Mathematical Modelling and an Optimisation Approach to Container Terminal Yard Management Operations

Hammed Oladiran Bisira

A Thesis Submitted for the Degree of

Doctor of Philosophy

Department of Mathematical Sciences

University of Essex

October, 2020

Dedication

To my wife Olabisi and our children Oladapo and Oladayo for their love, great support and encouragement.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my supervisor Prof Abdel Salhi for the continuous support of my Ph.D study and related research, for his patience, motivation and immerse knowledge. His guidance helped me in all the research and writing of this thesis. I could not have imagined having a better supervisor for my Ph.D study. I would also like to thank my board chairman Dr Spyridon Vrontos for guiding my research for the past years and for his suggestions.

I am grateful to Tom Corkhill and Guijarro-Rodriguez Alberto of the Port of Felixstowe for their unrelenting support and advice throughout the course of my research. Their input has contributed in no small way to the completion of this thesis.

I would like to thank my sponsors Lagos State Polytechnic and Tertiary Education Trust Fund for their financial support as a beneficiary of Academic Staff Training and Development (Ph.D programme).

To my friends Dr Felix Ngobigha, Cornelius Saiki, Dr Olaniyi Olayiwola and Olumide Ajala I say a big thank you.

Abstract

In container terminals, Yard Cranes (YCs) work at the interface between the storage yard and the internal and external trucks. A delay in the operations of the YCs will affect the overall operations of the container port. There is thus need for a good and reliable planning and scheduling of this resource for an effective day to day operation. Commonly, this is dealt with in an ad hoc way because of its complexity. A systematic approach for their planning and deployment is therefore required.

The first focus of this thesis is on developing a mathematical model, the solution of which will minimise unfinished work in the yard at the end of a planning period by allocating and changing the YCs movements between yard blocks at different times. We propose two models for the Yard Crane Scheduling Problem (YCSP). In the first model, we introduced additional set of constraints to improve the performance of an existing model. The second model has the objective function of minimising the linear combination of unfinished work and surplus capacity. The problem is formulated as a mixed integer linear programming model, real world instances of which are solved in near-real

time. This will help port managers to generate plans for YCs before the start of planning periods.

Furthermore, this thesis examines the container reshuffle problem as a refinement of the scheduling problem. Once the YCs have been allocated to a bay, the reshuffle problem is defined and solved. So, the overall efficiency problem is solved in two stages, allocating YCs and scheduling each one of them using reshuffles to deal with the local load at the level of a bay. Four new heuristics referred to as the Least Priority Heuristic, LPH1, LPH2, LPH3 and LPH4 were developed to solve the reshuffle problem for a realistic size problem as they arise in the port for both static and dynamic cases. A compatibility test was proposed to test how well the different heuristics work when they are combined.

Declaration

The work in this thesis is based on research carried out at the Department of Mathematical Sciences, University of Essex, United Kingdom. No part of this thesis has been submitted elsewhere for any other degree or qualification, and it is all my own work, unless referenced, to the contrary, in the text.

Acronyms

AOC	Area of Coverage
B&B	Branch and Bound
B&C	Branch and Cut
BIP	Binary Integer Programming
CRP	Container Reshuffle Problem
GA	Genetic Algorithm
GLPK	Gnu Linear Programming Kit
GLPSOL	Gnu Linear Programming Solver
GMPL	Gnu Mathematical Programming Language
LPH	Least Priority Heuristic
MILP	Mixed Integer Linear Programming
NP	Non-deterministic Polynomial
PIP	Pure Integer Programming
PoF	Port of Felixstowe
PSO	Particle Swam Optimisation
QC	Quay Crane
QCR	Quay Crane Rate
RI	Reshuffle Index
RMG	Rail Mounted Gantry
RTG	Rubber Tyred Gantry
SA	Simulated Annealing
TEU	Foot Equivalent Unit
TS	Tabu Search
TUM	Total Unimodular Matrix
YC	Yard Crane
YCSP	Yard Crane Scheduling Problem

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Background

The increase in international trade has made shipping the center of the global economy. Almost all overseas shiping of goods such as clothing, auto parts, electronics, furniture, toys, fruits, computers and other equipments is done through container shipment. According to [36], since the introduction of the container in April 1956 by Malcolm Mclean who moved fifty-eight 35-foot containers from Newark to Houston in America by a refitted oil tanker, container flows have increased continuously.

The increasing number of container shipments causes higher demands on the seaport container terminals, container logistics and management, as well on the technical equipment. An increased competition between seaports, especially close ones, is a result of this development. The competitiveness of a container

terminal is marked by different success factors, particularly the time a vessel spends at berth. The faster containers are loaded and unloaded unto or from vessels, the lesser the time spent at the port. This invariably reduces the turnaround time of vessels and increases the Quay Crane Rate (QCR).

A container is a steel metal box that comes in two different standard dimension 20-foot Equivalent Units called (1 TEU) and 40-foot equivalent units referred to as (2 TEUs or FEU). There are also 30-foot and 45-foot containers depending on the configuration of the yard.

The port container terminal management can be divided into three major operations: the quay side, the internal truck (IT) and the yard operations. Vessels that arrive at the port are allocated space to berth (dock) after which, Quay Cranes (QCs) are assigned to unload or load containers onto or from ITs. Loading and unloading are two separate activities, in which the former takes precedence. The ITs move the containers to the yard where Yard Cranes (YCs) are assigned to unload them and eventually store them in the yard area. The reverse is the case for export where the external trucks (ETs) bring in containers and are stored by YCs; Figure 1.1 shows a typical flow of operations in a container terminal. Note that we use YC and RTG (Rubber Tyred Gantry) to mean the same.

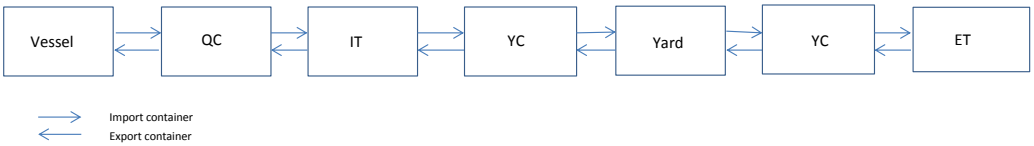


Figure 1.1: Container flow in a Container Terminal

The yard is a temporary holding area for containers between the time they arrive at the port and the time they are eventually moved out of the gate in case of import and onto the ships for export. The yard is divided into zones which are further sub-divided into blocks. A block holds an average of twenty to forty containers in length called bays with about five to seven containers in width called rows or columns and height of four to six containers called tiers or stacks. This depends on the configuration of a yard and its size. The height of a container stack depends on the type of RTG used in the yard. The blocks maintain equal distance between them to allow for both RTG and truck movement. Figure 1.2 shows a block with four tiers, six rows with four bays.

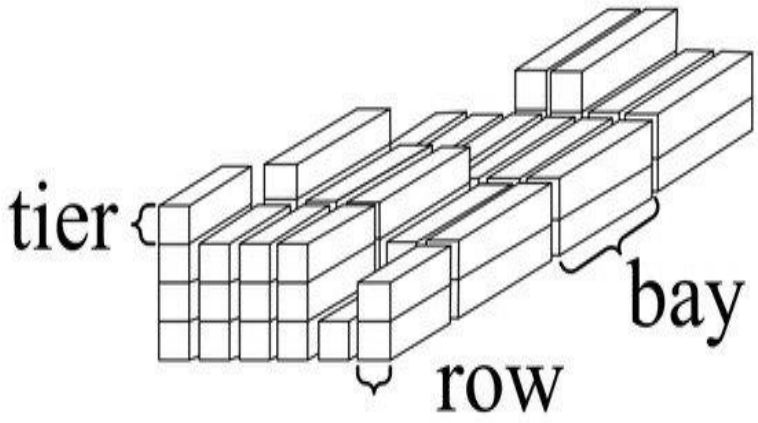


Figure 1.2: A typical block, (*Source*, Wang et.al 2019)

RTG operations can be divided into storage, reshuffle and retrieval. While storage and retrieval are regarded as productive activities, reshuffle is not. They need to move from one area of a block/zone to another in the course of their operations. The efficiency of these operations at the port especially at the storage yard level is greatly affected by how port managers are able to handle RTG scheduling and container reshuffle. In this thesis, the overall efficiency of

the yard management is improved in two steps;

- Allocation of RTGs to best positions from initial ones.
- Scheduling each RTG using minimum reshuffles to deal with the local load (at the level of a bay).

1.2 Overview of operations in Port of Felixstowe

Container terminals can be described as open systems of material flow with two external interfaces, these interfaces are the quayside with loading and unloading of ships, and the landside where containers are loaded and unloaded on/off trucks and trains. Containers are stored in stacks in the yard area facilitating quayside and landside operations.

Upon arrival at the port, a vessel is assigned a berth equipment with quay cranes to service the vessel. These berths are assigned according to several criteria such as berth depth, length, availability and etc. Furthermore, QCs will unload and load the containers according to QCs work schedule. The containers are then picked up by the ITs (IMV) and distributed to the respective locations in the yard. Additional moves are also performed for housekeeping, transit shed, empty depot and etc. In the port of Felixstowe, the yard strategy team is responsible to effectively use the yard area for both import and export containers.

Figure 1.3 shows the sea-side, QCs attending to vessels, the IT and the yard

area with different zones and RTGs.

