more generations but also is finding slightly better makespans on average (0.26%), especially for lighter instances (0.35%). Among 30 instances in categories light, medium, and heavy, 21 instances have improved makespans using DP for tier generation. The remaining 9 makespans are as good as the IP-based GA. The average makespan of the best GA solution (269.5) is about 8% higher than the average lower bound (249.4). Note that the quality of the GA solution improves as containers get heavier; on average, the makespan of the best GA solution is roughly 10%, 9%, and 6% above the lower bound for the light, medium, and heavy instances respectively.

Table 3.18 shows the results for the very large problem instances of size 50×50. The DP-supported GA creates and evaluates an average of 23,000 generations (1.15 million chromosomes) within the time limit. The results show that the DP-supported GA not only creates more chromosomes (2.8 times more) but also finds solutions with better makespans on average (0.15%). Among the 30 instances of size 50×50, 28 instances have better makespans when using DP for tier generation. The other two instances have worse makespans than the IP-based GA (heavy instances). The average makespan of the best GA solution (2806.5) is about 6.6% higher than the average lower bound (2637.2). On average, the makespan of the best GA solution is roughly 8%, 7%, and 5% above the lower bound for the light, medium, and heavy instances respectively. Overall, DP generates tier options more quickly than integer programming, and the performance of the GA is slightly enhanced when the tier options for TSCSP instances are generated using DP instead of integer programming.

Table 3.17. Experiment 3 results for TSCSP instances of size 10 × 23 (* indicates a better result than in experiment 1).

Instances	GA (using DP) (600 seconds)			LB	$\frac{M_{DP}-LB}{LB}$
	M_{DP}	M_0	$\frac{M_0-M_{DP}}{M_0}$		
10x23L01	237.3*	243.9	2.71%	216.9	9.41%
10x23L02	228.8*	234.2	2.31%	204.5	11.88%
10x23L03	215.9*	220.9	2.26%	196.6	9.82%
10x23L04	235.8*	238.2	1.01%	213.9	10.24%
10x23L05	227.5*	236.5	3.81%	206.8	10.01%
10x23L06	219.2*	225.0	2.58%	198.0	10.71%
10x23L07	224.7*	230.1	2.35%	203.1	10.64%
10x23L08	223.1	230.2	3.08%	203.2	9.79%
10x23L09	224.7*	232.8	3.48%	205.8	9.18%
10x23L10	229.9*	234.7	2.05%	207.7	10.69%
Average	226.7*	232.7	2.58%	205.6	10.26%
10x23M01	264.4*	270.3	2.18%	237.9	11.14%
10x23M02	263.7*	270.3	2.55%	240.6	9.48%
10x23M03	267.9*	276.5	3.11%	244.1	9.75%
10x23M04	278.7*	286.4	2.69%	254.0	9.72%
10x23M05	267.5*	272.0	1.65%	242.3	10.40%
10x23M06	271.1*	279.5	3.01%	249.8	8.53%
10x23M07	274.9*	283.4	3.00%	253.7	8.36%
10x23M08	274.5	283.3	3.11%	253.6	8.24%
10x23M09	272.9*	278.0	1.83%	248.3	9.91%
10x23M10	273.3*	278.9	2.01%	254.6	7.34%
Average	270.9*	277.9	2.53%	247.9	9.26%
10x23H01	310.8	317.0	1.96%	294.9	5.39%
10x23H02	310.5*	318.0	2.36%	299.1	3.81%
10x23H03	306.7	311.2	1.45%	284.2	7.92%
10x23H04	314.1	318.7	1.44%	300.9	4.39%
10x23H05	306.9*	313.5	2.11%	291.9	5.14%
10x23H06	317.4	323.6	1.92%	299.3	6.05%
10x23H07	311.7*	316.4	1.49%	292.1	6.71%
10x23H08	314.1	321.5	2.30%	297.2	5.69%
10x23H09	307.9	315.4	2.38%	291.1	5.77%
10x23H10	310.8	314.5	1.18%	295.6	5.14%
Average	311.1*	317.0	1.86%	294.6	5.60%
Overall	269.5*	276.9	2.31%	249.4	8.37%

Table 3.18. Experiment results for TSCSP instances of size 50 × 50 (*/Δ indicates a better/worse result than in experiment 1).