Figure 1.3: Typical Container Terminal (*Source*, Port of Felixstowe)

Based on the plan of ship arrival (BPOS) and the discharge amount of each vessel (other criteria such as size, feeder, etc. are also considered) the yard strategist allocate certain number of blocks to each vessel. Additionally, the Dynamic Import Grounding (DIG) system dispatches the imports to these locations. The DIG system allocates blocks (stack) to each import container in order to minimise the load of each block (other criteria are also taken into account). One of the problems with this method is the fact that the DIG system can not send the import containers to the exact blocks selected, it rather dispatches containers to multiple blocks which might be the one assigned to another vessel. This could sometimes cause cross contamination where IMVs move into blocks with more distance to the berthing location of the vessel.

In the case of export containers, the vessel planning team creates a vessel loading programme which consists of the locations of the containers in vessels as well as in quay crane schedule for loading these containers. The export locations are decided based on vessel service slot. For smaller vessels, export is spread into multiple zones and a single zone is not selected. Yard strategy team also has an eye on how to enable multi hook feeding to a single crane. This is the process where several RTGs feed one single quay crane.

Different type of ships has to be served at berths including deep-sea vessels with a loading capacity of up to 11,000 TEU to smaller feeder vessels for regional transshipments.

1.3 RTG operations

RTGs are relatively expensive equipment in the storage area. The efficiency of yard operations is often related to the productivity of these RTGs and their efficiency. It is a critical resource, whose performance in the storage yard affects waiting time of XT (hauliers), IMVs (tugs) and QCs. The waiting time of vessels is also indirectly affected by the productivity of RTG.

In the container yard, RTGs operate in the areas blocks. In every time period a RTG is involved in moving containers belonging to various type of jobs. These include:

- Shipping jobs (import/export, restow)

- Hauliers
- Rail jobs
- Shuffling (in-stack shuffle, trinity to berth 8.9 shuffle)
- Examination, X-ray, Warehousing

Each of these jobs can have a level of priority and importance that needs to be dealt with. Currently shipping jobs have the highest priority followed by rail jobs and all other jobs have the lowest priority.

The handling of jobs and allocations of the RTG operations can be categorised as follows:

- Import jobs. Once discharge locations are selected the import containers are sent to these locations. The DIG system ensures containers are sent evenly to several RTGs in order to balance their workload. However, the yard planners do not have control on where the discharged container will be loaded.
- Export jobs. Once the vessel planner creates the loading sequences and crane schedules, the plan is then executed by vessel controllers who then load the jobs to the system. These jobs then become visible to park management system (PMS). The vessel planner makes sure these jobs are being kept into minimum level, this means the jobs are visible to the PMS are in time horizon of an hour or so.

- Haulier jobs. The RTG will serve external trucks arriving at port for unloading export containers and collecting import containers. The RTG controller has control over the maximum number of hauliers in each block. If a haulier arrives and has to go to a specific block and the blocks maximum number of haulier has been reached, the system would guide the haulier to the Temporary Holding Area (THA). This process is done to avoid congestion in each block especially during high load of shipping jobs.

1.4 RTG Deployment

A yard management tool used at the Port of Felixstowe to schedule RTG movement is called Area of Coverage (AOC) Planner. It is designed as a simple tool with visual display of areas of the yard where the RTG deployment requires attention. The planner collects data and calculates where and when moves are required and also plans the best positions for RTGs within the yard. It has built in area of cover change, RTG routing and detailed views of current and future yard operations.

The main purpose of the planner is to assign RTGs to zones based on the volume of work in each zone and the urgency of the tasks. This deployment is based on workload distribution between the blocks. The RTG controller will update the planner every 500 seconds or less. It is expected that using the AOC planner will reduce idle times and increase yard productivity and other service

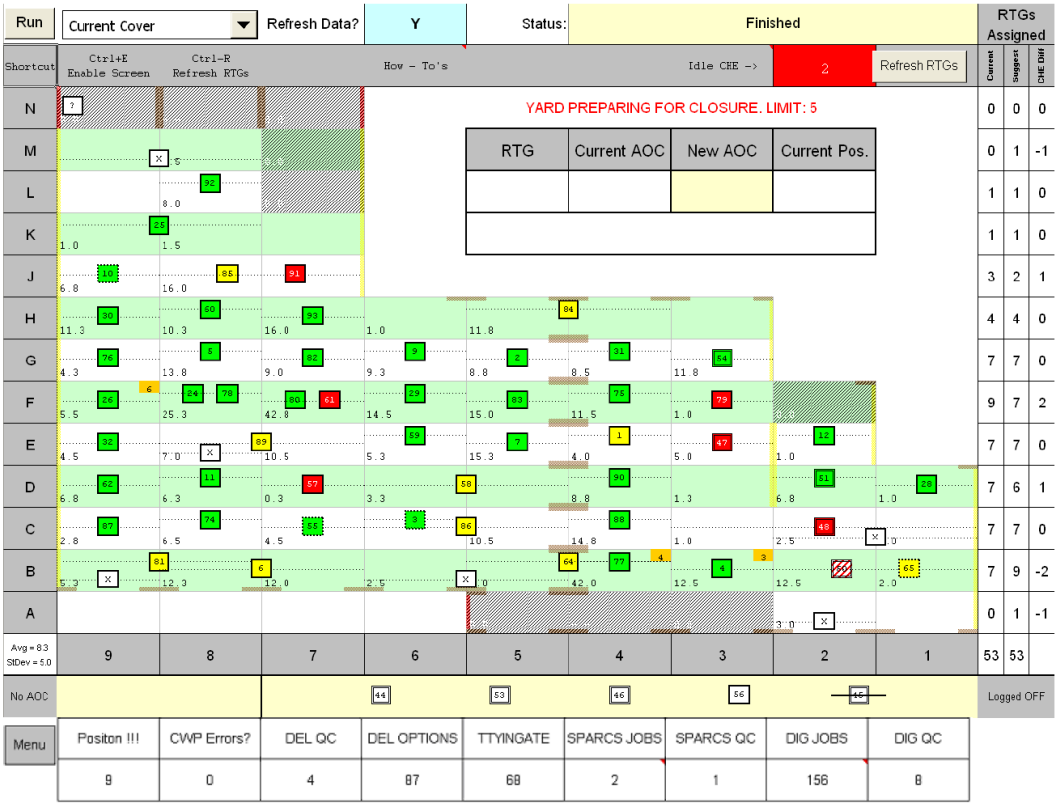
levels across the yard. Figure 1.4 shows the AOC planner.

The square shaped boxes are the RTGs with their ID numbers. The green colour indicates that the RTG remains there, the white requires an RTG in that position. The red colour boxes indicate RTGs that are surplus to requirement and the yellow colour boxes indicate RTGs that can be changed by the controller as suggested by the scheduler. Double clicking on a zone will reveal the jobs in that zone. Left clicking on an RTG while holding the shift button and right clicking on a desired zone will give the best route for a RTG to a given destination. The letters $A - M$ represent the rows in the yard i.e 12 rows. A block/zone can be identified by the zone number and lane e.g $09B$ is the first zone in row B from left to right. The zones with grey shades represent areas where there are no activities; hence, they are not assigned RTGs.

The scheduler works by refreshing the RTG button and clicking run. This will update (load) the work for the zones and allocate RTGs based on the workload distribution. By left clicking on a yellow RTG, it gives information as to the present position of the RTG and the suggested position. Clicking the update button will move the RTG to the suggested position.

The planner however does not indicate the unfinished work in each block at the end of the periods and needs to be refreshed after every run. This is not acceptable as work arises dynamically and in real time, hence decisions have to be taken instantly. The time used in loading and running the planner can potentially cause a delay in decision taking. Ideally, a solver should not only schedule the movements of the RTGs but also indicate their efficiencies in terms

of unfinished work.



MPH->	15.62			14	Manual Yard Blockages		Closed Zones (e.g 02F)		Manual Min CHE (To Cover Sparcs)	
StDev->	4.99									
Solve Group	Jobs	Covered VWLoad	Zones to Cover	Unique Zones	From Row	To Row			Zone	MinChe
1	39	96	7	7				03A		
2	30	55	5	5			07M	04A		
3	23	42	2	0			07N	05A		
4	20	21	3	0						
5	17	32.5	1	1						
6	80	52	18	15						
7	6	5	3	1						
8	32	12	4	1						
9	0	0	0	0						
10	0	0	0	0						
11	38	21	12	6						
12	0	0	0	0						
13	62	11.25	18	3						
14	39	7.75	13	0						
15	72	7	9	3						
16	5	0.03	2	0						

Figure 1.4: Area of Coverage Planner (*Source*, Port of Felixstowe)

1.5 Motivation

In a container terminal, RTGs serve as an interface between the trucks and the storage yard. There is thus the need for a good and reliable planning and scheduling of this scarce resource for an effective day-to-day operation. The coordination of these inter-related operations in a container terminal is very complex and as such requires serious planning. These operational problems are known to be NP-hard [91]. What we strive to achieve is near-optimal or a good solution since optimal ones are not guaranteed. Commonly, this problem is solved in an *ad hoc* way at the port because of its complexity.

This thesis studies the problem of RTG scheduling in a yard zone to minimise the unfinished work at the end of a planning period. Our model improves upon that of Linn and Zhang [72] which aims at finding the optimal frequency and routing of RTGs at the yard. The objective is to determine the best deployment period that achieves the minimum unfinished work. Our models will determine the number of RTGs that minimise the unfinished work in a planning period and minimise the number of reshuffles.

1.6 Research Objectives

The objective of this thesis is to investigate good models for the scheduling of RTG operations and container reshuffle in a container terminal port. Identify important factors that are necessary to build the models. The schedulers will

be able to use minimal number of RTGs to finish a given work in the shortest time using minimum reshuffle. The focus will be to reduce the unfinished work at a container terminal yard by allocating and changing the RTG movement between the blocks at different times using minimal resources. In order to achieve the set objectives, we aim to;

1. Improve on the Integer Programming models and solve using an exact approach.
2. Use heuristics, existing and new where exact methods do not work.
3. Validate our model with what is done specifically at the Port of Felixstowe.

1.7 Research Methodology

Review and understand what has been done so far in managing yard storage in container ports. Link this technology to what is in use currently at the PoF. Involve PoF staff dealing with the yard management and preceding problems such as the berthing of ships, allocation of QCs and there scheduling in this problem. This endeavour should build on the current links between PoF and the Department of Mathematical Sciences. Build a mathematical model that captures the situation in PoF for RTG scheduling and container reshuffle problem. Solve realistic problem instances generated from real data provided by the port. Solve with existing algorithm and software technology [105], if appropriate. Else, use a new approach such as heuristic to solve the problem. Evaluate

against alternative approaches and get an evaluation from the operation at port.

1.8 Contribution

Our models on RTG scheduling are more suited to real world problems and as such has the following advantages over the previous models;

1. Model1 gives a better RTG deployment decision when the workload distribution in the blocks are unstable within the planning horizon.
2. Model1 results in reduced computational time compare to L&Z model. This will make for fast decision making and ultimately a reduction in operational cost.
3. Model2 has an objective function that is a linear combination of unfinished work and the surplus capacity which has never been considered in literature.

The Least Priority Heuristic (LPH) for the container reshuffle problem has the following advantages over others;

1. Four new and effective heuristics LPH1, LPH2, LPH3 and LPH4 are introduced.
2. Compatibility test between the different heuristics has been proposed.

1.9 Thesis Organisation

This thesis is divided into six chapters, the remaining chapters are organised as follows;

Chapter 2 gives detail definitions of the terminology and concepts used in the research work. We start by describing computational complexity in optimisation problems. Then a brief description of Mixed Integer Programming and LP - relaxation is given. In addition, we reviewed solution approaches to solving these problems, the exact and approximate techniques. Some of these methods such as the B&B, DP and Branch & Cut were explained. We also review heuristics as an alternative to exact methods when the problem size gets prohibitive.

In Chapter 3, we have a comprehensive review of literatures relating to the study. The different operational problems in the container terminal are mentioned including, yard layout and type of equipments. However, our interest is on yard management operations especially YC scheduling problem (YCSP) and container reshuffle problem (CRP).

Chapter 4 presents the YCSP. We start by presenting the assumptions and description of the problem. The Linn and Zhang model in [72] was examined and solved using MATLAB MILP solver, `intlinprog` to get an idea of the tractability of the problem. Thereafter, we introduce an improved model of YCSP Model1 and solve it using GLPSOL. Computational experiments were conducted on the YCSP model for different yard sizes. Comparison between

Model1 and the previous model is presented. We also introduced Model2 which include surplus capacity in the objective function. A comparison was also made between L&Z, model1 and model2 in terms of the unfinished work and the surplus capacity..

In Chapter 5, we examine the Container Reshuffle Problem as an enhancement to YCSP. Once RTGs have been assigned to a block we investigate their operations at the bay level to improve the RTG schedule. We start by modifying the model in [95] and obtain exact solutions for small scale instances. For approximate solutions, we present the framework for three heuristics in [102] and [95]. A description of the heuristics, Reshuffle Index, H1 and H2 is given. Four new heuristics, Least Priority Heuristic LPH1, LPH2, LPH3 and LPH4 were introduced and developed with illustrations for static and dynamic cases. Computational experiments were conducted to compare the model using B&C and the seven heuristics for small and medium size instances. The model could not generate result for large instance problem so, only the heuristics were compared for large size. The heuristics were also tested for compatibility. The four new heuristics were compared with the three previous and they seem to be competitive.

Chapter 6 highlights the conclusions and further work arising from the thesis. The main contributions of the thesis is also highlighted, certain limitations of the work are identified, and areas for possible further study are suggested.

Chapter 2

Optimisation Problems and Solution Approaches

2.1 Introduction

An optimisation problem is that of finding the best possible solution for a given problem under given sets of constraints. Except in some particular cases, this is achieved using mathematical methods. Three types of approaches to solve optimisation problems are discussed; exact methods, approximation algorithms and heuristic/metaheuristic methods. Exact methods guarantee optimality. However, they often involve a lot of computational power in terms of storage space and time. Approximation algorithms tend to be computationally less expensive but do not guarantee optimality. They try to generate solutions within predefined bounds, which may exclude the optimum. They usually

have strong mathematical underpinnings. Heuristic/metaheuristic approaches have become popular in the last few decades since exact methods are often inconvenient for large real-world problems. Unlike approximation algorithm, heuristics search for solutions within no specific bounds of the optimal solution. These generally do not guarantee to find the optimum solution. But, they are usually able to find near-optimum solutions in reasonable computational times.

2.2 The Optimisation Problem: General Definition

Optimisation is the operation of maximising or minimising an objective function possibly subject to constraints. The mathematical representation of such a problem is

$$\text{maximise / minimise} \qquad f(\mathbf{x}), \qquad \mathbf{x} = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n \tag{2.3}$$

$$\text{subject to} \qquad \lambda_k(\mathbf{x}) \leq 0, \qquad (k = 1, 2, ..., N), \tag{2.4}$$

where $f(\mathbf{x})$ and $\lambda_k(\mathbf{x})$ are real-valued functions of $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n$.

In (2.3), vectors $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ are the decision variables. $f(\mathbf{x})$ is called the objective function and $\lambda_k(\mathbf{x}) \leq 0$ is called the inequality constraint. The feasible region/search space consists all $\mathbf{x}$ points that satisfy all the constraints, [84,110]. Where the objective function and all the constraints are linear, we have a linear programming problem otherwise it is non-linear. Feasible solution attaining

the maximum/minimum is called the optimum solution. The YCSP models in Chapter 4 and the reshuffle model in Chapter 5 are cases of minimisation.

Decision variables can be continuous, discrete or mixed i.e. both real and integer points. In continuous optimisation problems the decision variables take real values, and in the discrete ones, they take integer values. If the optimisation problem does not have any constraints, it is called an unconstrained optimisation problem, otherwise, it is constrained. The simplest unconstrained optimisation is probably the search for the maxima or minima of a function. In optimisation since we are interested in finding the best solution, it is important to know how far we are from the optimum.

Definition 2.2.1 Let a real valued function f be defined over a feasible set $S \subset \mathbb{R}^n$. The solution $\mathbf{x}^*$ is said to be in the neighbourhood of all solutions which satisfy $|\mathbf{x} - \mathbf{x}^*| < \epsilon$, [38]. It is a movement in any direction away from a point.

For continuous optimisation problems, ϵ is usually a small positive number larger than 0. For combinatorial problems, for instance, it can be defined as the number of changes in the permutation, [84].

The primary goal in solving an optimisation problem is to find a global optimum solution. Hence, we need to know if the present/local optimum is also the global. Let us define those terms here.

Definition 2.2.2 Let a real valued function f be defined over a set $S \subset \mathbb{R}^n$. A solution $\mathbf{x}^*$ is said to be a global maximum if $f(\mathbf{x}^*) \geq f(\mathbf{x})$, $\forall\ \mathbf{x} \in S \subset \mathbb{R}^n$. If

$f(\mathbf{x}^*) \leq f(\mathbf{x})$, $\forall\ \mathbf{x} \in S \subset \mathbb{R}^n$, the solution $\mathbf{x}^*$ is said to be a global minimum, [38].

Definition 2.2.3 Let a real valued function f be defined over a set $S \subset \mathbb{R}^n$. A solution $\mathbf{x}^*$ is said to be a local maximum if $f(\mathbf{x}^*) \geq f(\mathbf{x})$, $\forall\ \mathbf{x} \in N(\mathbf{x}^*, \epsilon)$. If $f(\mathbf{x}^*) \leq f(\mathbf{x})$, $\forall\ \mathbf{x} \in N(\mathbf{x}^*, \epsilon)$, then $\mathbf{x}^*$ is said to be a local minimum, where $N(\mathbf{x}^*, \epsilon)$ denotes the ϵ-neighborhood of $\mathbf{x}^*$, [38].

Note, a global optimum must necessarily be the local optimum. The converse is not always true.

2.2.1 Integer Linear Programming

Real world problems are hard to formulate as Linear Programs due to the impossibility of fractional decision variables as opposed to the assumption of divisibility in LP. While most decision variables in practise are integer values, LP on the other hand assumes decision variables to be continuous. An example is the number of cars a company produce or number of students to allocate to a classroom for examination purpose. Clearly, both decisions can not have fractional values since car or student are indivisible. An optimisation problem where all the variables are integer is referred to as Pure Integer Programming (PIP). In a case where the integer variables are restricted to only 0 or 1, we have Binary Integer Programming (BIP).

BIP reduces a problem to a "yes-no" or a "do-don't" scenarios where the variable is 1 when the decision is "yes or do" and 0 when it is "no or don't". An example is modelling the decision to reshuffle a container by an RTG as

follows. The variable y_{si} is 1 when container i is reshuffled at stage s and 0 when container i is not reshuffled. Another example is whether to build a fire station or not in a location; a typical set covering problem. The container reshuffle treated in Chapter 5 is modelled as a BIP problem and solved exactly using the B&C and heuristics.