Instances	GA (using DP) (600 seconds)			LB	$\frac{M_{DP}-LB}{LB}$
	M_{DP}	M_0	$\frac{M_0-M_{DP}}{M_0}$		
50x50L01	2328.5*	2363.0	1.46%	2160.5	7.78%
50x50L02	2321.7*	2361.8	1.70%	2145.8	8.20%
50x50L03	2360.7*	2390.8	1.26%	2177.5	8.41%
50x50L04	2407.6*	2443.7	1.48%	2222.3	8.34%
50x50L05	2316.8*	2373.8	2.40%	2149.7	7.77%
50x50L06	2384.5*	2427.4	1.77%	2211.4	7.83%
50x50L07	2344.9*	2390.6	1.91%	2158.4	8.64%
50x50L08	2357.9*	2419.6	2.55%	2182.0	8.06%
50x50L09	2391.7*	2421.2	1.22%	2205.2	8.46%
50x50L10	2341.8*	2393.2	2.15%	2163.7	8.23%
Average	2355.6*	2398.5	1.79%	2177.6	8.17%
50x50M01	2796.9*	2812.7	0.56%	2621.0	6.71%
50x50M02	2778.6*	2800.6	0.79%	2595.4	7.06%
50x50M03	2778.2*	2808.7	1.09%	2598.1	6.93%
50x50M04	2797.1*	2814.6	0.62%	2620.2	6.75%
50x50M05	2830.0*	2845.0	0.53%	2650.7	6.76%
50x50M06	2851.8*	2873.3	0.75%	2670.8	6.78%
50x50M07	2830.4*	2864.8	1.20%	2651.5	6.75%
50x50M08	2840.1*	2872.0	1.11%	2669.5	6.39%
50x50M09	2828.2*	2864.5	1.27%	2645.8	6.89%
50x50M10	2795.6*	2822.9	0.97%	2628.5	6.36%
Average	2812.7*	2837.9	0.89%	2635.1	6.74%
50x50H01	3246.9*	3288.5	1.27%	3096.8	4.85%
50x50H02	3236.5*	3282.7	1.41%	3085.6	4.89%
50x50H03	3273.8*	3324.8	1.53%	3127.7	4.67%
50x50H04	3233.7*	3276.4	1.30%	3073.9	5.20%
50x50H05	3268.0*	3310.2	1.27%	3113.1	4.98%
50x50H06	3264.6*	3304.2	1.20%	3109.8	4.98%
50x50H07	3236.3*	3293.2	1.73%	3085.3	4.89%
50x50H08	3276.5Δ	3322.2	1.38%	3125.1	4.84%
50x50H09	3242.2Δ	3275.9	1.03%	3092.3	4.85%
50x50H10	3233.7*	3279.4	1.39%	3079.6	5.00%
Average	3251.2	3295.8	1.35%	3098.9	4.92%
Overall	2806.5	2844.1	1.34%	2637.2	6.61%

Chapter 4:

Conclusion

4.1 Concluding remarks

In this dissertation, we investigated two new crane scheduling problems—the dual-spreader crane scheduling problem (DSCSP) and triple-spreader crane scheduling problem (TSCSP)—which are inspired by the multi-spreader (i.e. tandem-lift) quay crane (QC), an emerging technology for handling cargo at seaport container terminals. The efficient operation of such cranes may allow containerships to be unloaded more quickly and thereby improve overall container terminal efficiency.

In Chapter 2, we formulated the DSCSP as a mixed-integer linear program, developed a tight lower bound on the optimal value, and devised a heuristic approach for handling large problem instances. The heuristic approach begins with an excellent initial feasible solution that effectively utilizes the problem structure. A simulated annealing framework was used to improve upon the initial feasible solution. Numerical experiments indicate that the heuristic approach finds the same optimal solutions as CPLEX for small-sized instances. For medium-sized instances, the heuristic outperforms CPLEX. The comparison between the optimal value and lower bound for small-sized instances suggests that the lower bound is tight, providing a good guide for solution quality. Overall, the heuristic approach produces crane schedules whose makespans, on average, are within 6% of the lower bound for each of the four problem sizes considered.

In Chapter 3, we formulated the TSCSP as an integer linear program, calculated a tight lower bound on the optimal value, and deployed a genetic algorithm (GA) with an embedded

dynamic programming (DP) routine for attacking large problem instances. The genes in the GA chromosomes are tier options that are constructed based on knowledge of the problem structure. Numerical experiments indicate that the GA finds the same optimal solutions as CPLEX for small problem instances. For instances with at least 50 containers, the GA outperforms CPLEX. On average, the GA finds crane schedules whose makespans are within (5.53%, 6.63%, 8.37%, 6.61%) of the lower bound for (small, medium-sized, large, very large) problem instances—an overall average of 6.8% above the lower bound. The GA also outperforms the simulated-annealing-based method proposed in Chapter 2 on instances of the dual-spreader crane scheduling problem (DSCSP). Overall, the GA appears to be an effective method for addressing both the TSCSP and DSCSP.

Although we solely consider the unloading of a storage bay, our approach can apply to the loading of a storage bay. Indeed, reversing the sequence of operations—single-spreader lifts, dual-spreader lifts, triple-spreader lifts, and changeovers—creates a schedule for loading a storage bay in the same amount of time in which it is unloaded.

4.2 Future work

Future work might proceed in several directions. First, the NP-hardness of the DSCSP (or lack thereof) could be established. Second, the problem of scheduling multiple quay cranes to unload containers from the deck of a containership to minimize total unloading time can be investigated. This problem may include constraints on the movement of the quay cranes.

Third, DSCSP and TSCSP instances with other container weight distributions; lift weight limits L_2 and L_3; lift durations H_1, H_2, and H_3; and spreader changeover times C_{pq} could be considered. Perhaps most importantly, more realistic variations of this problem—that consider

more realistic positions of heavy and light containers, multiple QCs working together to unload a vessel, additional real-world constraints, and/or the distance moved by the spreader—might be considered.

Finally, the solutions developed in this study might be integrated into an end-to-end container shipping transportation problem. This problem might focus on a containership that 1) loads containers at an origin seaport, 2) stops at multiple intermediate ports to unload some onboard containers and load new containers, and 3) travels all the way to a final destination port to unload the remaining containers. In this complex problem, many new factors must be taken into account. First and foremost, loading containers must follow a schedule subject to the destination of each container. In order to have an optimal 3D deck layout, the position of each container should be a variable in the mathematical model. In addition, the weight distribution of containers should follow certain standards to keep the vessel balanced while it is at port and traveling the high seas.

References

1. Allison, L. (2020). Dynamic Programming, accessed on July 19th 2020, Retrieved from http://users.monash.edu/~lloyd/tildeAlgDS/Dynamic/

2. Angeloudis, P., & Bell, M. G. (2011). A review of container terminal simulation models. *Maritime Policy & Management*, 38 (5), 523–540.

3. Bernhofen, D. M., El-Sahli, Z. & Kneller, R. (2016). Estimating the effects of the container revolution on world trade. *Journal of International Economics*, 98, 36–50.

4. 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.

5. 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.

6. Broeze, F. (2002). The Globalization of the Oceans: Containerization from the 1950s to the Present. 23, Oxford University Press.

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

8. Carlo, H. J., Vis, I. F., & Roodbergen, K. J. (2014b). Transport operations in container terminals: Literature overview, trends, research directions and classification scheme. *European Journal of Operational Research*, 236 (1), 1–13.

9. Carlo, H. J., Vis, I. F., & Roodbergen, K. J. (2015). Seaside operations in container terminals: Literature overview, trends, and research directions. *Flexible Services and Manufacturing Journal*, 27 (2–3), 224–262.

10. Chao, S.-L., & Lin, Y.-J. (2011). Evaluating advanced quay cranes in container terminals. *Transportation Research Part E: Logistics and Transportation Review*, 47 (4), 432–445.

11. Chen, J. H., Lee, D.-H., & Cao, J. X. (2011). Heuristics for quay crane scheduling at in- dented berth. Transportation *Research Part E: Logistics and Transportation Review*, 47 (6), 1005–1020.