In Mixed Integer Linear Programming (MILP), some variables take integer values while others take continuous values. The case of YCSP fits into the class of these problems because while unfinished work can take continuous value, the number of RTG must necessarily be integers and there are binary variables as well. The problem can generally be formulated as follows:

$$\text{minimise} \quad \mathbf{c}^T\mathbf{x} + \mathbf{d}^T\mathbf{y} \tag{2.5}$$

$$\text{subject to} \quad \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} \leq \mathbf{b} \tag{2.6}$$

$$\mathbf{x} \geq 0 \quad \text{or} \quad \mathbf{x} \in \mathbb{R}^n_+ \tag{2.7}$$

$$\mathbf{y} \in \mathbb{Z}^m_+ \tag{2.8}$$

where $\mathbf{x}$ is an n-dimensional vector of continuous decision variables and $\mathbf{y}$ is an m-dimensional vector of integer decision variables. $\mathbf{c}$ and $\mathbf{d}$ are cost vectors. $\mathbf{A}$ and $\mathbf{B}$ are constraint matrices for continuous and integer variables respectively. $\mathbf{b}$ is a p-dimensional column vector of constants. MILP can also be loosely referred to as MIP (Mixed Integer Programming), ILP (Integer Linear Programming) or simply IP (Integer Programming). We also have the MINLP (Mixed Integer Nonlinear Programming) which has either non-linear objective

function or constraints, or both.

2.2.2 LP Relaxation

The first step in solving an MIP is to disregard the integer restriction on the decision variables. This is called the LP relaxation of the original problem. The convex hull of feasible solutions to an MILP is a subset of the convex hull of the LP for the same problem i.e $Z_{MILP} \leq Z_{LP}$. Hence, the closer feasible set of possible solutions of the MILP is to the convex hull of the LP, the faster the solution is generated for any algorithm. This is very useful in determining the Integrality gap i.e how far the best MIP solution is from the LP solution. As the gap gets smaller, the closer we are to realising the best MIP solution. Hence, we achieve what we refer to as strong LP relaxation, see Section 2.4.1.1 for more explanation. The relative MIP gap is calculated in [75] as follows;

$$\frac{|\text{best\_MIP} - \text{best\_bound}|}{|\text{best\_MIP}| + \epsilon} \tag{2.9}$$

Where **best_MIP** is the best integer feasible solution found so far and **best_bound** is the best or global bound.

2.3 Complexity and NP-Completeness

The time complexity of an algorithm is the number of operations it requires in the worst-case to solve a problem of a given size. $Big - O$ notation is used to represent the dominant aspect of the required computation. If the argument of the O notation is a polynomial function p, the algorithm is said to be "efficient". Otherwise, it is "inefficient" and the problem is intractable [33].

A problem is referred to as tractable if a polynomial-time algorithm can solve it. Otherwise, it is said to be intractable; in other words, no polynomial-time algorithm can solve the problem. In Chapter 4, we checked for the intractability or otherwise of the YCSP using MATLAB integer Linear Programming solver `intlinprog`. However, in practice, some non- polynomial time algorithms are known to work well on small size instances.

A problem is in the nondeterministic polynomial (NP) class, if the answer to its decision problem form can be verified in polynomial time. There is another class of problems called NP-hard. They do not necessarily have to be in NP and they do not have to be decision problems. A problem is said to be in NP-hard class if it cannot be solved in polynomial time and also another problem in NP-complete class can be reduced to it in polynomial time. A problem is said to be in NP-complete class, if it is in both NP-hard and NP-class and all other problems in NP-class can be reduced to it in polynomial time [33]. The YCSP has been proved to be NP-hard [8] and mathematical models solutions are only benchmark obtained through exact methods. The alternative is to

use heuristics methods due to the size and complexity of real world problems. Even where a single YC is scheduled to work in a section of a block, each of these single machine scheduling problem has been shown to be NP-hard [12]. Therefore, the multiple YCSP is also NP-hard.

2.4 Solution Approaches to Optimisation Problems

These problems are combinatorial in nature and require near real time solution in order for their outputs to be used in the decision process. There are a number of exact methods of solution for this type of problem, B&B is one of them. However, a fairly recent hybridisation of B&B with the Cutting Plane Algorithm has led to a more efficient approach. It is implemented in the GLPK LP/MIP Solver (GLPSOL), Version 4.57 which solves large-scale LP and MIP problems. The algorithm works by first generating Cutting Planes as in Gomory's algorithm, MIR (Mixed Integer Round), Cover or Clique cuts which remove a chunk of the feasible region of the LP relaxation of the given problem, before solving the remaining MILP using B&B. To solve the ILP problems formulated in this thesis, we make use of GLPSOL. The reader is referred to [75].

In this section, exact methods, approximation algorithms and heuristic methods will be reviewed next.

2.4.1 Exact Methods

Exact methods are used to find the optimum solution for a given combinatorial optimisation problem. The drawback of these methods is, as the size of the instance increases, the total computation time also increases excessively. Nevertheless, instances of small size can be solved efficiently by these methods. Some of them will be reviewed next.

Branch and Bound (B&B) and Dynamic Programming (DP) are two of the classical methods that give exact optimum solutions by partially searching the feasible solution set. In B&B, a branch is a subset of solutions of the partitioned problem, and the bound is the lower bound computed that helps to find the optimum [58,84]. As DP solves subproblems, it keeps the solutions as a future reference, i.e if it finds the same subproblem it uses the result that was stored. In order to find the optimum value it starts from the bottom subproblem and goes to the main problem, which also guarantees that the subproblems are solved [26,31,84]. B&B solves the problem by partitioning it forwards whereas DP solves it backwards.

The cutting-plane method is another rigorous approach for combinatorial optimisation problems. In this method, the feasible set is renewed by adding linear inequalities at each iteration. These inequalities are referred as cuts. This method is inefficient, and has low convergence rate [79].

Although the cutting-plane method is known to be inefficient itself, its combination with B&B has been proven to be very efficient. The result is the Branch

and Cut Algorithm described below.

2.4.2 Branch and Cut

This algorithm combines the B&B with cutting planes. It is based on the idea of B&B but with additional cuts generated at each node of the branching tree, before pruning and further branching. It proceeds by solving the LP-relaxation of the original problem. Where the solution to the LP-relaxation represented here as Z_{LP} is infeasible, we can terminate since the solution to the MILP presented as Z_{MILP} will also be infeasible. Otherwise, this initial solution acts as an upper bound Z_{up} until better solution i.e lower upper bound can be found and the solution is then updated. If the solution to the LP-relaxation is integral, we can stop because this will also be the optimal solution to MILP of the same problem.

Typically, the LP-relaxation is solved using the simplex algorithm. This solution will be evaluated to check for integrability requirement. Where LP-relaxation returns fractional values, we add a cutting plane (inequality) at the node and split into two sub problems. This inequality should satisfy every integral point that is feasible in the current LP-relaxation. The cutting plane " cut off " a part of the feasible region thereby reducing the size of the search space which strengthen the bounds. This helps to quickly provide a lower bound for a problem of minimisation and an upper bound for maximisation which can provide a guarantee on how far we are from optimality. The remaining

problem is then solved using B&B. The process continues until integer variables are obtained as required.

While the Cutting planes are known to be fast but can be unreliable, B&B is known to be reliable but can be slow. The combined algorithm is guaranteed to produce optimal solution in reasonable time. The method is so successful that most commercial software tools implement it in their solution packages, it has been used here to get an exact solution to the YCSP and reshuffle problem. The pseudo-code is shown in Algorithm 2.1

Algorithm 2.1 Branch and Cut, (*Source*, Mitchell 2002)

1: Denote the initial integer programming problem by ILP^0 and set the active nodes to be $L = ILP^0$. Set the upper bound to be $\bar{z} = +\infty$. Set $z^* = -\infty$ for the problem $l \in L$
2: If $L = \emptyset$, the the solution x^* which yielded the incumbent objective value $\bar{z}$, is optimal. If no such x^* exists (*i.e.*, $\bar{z} = +\infty$) then ILP is feasible.
3: Select and delete a problem ILP^l from L
4: Solve the linear programming relaxation of ILP^l. If the relaxation is feasible, set $z^* = +\infty$ and go to Step 6. Let z^* denote the optimal objective value of the relaxation if it is finite and let x^lR be an optimal solution; otherwise set $z^* = -\infty$.
5: If desired, search for cutting planes that are violated by x^lR; if any are found, add them to the relaxation and return to Step 4.
6: If $z^* \geq \bar{z}$ go to Step 2. If $z^* < \bar{z}$ and x^{lR} is integral feasible, update $\bar{z} = z^*$, delete from L all problems with $z^* \geq \bar{z}$, and go to Step 2.
7: Let $\{S^{lj}\}_{j=1}^{j=k}$ be a partition of the set S^l of problem ILP^l. Add problems $\{ILP^{lj}\}_{j=1}^{j=k}$ to L, where ILP^{lj} is ILP^l with feasible region restricted to S^{lj} and z^*lj for $j = 1, ..., k$ is set to the value of z^* for the parent problem l. Go to Step 2
8: **Return** x^*

2.4.3 Approximation Methods

Where the problem is difficult to solve exactly due to the size, approximation methods can be adopted. This can either be achieved by using approximation algorithms or heuristics.

2.4.3.1 Approximation Algorithms

Approximation algorithms for combinatorial optimization problems do not necessarily provide an optimal solution. However, they approximate the optimum solution to a guaranteed error value α [51,94].