12. Chen, L. H., Cao, J. X., & Zhao, Q. Y. (2014). Tandem lift quay cranes and yard trucks scheduling problem at container terminals. *In Applied mechanics and materials: 505*(pp. 927–930). Trans Tech Publ.

13. Cheng, C., Petering, M. E. H., & Wu, Y. (2020). The multi-spreader crane scheduling problem: partitions and supersequences. Submitted to Discrete Applied Mathematics.

14. Choi, S.-H., Im, H., & Lee, C. (2014). Development of an operating system for optimization of the container terminal by using the tandem-lift quay crane. In J. J. Park, I. Stojmenovic, M. Choi, & F. Xhafa (Eds.), *Future information technology* (pp. 399–404). Berlin: Springer.

15. Colorado Springs Business Journal, Imports at container ports set new monthly record, accessed on May 29th, 2020, https://www.csbj.com/2018/12/12/imports-at-container-ports-set-new-monthly-record/

16. Cudahy, B. J. Box boats: How container ships changed the world, Fordham Univ Press, 2006.

17. Di Fonzo, T., Costas Paris, L. (2018). How a Steel Box Changed the World: A Brief History of Shipping. Retrieved from www.wsj.com/video/series/a-brief-history-of/how-a-steel-box-changed-the-world-a-brief-history-of-shipping/CF460889-9984-483E-AF44-324330B89ECA.

18. Ezugwu, A.E., Adeleke, O.J., Akinyelu, A.A., Viriri, S. (2020). A conceptual comparison of several metaheuristic algorithms on continuous optimisation problems. Neural Computing & Applications, 32, 6207–6251.

19. Fernández, A. (2018). Understanding Genetic Algorithms. A Use Case in the Organizational Field. Medium, Becoming Human: Artificial Intelligence Magazine, accessed on 12 Nov. 2018, becominghuman.ai/understanding-genetic-algorithms-a-use-case-in-organizational-field-2087c30fb61e.

20. Gharehgozli, A. H., Roy, D., & de Koster, R. (2015). Sea container terminals: New technologies and or models. *Maritime Economics & Logistics*.

21. Goussiatiner, A. (2007a). In pursuit of productivity. Container Management August.

22. Goussiatiner, A. (2007b). In pursuit of productivity 2. Container Management September.

23. Imai, A., Chen, H. C., Nishimura, E., & Papadimitriou, S. (2008). The simultaneous berth and quay crane allocation problem. *Transportation Research Part E: Logistics and Transportation Review*, 44 (5), 900–920.

24. 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.

25. Kim, K. H., & Park, Y.-M. (2004). A crane scheduling method for port container terminals. *European Journal of Operational Research*, 156 (3), 752–768.

26. King, R. C., Adams, G. M. & Wilson, G. L. (1936). The freight container as a contribution to efficiency in transportation. *The ANNALS of the American Academy of Political and Social Science* 187, 27–36.

27. Kite-Powell, H. (2001) Shipping and ports, Academic Press.

28. Lashkari, S., Wu, Y., Petering, M. E. H. (2017). Sequencing dual-spreader crane operations: Mathematical formulation and heuristic algorithm. *European Journal of Operational Research,* 262 (2), 521–534.

29. Levine, J. (2019). The History of the Shipping Container, accessed on June 12st, 2020, Retrieved from https://www.freightos.com/the-history-of-the-shipping-container/

30. Levinson, M. (2016). The Box: How the Shipping Container Made the World Smaller and the World Economy Bigger, Princeton University Press.

31. Liu, C.I., H. Jula, K. Vukadinovic, and P.A. Ioannou. Comparing Different Technologies for Containers Movement in Marine Container Terminals. Proc. *3rd IEEE International Conference on Intelligent Transportation Systems*, pp. 488–493.

32. McCarthy, P. W., Jordan, M. A., & Wright, L. (2007). Dual-hoist, tandem 40 crane considerations. *Port Technology International*, 34, 111–113.

33. 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.

34. 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.

35. Ng, W., & Mak, K. (2006). Quay crane scheduling in container terminals. *Engineering Optimization*, 38 (6), 723–737.

36. North, D. C. (1968). Sources of productivity change in ocean shipping, 1600-1850. *Journal of Political Economy,* 76, 953–970.