Greedy and local search algorithms are two standard approximation algorithms. In the greedy algorithm each step guarantees that the solution provided is locally optimal. The local search algorithm, on the other hand, starts with an initial solution and iteratively improves it by making changes to find a better local optimum solution [106]. The deterministic rounding algorithm is an example of a local search algorithm. It consists of solving the relaxation of an integer programming model of the problem, i.e. as a linear programming model and then getting integer solutions by rounding the obtained results [5].

2.4.3.2 Heuristics

Real life problems are hard to solve. Most of the combinatorial ones are $NP-hard$. Thus, exact algorithms are inefficient and costly, especially when the

problem size is large. Instead of finding the optimum solution, heuristics generally find good approximations to it in acceptable computational times.

They are either problem specific i.e., fashioned or designed to solve specific problem or instances (Heuristics). On the other hand, they can be general purpose oriented that can be adapted to solve different problems or instances (Metaheuristics). Both are loosely called heuristics. In this thesis, we refer to the specific problem oriented heuristics. The BIP in problem Section 5.5 can only be solved for small instances exactly in reasonable time. We have developed four new heuristics in Section 5.6 to solve medium to large size instances of the problem as they arise in real world.

2.5 Summary

In this chapter, we have looked at the Integer Linear Programming Problems, and reviewed some of the most popular solution approaches both exact and approximate. The most prominent exact approaches for the ILP are B&B, the Cutting Plane algorithm and a hybrid based on both namely, Branch & Cut. On the approximate side, there are the ever so popular and effective heuristic algorithms. The next Chapter reviews the most prominent problems encountered in a modern Container terminal.

Chapter 3

Literature Review

3.1 Introduction

There is an extensive literature on container terminals ranging from berth allocation problems, QC scheduling, IT scheduling to storage space allocation and YC scheduling. The storage space allocation and YC scheduling are central problems of yard management. The first problem that arises at a container port is that of allocating space for an arriving vessel to dock, otherwise called berthing. These vessels arrive and leave the port at different times [80], therefore, there is need for proper allocation of the berthing space to manage them. In [37], the area for the vessel to berth is determined using berth template problem. Some other research into this area includes, [13] and [97]. This is referred to as the Berth Allocation Problem (BAP).

After berthing, the containers are unloaded or loaded by special equipment called QCs. Often more than one QC will be assigned to handle different areas of a vessel depending on its length. The assignment could be based on

length, width or stability of the vessel [98]. Other considerations could be determining specific QCs to assign to each vessels when different vessels berth simultaneously [96]. Thus, there is a need to assign these QCs through the solution of the Quay Crane Assignment Problem (QCAP). Once the QCs have been assigned, there is still the need for scheduling their operations because there exist priority conditions imposed by the stowage plan of each vessel. Studies into the Quay Crane Scheduling Problem (QCSP) includes [7], [67] amongst others.

The containers from the vessels are loaded onto internal trucks which will take them to the yard area. These trucks must be properly managed in order to avoid bottlenecks in their operations. The problem involves determining the truck to pick which container and their routes; this is the Internal Truck Scheduling Problem (ITSP) (See [93] and [16] for extensive reviews). Our focus however is on yard management problems. There are two basic problems at the yard which are YC scheduling and container reshuffle. These two issues have to be properly managed for a smooth running of operations at the yard. However, there are other problems too. They include the yard layout and the choice of equipment among others.

3.2 Yard Crane Scheduling

The YCSP can be categories into two: The schedule of single YC to retrieve and store containers within a block and schedule of multiple YCs to several

blocks. The focus in this thesis is on the second category. Most research on the YC scheduling problem (YCSP) proceeds by representing the problem as an integer program and solving it using different approaches. Due to size of the problem, the integer models are solved using exact methods for small problem instances and heuristics/metaheuristics are employed to solve realistic size problems.

3.2.1 Heuristics

The least cost heuristic was developed in [72] to solve a Mixed Integer Programming (MIP) model for dynamic crane deployment in a container yard on a shift-to-shift basis based on a rolling horizon. They determined the optimum deployment frequency/period using a simulation technique. A continuous time MIP model using a heuristic and a rolling horizon algorithm was considered in [70]. They concluded that, in comparison with other existing heuristics, their algorithm performed better by yielding higher solution quality at a faster rate. In order to increase the efficiency of RTG and reducing the cost of energy by real-time tracking data of RTG, a heuristic was proposed in [112] to solve instances based on real time data.

RTG dispatch was modelled as an MIP and the heuristic was used to assign jobs to the RTGs decided by the time window. A dynamic programming based heuristic and an algorithm was adopted in [82] to find lower bounds for the YCSP considering inter-crane interference after proposing an integer

program for the problem. The problem of multiple YCs handling jobs with different ready times within a yard zone was studied. An approximate dynamic programming based heuristic was applied in [77] to solve real time dispatching model of RTGs to increase the throughput of import and export containers and to reduce a container ship's berthing time. They claimed based on real runtime analysis that the method is good in conjunction with grounding policy.

A modified Lagrangean relaxation method was used in [113] to solve the MIP problem arising from YC scheduling. In order to reduce the large duality gap, they augmented the original model with additional constraints for the Langrangian relaxation problem. While a successive piecewise linear approximation was introduced in [23] to the problem and approximated by a linear network flow problem. They found their method more efficient than the Lagrangean decomposition approach for the same purpose. A Recursive Backtracking algorithm was deployed in [39]. It combines advantages of A^* search (which finds paths from a start node to leaf nodes in a search tree) along with an admissible heuristic and backtracking with a prioritized search order. Using real time data-driven simulation, they predicted the time taken by YCs to perform their operations and to get the optimal dispatch sequence.

B&B was applied in [83] to solve an MIP formulation of scheduling YC to perform different tasks in a block. The study was based on loading and unloading operations of a YC at different ready times. Different models were examined on YCSP in [100] and after making some modifications a time decomposition method was adopted to solve the reduced complex problem based on Benders

decomposition. The problem was modelled as an MIP and the rolling horizon method applied. It was observed that fixing zones for the yard cranes seems to result in better models than time discretization and that time decomposition shows good results therefore advisable when the number of cranes is limited.

3.2.2 Metaheuristics

Metaheuristics were used by some authors to get near optimal solutions. This includes [117] which investigated the problem of multiple YC scheduling as a MIP that minimizes the sum of truck waiting cost and YC moving cost. They claimed that using a Genetic Algorithm (GA) improved operational efficiency and reduced the production cost. A hybrid algorithm combining GA and Tabu Search (TS) was applied in [74] to study the problem of scheduling YC to load and unload containers with different ready times in a yard zone. Two new operators, TS crossover and TS mutation were introduced and found to be effective and efficient. A novel GA was adopted in [111] for multiple YC scheduling after formulating the problem as an MIP model. The aim was to determine the sequence of YCs operations to complete a given task in the shortest time while accounting for the waiting time caused by precedence relations between those tasks. They compared their GA to SA and concluded that their method is better.

GA and TS were employed in [20] to study the problem of scheduling multiple YCs in loading operations of container terminals. They claimed that both

algorithms performed well on instances of small size. TS was found to perform better than GA for larger instances in terms of quality and efficiency of the solution. The impact of uncertainties of vessel and truck arrivals and container handling time on YCSP was addressed in [114]. The MIP model of the problem was formulated to balance the initial cost of the baseline schedule and the expected cost. Subsequently, a simulation based GA was applied.

3.2.3 Combination of Heuristics and Metaheuristics

A combination of heuristics, metaheuristics and exact methods was adopted by some authors. A hybrid algorithm which integrated a heuristic and a parallel GA (PGA) was established in [42] to schedule YCs based on a rolling-horizon basis. Thereafter, a simulation model incorporating the YC scheduling model and algorithm was introduced to evaluate their approach. A novel dynamic rolling-horizon strategy was developed in [19]. Due to the complexity of the problem, a heuristic algorithm together with a simulation model was adopted. The GA was consequently used to generate the initial solutions. GA was deployed in [49] to solve the YC scheduling with non-interference constraints after formulating the problem as a MIP model. For this problem, the authors demonstrated that GA is more efficient as a heuristic method compared to the exact method of B&B as implemented in LINGO 8.0.

The Particle Swarm Optimization with GA Operators (PSOGAO) was combined in [59] to solve YCSP with different ready times. The objective was to

minimise the sum of job waiting times of YCs to complete a job having different ready times. The trade off between YC Scheduling and energy saving was examined in [43]. They formulated the problem as MIP with the objective of minimising total completion delay of all task groups and energy consumption. Using simulation which integrates GA and PSO, they claimed their method was capable of solving YCSP with different sizes in real time. An agent-based approach was developed in [92] to assign and relocate YCs between blocks based on forecast workload. Preference functions were given to the YCs and blocks. They claimed that their model can reduce incomplete workload for real world problems.

3.2.4 Integrated problems

There is research on integrating the different problems that arise at container terminals. This includes [44] which studied the coordinated schedule of the three main equipment (QC, IT and YC) used in a container terminal while considering energy use. The problem was formulated as MIP with the objective of minimising the total departure delay of all vessels and the total transportation energy consumption of all task. An integrated simulation-based optimisation method was developed. This incorporates simulation combined with GA and PSO. The yard truck and YCSP was formulated in [14] for loading operations in a container terminal as a MIP with the objective of minimising the makespan of loading all outbound containers in a planning horizon. Due to the problem of intractability, they adopted Benders cut method and combinatorial Benders

cut-based method for practical size problems.