37. Shipping Container History: Boxes to Buildings. (2020), accessed on June 22st, 2020, Retrieved from https://www.discovercontainers.com/a-complete-history-of-the-shipping-container/

38. Song, J.-H. (2011). Tandem operation and double cycling in container terminals. *Port Technology International*, 51, 73–79.

39. Stahlbock, R., & Voß, S. (2008). Operations research at container terminals: a literature update. *OR Spectrum*, 30 (1), 1–52.

40. Statista, Capacity of container ships in seaborne trade, accessed on June 18th, 2020, https://www.statista.com/statistics/267603/capacity-of-container-ships-in-the-global-seaborne-trade/

41. Steenken, D., Voß, S., & Stahlbock, R. (2004). Container terminal operation and operations research-a classification and literature review. *OR spectrum*, 26 (1), 3–49.

42. Suman, B., and Kumar, P., (2006). A Survey of Simulated Annealing as a Tool for Single and Multi-Objective Optimization. *The Journal of the Operational Research Society*, 57, 1143 – 1160.

43. Talley, W. K. (2000). Ocean container shipping: impacts of a technological improvement. *Journal of economic issues* 34, 933–948.

44. Tang, L., Zhao, J., & Liu, J. (2014). Modeling and solution of the joint quay crane and truck scheduling problem. *European Journal of Operational Research*, 236 (3), 978–990.

45. UNCTAD Stat (2020a), Liner shipping connectivity index, accessed on June 11th, 2020, https://unctadstat.unctad.org/wds/TableViewer/tableView.aspx?ReportId=92.

46. UNCTAD Stat (2020b), World seaborne trade by types of cargo and by group of economies, annual, accessed on June 19th, 2020, https://unctadstat.unctad.org/wds/TableViewer/tableView.aspx?ReportId=32363.

47. University press of Liverpool. (1954) The Dock Worker: An Analysis of Conditions of Employment in the Port of Manchester.

48. Unsal, O., & Oguz, C. (2013). Constraint programming approach to quay crane scheduling problem. *Transportation Research Part E: Logistics and Transportation Review*, 59, 108–122.

49. Van Ham, H., Rijsenbrij, J. (2012). Development of containerization: Success through vision, drive and technology, IOS Press.

50. Vis, I. F., & De Koster, R. (2003). Transshipment of containers at a container terminal: an overview. *European Journal of Operational Research*, 147 (1), 1–16.

51. World Cargo News (2007). Make mine a double – or even a triple. Wu, Y., Li, W.-K., Petering, M., Goh, M., & de Souza, R. (2015). Scheduling multiple yard cranes with crane interference and safety distance requirement. *Transportation Science*, 49 (4), 990–1005.

52. World Shipping Council, (2020a). Partners in Trade (n.d.), accessed on June 22st, 2020. Retrieved from http://www.worldshipping.org/about-the-industry/containers.

53. World Shipping Council, (2020b). The birth of intermodalism, accessed on June 21st, 2020. http://www.worldshipping.org/about-the-industry/history-of-containerization/the-birth-of-intermodalism.

54. Wu, Y., Li, W., Petering, M. E. H., Goh, M., de Souza, R. (2015). Scheduling multiple yard cranes with crane interference and safety distance requirement. *Transportation Science*, 49, 990–1005.

55. Xing, Y., Yin, K., Quadrifoglio, L., & Wang, B. (2012). Dispatch problem of automated guided vehicles for serving tandem lift quay crane. *Transportation Research Record*, (2273), 79–86.

CURRICULUM VITAE

Shabnam Lashkari

Education

PhD in Industrial Engineering | GPA: 3.82, University of Wisconsin Milwaukee 2014 – 2020

- **Thesis:** Sequencing multiple-spreader crane operations: mathematical formulation and heuristic algorithm
- **Advisor:** Dr. Matthew E.H. Petering | **Minor:** Business Administration

BSc. in Industrial Engineering | GPA: 3.35, Iran University of Science & Technology 2012

Career Summary

Senior DevOps Engineer, CCC Information Services September 2019 – Present

- Development of multi-staged pipeline
- Automation of data ingestion that prepares raw data for training models, resulting in the creation and deployment of intelligent Deep Learning based solutions
- Automation of monitoring systems across on-premise GPUs, AWS and Oracle Cloud services
- Infrastructure architecture design in AWS enterprise account