The reshuffle in YCSP was solved in [115] by formulating the problem of choosing appropriate locations for reshuffled containers and obtain the pick up sequence for all the containers by the YCs. The objective was to reduce the sum of the total completion time for all the containers and the reshuffle time for containers. Using PSO for practical size problem, they claimed that their result is better than solving the two problems separately. Note that combining problems leads to very large models and instances which can be very challenging to solve. See [1] for the integration of BAP, QCAP and QCSP for instance.

3.2.5 Shop scheduling

YCSP is a special case of machine scheduling where machines in this case YCs are assigned to process different tasks. On the other hand, when tasks are asigned to different machines, we have Shop scheduling. The problem is that of a number of tasks having to be completed by different machines. Each task have a processing time on the machines. The aim is to find a schedule that minimises the completion time of the tasks. A machine can only process one task at a time and a task can only be process by a machine at a time. Some notable cases are Flow shop where each task is processed on each machine in precedence condition i.e.., each task is processed on machine 1, machine 2 , machine 3 and so on. Recent works are [107], [86] and [48]. Another case

is the Job shop where though the tasks have to go through a fixed route but not compulsory for all the tasks. Works on this area includes, [6] and [68]. Furthermore, we also have the Open shop scheduling. Where the route of the tasks are not important, we have a case of open shop scheduling problem. Research in this area includes, [103], [3] and [4]

3.3 Container Reshuffle

Stacking policies play a major role in the proper management of the yard as these can reduce the need for reshuffles otherwise called rehandling. Workload should be spread properly through the yard between blocks. Reshuffle is where you need to move a container in order to access another one. This creates a waste of productive time and hence should be avoided as much as possible. Proper assignment of containers in the yard area is one of the issues that needs to be addressed in a container terminal. This involves the allocation of containers to appropriate locations right from the time they are stored. There are many ways reshuffles can be minimised at the port; this includes having a appropriate storage strategy for the in-coming containers, re-arranging/pre-marshalling containers during idle times of YCs and relocation when the containers are being retrieved.

3.3.1 Storage Strategy

Literature on selecting a storage strategy includes [99] which investigated the role announcing truck arrival could have on the stacking policy of a container terminal. They concluded that it will be more beneficial to improve available information at the time of stacking than attempting to fix poor stacking decisions. This was done using a discrete-event simulation model in evaluating expected departure time for an import container to schedule the pre-emptive remarshalling moves. It was a follow-up on the work of [27] that examined various stacking strategies for an automated container terminal and [9] where a study of departure and stacking further away or close to exits points were considered. It was discovered that the trade-off between where to stack and accepting more reshuffles leads to improvements over the benchmark.

In [116], a simulation technique was deployed to investigate the truck arrival information and container rehandles in import container retrieval process. They found out that a complete arrival order is not required to significantly reduce rehandles. However, benefit can be obtained from information about truck arrival. There are benefits in stacking higher and using a larger number of rows. A storage system that simulates and optimises the movement and storage of containers within the terminal was developed in [18]. A mathematical model was presented that minimises rehandling moves, while having total job times as the objective function. Using four heuristics and three meta-heuristics, they concluded that the model and optimisation should be used within a larger setting. The larger program will focus on optimising the movements of

machines and ensure they arrive at their final destination on time.

The problem of assigning containers to storage spaces that minimises the total expected number of relocations was evaluated in [109]. The paper addressed both dynamic and static location problems. The static model was solved using GA and the dynamic by minimum space waste rule, which was found to outperform GA. The problem of container reshuffle by developing five new heuristics and improving an existing model was addressed in [95]. A simulation model was developed to animate the stacking, retrieving, and reshuffling operations and also to test the performance of the heuristics for both static and dynamic states. This was a follow up on the work in [102] which examined the problem of assigning locations to incoming containers and the need for reshuffle. An IP model of the problem was formulated for the first time with the objective of minimising the number of reshuffles in assigning storage locations for incoming container and reshuffled export containers. A variant of the IP-based heuristic was applied to solve the problem.

The yard operations in some container terminals was investigated in [22] and came to the conclusion that higher container stacking has an impact on the number of unproductive moves. They claimed that the major impact was on the delivery operation. The use of heuristics that relies on ϵ- optimal policies to compute specific allocations for empty containers between different ports was proposed in [69]. The problem was formulated as a multi-port containerisation model. The Harmony Search was adopted in [2] to solve the storage allocation problem for inbound containers. The objective was to find an optimal container

arrangement considering the departure dates and also minimise rehandles of these containers. They claim that their approach is competitive with the LIFO and GA.

The application of decision trees from a set of optimal solutions to locate export containers in a container terminal yard considering weight was examined in [57]. A dynamic programming model of the problem was formulated to determine the storage location to minimise the number of relocations expected for the loading operation. SA was used in [52] to derive the best stacking strategy for containers in a yard with uncertain weight information. Simulated experiment showed that the strategy was able to reduce the number of re-handles compare to traditional same-weight group stacking strategy. They advised that more improvement can be obtained where the accuracy of the weight classification is done by machine learning. A new domain-dependent planning heuristics was developed in [90] to find an appropriate configuration for containers in a bay. The problem was modelled as an Artificial Intelligence planning problem and likened to the Blocks World domain with some differences. The authors claimed that their heuristic was able to allocate containers to bays with minimal need for reshuffle. A comparison was made between three different stacking alternatives using RMG in [89]. Simulation method was adopted to study their efficiency in terms of productivity, flexibility, area utilisation and cost. They discovered cross-over RMG to be the best performing especially when the workload is balance. The twin RMG was observed to be most suited for situations with a balance workload at sea and landside.

While the single RMG was observed to be most suited for situations with a low landside peak.

3.3.2 Pre-marshalling

The integer programming model, a neighbourhood search process and three minor subroutines were deployed in [64] to minimise the number of container pre-marshallings and reduce re-handles. They assumed there are no containers entering or leaving the bay during the pre-marshalling process. The pre-marshalling problem was modelled as an integer programming problem based on multi-commodity network flow in [65]. The optimisation objective was to minimize the number of container movements during pre-marshalling. A heuristic called the tree search procedure was developed in [10] for solving the container pre-marshalling problem. It is based on natural classification of possible moves, making use of lower bounds and applying a branching scheme of move sequences rather than single moves. It proved effective on large real-world instances, according to the authors.

The problem of rearranging containers before they are shipped was examined in [54] by formulating it as a dynamic programming model of the bay matching and task sequencing problem and solving it. The move planning was solved by using a transportation model. However, a lot of computational time was used hence, heuristics were advised. A two-step based SA was proposed in [24] to investigate intra-block remarshalling plan that is free from rehandles during

both the loading and remarshalling, considering twin Automatic Straddle Carriers (ASCs). The first step identifies the slot that minimises reshuffles and the second step schedules ASC that minimises the interference between the ASCs. Integer programming was applied in [63] to solve the terminal allocation problem for vessels and yard allocation problem for transhipment container movements for a port that has multiple terminals. A two level heuristic approach was adopted to solve the integrated problem.

3.3.3 Relocation

Work on the relocation problem includes [50] which proposed an improved greedy look ahead heuristic in solving the container relocation problem (CRP). The objective of CRP is to find an optimal operation plan for the crane with fewest number of container relocations. The method was found to be efficient especially for large scale instances. Three heuristic methods; index based, binary IP and beam search were developed in [41] to solve a binary integer programming model of CRP. They concluded that the beam search heuristics out- performed other heuristics. Emptying a stack without new arrival was modelled as integer programming problem to derive an optimum reshuffle sequence. The problem was broken into parts and solved by four heuristics, IP-based, Lowest slot, Reshuffle Index (RI) and the Expected Number of Additional Relocation (ENAR). They claimed that their heuristic MRIP-Dk (MRIP that minimises the number of reshuffles in retrieving k containers plus estimated future reshuffles in stack after k containers are retrieved) outperformed

other heuristics found in literature.

A tree search procedure was employed in [32] to solve the container relocation problem. The heuristic is based on natural classification of possible moves and used branching scheme that move sequences of promising single moves. They claimed the method was compared to other CRP and it was found to be competitive to others especially for large real world instances. The problem of minimising the expected number of reshuffle was modelled as a stochastic DP problem in [35]. Due to the overwhelming number of states of the model, a decision tree heuristic was developed to solve realistic size problems. They claimed that their heuristic outperformed other stacking policies commonly used in practice.

A meta-heuristic called the Corridor method was developed in [17] to solve the CRP in stacking containers in a container terminal yard, pallets and boxes in a warehouse. The objective is to find the block location that minimises the number of movements that is required in the desired retrieval sequence. The imposition of exogenous constraints reduced the size of the problem and made use of constrained DP a practical approach even on large instances. In [66] a three-phase heuristic was proposed to optimise the work plan for a crane to retrieve containers from a yard according to a given order. They aimed at minimising the weighted sum of the number of container movements and the total cranes working time. The first phase generates an initial feasible movement sequence. The other two phases are iterative and terminate when a number of consecutive iterations cannot improve the current solution. This

study is similar to the work in [10] on the tree search procedure for the pre-marshalling of containers in a container terminal.

Scheduling of container movement by using an autonomous learning method was addressed in [46]. This is based on a new learning model considering container groups and the Q-learning algorithm. The desired position of containers in a group is considered by the algorithm based on the Markov Decision Process (MDP). Using simulations, the proposed method was able to find solutions that had a smaller number of rehandles compared to conventional methods. This was a follow-up to previous works in [45] where they discovered that the number of container arrangements increases exponentially with increase in the total count of containers.

The desirable movements of containers that reduce the total turnaround time of ships was determined in [47], using the Q-learning algorithm. They concluded that the number of container movements generated by their method is smaller than that generated by human operators in real-scale problems. In [30] heuristics was deployed to minimise the number of movements required to locate all containers. Features that consider the occupancy rate of the bays and percentage with high priority were considered. An instance generator was also suggested for instances with varying degrees of difficulty. They concluded that both the heuristic and instance generator out-performed other methods found in the literatures.

3.4 Yard Layout

One of the significant differences between ports is the layout of the yard area. It affects the storage as well as the retrieval of containers. The layout of a container terminal can be categories into two; overall layout design and block design. The layout of the Port of Felixstowe is shown in Figure 3.1.

Figure 3.1: PoF Layout, (*Source*, Port of Felixstowe)

3.4.1 Overall yard layout

There are two types of layout that container terminals can adopt: the Asian and the European. The Asian layout uses a parallel approach in locating blocks in the yard compared to the quay area and are characterised by non-automated

equipments. The European layout on the other hand use the perpendicular approach and used mainly automated equipments [15].

An optimisation model was proposed in [62] to determine the best layout of a container yard while considering such factors as, storage space requirements, throughput capacities of YCs and the transporters. The objective was to minimise the total cost consisting of the construction cost for the ground space, the fixed overhead and the operating cost of YCs and the operating cost of transporters. They considered the two types of yard layouts: parallel and perpendicular.

A critical-shaking neighbourhood search method was applied in [71] to solve the yard allocation problem. It improves the quality of the priority sequence iteratively, from an initial random sequence. The method picks some critical request, shakes the priorities randomly and then implements a local search. The multi-criteria decision making (MCDM) was developed in [73] to investigate the impact of layout and automation on terminal performance. As expected, there was a positive correlation between the use of automation and the efficiency of the terminal. The relationship between the layout and the choice of equipment in different areas of a container terminal was examined in [11]. The conclusion is that their dependence is affected by the number of containers handled, available area to operate and the mode of hinterland transportation.

MIP was applied in [104] to find the layout that minimises the total distance travelled between the quay area, the yard area and the yard gate. Using simulation, they showed that MIP performs better that man-made layouts in

terms of the quay crane moves per hour. A method of designing the optimal layout of the container yards was proposed in [56]. Their method suggested the best layout, the outline of the yard and the number of vertical and horizontal aisles. They argued that the Asian layout outperforms the European one. However, the study assumed that the shape of the yard is rectangular. A study of the layout design of a yard and the cycle-time of YCs in a container terminal was examined in [60]. They studied the relationship between different YCs operations, in terms of receiving, delivery, discharging and loading, the cycle times and the movements. It was concluded that the Asian layout has a lower expected cycle-time than the European one.

3.4.2 Block design layout

The relationship between the size of a block in the layout design of a terminal and its efficiency was examined in [81]. It was suggested that the only practical way to deal with optimising decision making at a container terminal port is to treat it as a multi-objective problem optimising the various factors influencing the QCR. These factors are cross gantry frequencies of YCs, congestions on roads by ITs, YC overloading due to QCs demand and YC clashing frequencies. The study of optimal block size and layout of a container terminal was investigated in [61], using four optimisation models to minimise the weighted YC cycle time and truck waiting times subject to minimum storage capacity, and maximising the storage capacity subject to maximum YC cycle time and truck waiting time. They concluded that the optimal number of block-bays

(block-rows) for the Asian layout was larger (smaller) than for the European.

A discrete event simulation model was designed to investigate the effect of block width and layout on terminal performance in [85]. They discovered that the overall performance improves as the terminal becomes more square shaped. This was corroborated in [53] where by using simulation it was concluded that the optimal block width decreases with more equipment deployment. They concluded that the advantage of more deployment diminishes as block width increases. They also indicated that the QCR is concave with respect to the block width.

A model was developed in [25] to determine the number of bays for different yard sizes based on different YCs such as straddle carriers, RTGs and RMGs. They concluded that for the port under study, straddle carriers perform better in a one berth terminal but was outperformed by gantry cranes for larger terminals. The travel time of equipment used in storage and retrieval of containers in a container yard was investigated in [101]. By deploying a simulation model to investigate the relationship between choice of equipment and the yard layout, they discovered that the ASC performs better than straddle carriers where the number of rows for each crane is smaller than nine containers.

Integer Linear Programming was deployed in [28] to compare the performance between routes of one and two ASCs in a block working with SCs. They concluded that one ASC working alone for over four hours required about 70% more time than two ASCs working together to accomplish the same moves. The impact of yard and block layout on specific performance measures in a

container terminal was evaluated in [21]. They focused on minimizing the yard space subject to the space requirement and transforming the problem into a directed acyclic graph. Many meta-heuristics were used but they discovered GA outperformed other heuristics.

3.5 Equipment Types

The yard management is also affected by the type of equipment and their location in the yard. Typically, not more than two cranes are used in each block to avoid collision, maintaining equal distance between them. Furthermore, each crane serves only one block at a time [88]. The Asian layout is noted for using cranes and is not mostly automated, while the European layout uses mostly automated equipment such as automated guided vehicles and straddles. Invariably, the European layout is more expensive compared to the asian layout due to the automation. Hence, there is a relationship between the choice of equipment and layout of the yard in a port container terminal. Regression analysis was used in [53] on the simulated result to investigate the performance of cranes and layout of the blocks. They concluded that there is a strong relationship between yard layout and the performance of the cranes.

Figure 3.2: RTG in operation, (*Source*, Port of Felixstowe)

Figure 3.3: RMG in operation, (*Source*, Port of Felixstowe)

3.6 Summary

In this Chapter, we have briefly reviewed the extensive literature on yard management and operations. While the yards operations problems are similar in different container terminals, solution approaches are varied. There is also a plethora of models to capture the different situations used and to keep up with the ever increasing flow of goods. A common aspect in the solution approaches is their approximate nature. This is understandable given the intractability nature of most of the models reviewed. The next Chapter is on the specific problem of yard crane scheduling.

Chapter 4

Yard Crane Scheduling Problem

4.1 Introduction

One of the major challenges in the storage area of a container terminal is determining **which** container will be needed, **when** and **where**. This invariably determines the workload in blocks and the need for RTGs. This challenge has two aspects to it:

1. Determining the number of RTGs to assign to blocks in order to account for the difference in workload distribution.

2. Determining the routes of RTGs from one block to another for the given period.

RTGs are often planned by port supervisors in an *ad hoc* manner based on experience. However, the size of the problem and its combinatorial nature makes it difficult for an *ad hoc* approach to guarantee good solutions. In view of this, the problem will be solved in a rolling horizon plan. A short planning

period in the immediate future will be fixed and updated regularly as new information is made available. This makes the computation less burdensome and more flexible in view of the unpredictability of future work. A long planning period may include much uncertain information and as such is less reliable.

Each RTG serves one area of the yard at any point in time. They are large equipments used at the yard to store and retrieve containers from their location. There are two types;

1. The RTG crane moves on tyres and hence has greater flexibility and access to the containers. It can move forward, backwards and can also turn by making one or two 90^0 moves to access containers.

2. The RMG or Rail Mounted Gantry crane which moves along a rail and is thus limited in movement to where the rail is laid. This can only move forward or backward.

A YC, here an RTG, is a big apparatus; its size hampers movement around the yard. This movement between blocks takes time and results in loss in productivity. The effective management of YC operations is therefore crucial to the overall operation of the port. It has to move after completing the work in a block to another block where there is more work within a deployment period provided that it can still get there before the end of that period. It can move easily from one block to another horizontally but it is difficult to move vertically because it has to make two 90^o turns before reaching the desired

block. This kind of movement is called cross-gantry and should be avoided as much as possible.

Figure 4.1 shows a typical horizontal move between blocks B1 to B2 or B3 to B4, while a move between blocks B1 and B3 or B1 and B4 is vertical, requiring a 90^o turn twice. Our model focuses on intra-zone, i.e. inter-block, moves within a planning period.

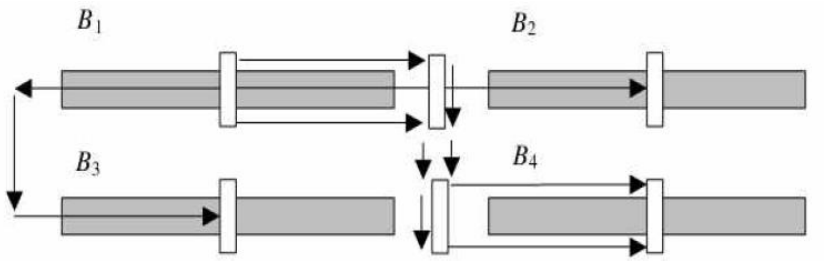


Figure 4.1: RTG movement between blocks, (*Source*, Zhang 2002)

RTGs perform three major operations: storage, retrieval and reshuffle. These operations can be measured either in terms of the number of moves or the time spent on the task. Linearity is assumed in the use of RTGs in these operations. If each container move takes 3 minutes, then a container with two moves will require 6 minutes. Depending on the location, a container may require more moves or time than another. We have adopted time as a measure of the workload in this thesis.

The problem is to find the path of every RTG from their initial position to their final position that minimises the total unfinished work in the yard at the end of a planning period. The rolling horizon approach is adopted in this thesis. This involves dividing the planning horizon into short immediate future plans and

updating them as new information is made available for the next plan. This process continues until end of the planning period.

In this Chapter, we introduce two models based on formulation in [72]. Model1 introduces a new set of constraints that specifies the number of RTGs available per deployment. While Model2 has an objective function that is a linear combination of unfinished work and surplus capacity.