DevOps Intern, CCC Information Services July – September 2019

- Developing monitoring system and integrating the UI with AI-Pipeline used across the Architecture division
- Automation of scheduling big data download and pre-processing using Airflow and Python

Operations Research Analyst, University of Wisconsin Milwaukee 2014 – August 2020

- **Operations Research**
  - Sequencing Multiple–Spreader Crane Operations: Mathematical Formulations and Heuristic Algorithms: Devised innovative strategies for scheduling a multiple-spreader quay crane; developed mathematical models, determined optimal objective value, and designed simulated annealing and genetic heuristic algorithms.
- **Data Analytics**
  - Predicted average time to crime across 50 states by modeling and analyzing 'time to crime' data. Predicted the impact of variable values by performing multiple regression analysis and using Quantile and Ridge regression models.
- **Machine Learning**
  - Predicted arrival and departure delays in national flights by modeling the "flight data" extracted from Bureau of Transportation using machine learning methods such as logistics, decision trees, random forest, clustering, SMO, neural networks and ensembles using Weka software.
- **Supply Chain Management**
  - SAP-ERP Network Modeling and Analysis Improved revenue for a manufacturing company by defining business intelligence strategies, analyzing data collected from SAP and produced 6 types of serials in the market with multiple vendor suppliers.
- **Non–linear Optimization**
  - Optimized inventory placement cost in a supply chain network by using non-linear optimization techniques to minimize transportation cost resulted in 25% cost reduction.

Instructor & Teaching Assistant, University of Wisconsin Milwaukee 2014 – 2019

- Collaborated with Rockwell Automation Company to teach a course on IoT. Brainstormed new solutions, designed equipment, and aided in time study and analysis for the Methods Engineering lab.
- Teaching the course 'Intro to Operations Analysis' as an instructor to a class of 27 undergraduate students.
- Taught ProModel on real-world manufacturing and service system problems in the simulation methodology lab.
- Recognized as the best TA in the IME Department for bringing interactive learning opportunities for students.

Supply Planning Analyst, Kalleh Co. 2012 – 2013

- Established an optimal inventory plan by forecasting demand for 1000+ products. Oversaw supply planning and developed a predictive model based on time series analysis. Presented monthly data reports by analyzing sales records.

Data Analyst Intern, ArmanSanat Company Summer 2009

- Facilitated change in management strategies for improved innovation appreciation and feedback system implementation across 3 companies. Evaluated creative management methods, designed questionnaires, and collected data for analysis.

Technical Skills

Industrial Skills: Linux**,** Docker, Kubernetes, Prometheus, Grafana, Airflow, AWS Architecture Design, Packer, Terraform, Ansible, Database Support, OCI, Git, Apache, SQL

Academic Skills

- Programming: C++, Python
- Industrial tools: CPLEX, ProModel, SAP, Tableau, Weka, Minitab, Mixed Integer Programming, LP, NLP, Non-Linear Optimization, Data Modeling, Data mining, Statistical & Predictive Analysis, Machine Learning, Algorithm Design and Complexity Analysis, Simulation Modeling, Supply Chain Technology & Simulation
- Tools: Advanced Excel, Powerpoint, Word, Access

Publications

- **Lashkari**, S., Wu, Y., Petering, M.E.H. (2017) "Sequencing dual-spreader crane operations: mathematical formulation and heuristic algorithm," *European Journal of Operational Research*, 262(2), 521-534.
- Petering, M.E.H, **Lashkari**, S., Wu, Y. "Sequencing triple-spreader crane operations: mathematical formulation and heuristic algorithm," *European Journal of Operational Research* [Under review].

Honors & Awards

- Graduate Student Excellence Fellowship (top 1% graduate student), UWM Graduate School 2018
- Honorable Mention Award in Poster Competition, UWM College of Engineering & Applied Science 2018
- Finalist in 3-Minute Thesis Competition, UWM Graduate School 2018
- Graduate Student Travel Award, UWM Graduate School 2017
- 3-Times Chancellor's Award Winner, UWM Industrial & Manufacturing Department 2015 – 2018
- Ranked best teaching assistant in IME department, UWM 2016 – 2017