4.2 Model Assumptions

- The number of RTGs will not change as we move from one period to another i.e we assume no fault to the cranes, no crane driver falling sick or loss of cranes to the cause of operations
- All RTGs have the same capacity which is measured in minutes. It is assumed that all the cranes can work throughout the planning period i.e 60 minutes per hour
- Each RTG can work in any block. No crane is designated to work in a particular block
- The planning period is constant. This is divided into equal time periods of say 10 or 15 minutes depending on the frequency of deployment.
- When work is not completed in a particular period, such work is carried over to the next period as part of the workload for the period
- Workloads are known in advance at the start of the planning period.

Work is distributed between the blocks before the start of the planning period

- RTGs can move from a block where there is no work to another where there is, but only one movement per period is allowed. This is in order to avoid traffic congestion and loss of crane capacity

- RTGs can work in a block in more than one period. Unless deployed to another block, RTGs once assigned are expected to continue working in that block

4.3 The Linn and Zhang Model

The objective is to minimise the total time it take to deal with the workload subject to a number of constraints as in [72]. The parameters and decision variables used are as follows.

Parameters:

C	The capacity of each RTG in a deployment period
B	The total number of blocks
T	The number of periods in a planning period
B_{it}	The workload in block i at time t
τ_{ij}	The RTG travel time from block i to block j; it is given

Decision Variables:

X_{ijt}	The number of RTGs moving from block i to block j at the start of time t
W_{it}	The workload left in block i at the end of time t
W_{it}^{+}	The surplus RTG capacity in block i at the end of time t

The Model:

$$\min \sum_{t=1}^{T} \sum_{i=1}^{B} W_{it} \tag{4.5}$$

subject to :

$$\sum_{j=1}^{B} X_{ijt} = \sum_{j=1}^{B} X_{ji(t-1)} \quad \text{for} \quad i = 1, 2...B, \quad t = 1, 2...T \tag{4.6}$$

$$\sum_{j=1}^{B} X_{jit} \leq 2 \quad \text{for} \quad i = 1, 2...B, \quad t = 1, 2...T \tag{4.7}$$

$$W_{i(t-1)} + B_{it} - W_{it} - \sum_{j=1}^{B} (C - \tau_{ij}) X_{jit} + W_{it}^{+} = 0$$
$$\text{for} \quad i = 1, 2...B, \quad t = 1, 2...T \tag{4.8}$$

$$(C - \tau_{ij}) X_{jit} \geq 0 \quad \text{for} \quad i, j = 1, 2...B, \quad t = 1, 2...T \tag{4.9}$$

$$X_{ii0} = 1, \quad \text{for} \quad i = 1, 2...B \tag{4.10}$$

$$X_{ij0} = 0 \quad \text{for} \quad i = 1, 2...B, \quad j = 1, 2...B \quad \text{and} \quad i \neq j \tag{4.11}$$

$$W_{it} = 0 \quad \text{for} \quad i = 1, 2...B, \quad t = 0 \tag{4.12}$$

$$W_{it}^{+} \geq 0, \quad W_{it} \geq 0 \quad \text{for} \quad i = 1, 2...B, \quad t = 1, 2...T \tag{4.13}$$

$$X_{ijt} \geq 0, \text{integer for} \quad i = 1, 2...B, \quad j = 1, 2...B, \quad t = 1, 2...T \tag{4.14}$$

Equation (4.5) is the objective function which is the total unfinished work over the whole deployments. RTG conservation flow, equation (4.6), ensures that the number of RTGs used per deployment be the same as it moves from period to period. At most two RTGs can work in a block in a given deployment period is represented by equation (4.7). Equation (4.8) maintains the balance between demand in terms of workload and the supply in terms of the net-crane

capacity. The fact that RTGs cannot move from one block to another where they cannot reach is depicted by equation (4.9). Equation (4.10) states that one RTG is assigned per block at start of the deployment. No RTG can move from one block to another at start of the deployment is represented by equation (4.11). Equation (4.12) states that there is no unfinished workload at start of the planning period as it is already included as work for the first deployment period. Equations (4.13) and (4.14) are conditions on the decision variables.

4.3.1 Travelling Time and Net Crane Capacity

While the planning period varies, the travel time between blocks are constant. It is important to note that $\tau_{ij} = \tau_{ji}$ and $\tau_{ij} + \tau_{jk} > \tau_{ik}$ for all $i, j, k = 1, 2, ..., B$. This means the travel times are symmetric and also satisfy the triangle inequality theorem. The net crane capacity is the difference between deployment period and travel time for each RTG.

4.3.2 An instance of the Linn and Zhang Model

To check for the intractability of the model, we solve a small instance of the problem using MATLAB MILP solver, `intlinprog`. The instance of the model is shown below with two blocks; two RTGs and a deployment period of 15 minutes. The workloads are 18.75 and 3.75 mins in blocks 1 and 2 for the first period. Workloads are 3.50 and 14.50 mins for blocks 1 and 2 respectively in

the second period.

$$\min \quad W_{11} + W_{21} + W_{12} + W_{22} \tag{4.15}$$

subject to :

$$X_{110} + X_{210} = X_{111} + X_{121} \tag{4.16}$$

$$X_{211} + X_{221} = X_{120} + X_{220} \tag{4.17}$$

$$X_{112} + X_{122} = X_{111} + X_{211} \tag{4.18}$$

$$X_{212} + X_{222} = X_{121} + X_{221} \tag{4.19}$$

$$X_{111} + X_{211} \leq 2 \tag{4.20}$$

$$X_{121} + X_{221} \leq 2 \tag{4.21}$$

$$X_{112} + X_{212} \leq 2 \tag{4.22}$$

$$X_{122} + X_{222} \leq 2 \tag{4.23}$$

$$15X_{111} + 10X_{211} + W_{11} - W_{11}^{+} = 18.75 \tag{4.24}$$

$$10X_{121} + 15X_{221} + W_{21} - W_{21}^{+} = 3.75 \tag{4.25}$$

$$15X_{112} + 10X_{212} + W_{12} - W_{12}^{+} - W_{11} = 3.50 \tag{4.26}$$

$$10X_{122} + 15X_{222} + W_{22} - W_{22}^{+} - W_{21} = 14.50 \tag{4.27}$$

$$W_{it}^{+} \geq 0, \quad W_{it} \geq 0 \quad \text{for} \quad i = 1, 2 \quad \textit{and} \quad t = 1, 2 \tag{4.28}$$

$$X_{ijt} \geq 0, \quad \text{integer for} \quad i, j, t = 1, 2 \tag{4.29}$$

The travel time between blocks are as shown in Table 4.1. This represent the time taken by RTGs to move from one block to another within the deployment period.

Table 4.1: Travel time between blocks, (*Adapted from*, Linn & Zhang 2003)

Blocks	1	2	3	4	5	6	7	8	9	10
1	0	5	10	10	15	15	20	20	25	25
2	5	0	5	10	10	15	15	20	20	25
3	10	5	0	5	10	10	15	15	20	20
4	10	10	5	0	5	10	10	15	15	20
5	15	10	10	5	0	5	10	10	15	15
6	15	15	10	10	5	0	5	10	10	15
7	20	15	15	10	10	5	0	5	10	10
8	20	20	15	15	10	10	5	0	5	10
9	25	20	20	15	15	10	10	5	0	5
10	25	25	20	20	15	15	10	10	5	0

The instance above has 8 integer variables, 8 continuous variables and 12 constraints. It is important to note that the computational time of the MIP increases as the number of integer variables increases. This however depends on the number of blocks and RTGs. Furthermore, the higher the workload relative to the net-crane capacity, the higher the run time. See solution model in Appendix A.

4.3.3 Interpretation of Results

The total unfinished work at end of the planning period is 3.75 mins, this is attributed to work unfinished in block 1 at period 1. There was no unfinished work in any other block in the planning period. There was however surplus capacity of 11.25 mins in block 2 at period 1. In period 1, RTG in block 1 should remain in block 1 and RTG in block 2 should also remain in block 2. While in period 2, RTG in block 1 should remain in block 1 and RTG in block 2 should remain in block 2. There are surplus capacity of 7.75 mins and 0.5 mins in

blocks 1 and 2 respectively in period 2.

Even for this small problem, we can see how prohibitive the calculation can get as the problem size increases. In view of this, we need to modify the model in order to manage the problem.

4.4 YCSP model1

The L&Z model cannot produce the optimal solution in a timely manner hence, a least cost heuristic was therefore developed [72]. Future research will further investigate approaches to last-minute jobs handling and look into developing robust RTG scheduling models with uncertain truck arrival times [70]. The shortcomings highlighted by these authors call for and justify a new and more adequate model. The alternative model we suggest here alleviates these shortcomings.

The notation in this thesis is similar to that used in [72] for easy comparison. The parameters and decision variables used are as follows:

Parameters:

C	The time available to each RTG including travel time between blocks
B	The total number of blocks
T	The number of deployment periods in a planning period
B_{it}	The workload in block i at time t
τ_{ij}	The RTGs travel time from block i to block j; it is given

Decision Variables:

X_{ijt}	The number of RTGs moving from block i to block j at the start of time t
W_{it}	The workload left in block i at the end of time t
W_{it}^{+}	The surplus RTG capacity in block i at the end of time t

YCSP Model1:

$$\min \sum_{t=1}^{T} \sum_{i=1}^{B} W_{it} \tag{4.30}$$

subject to :

$$\sum_{j=1}^{B} X_{ijt} = \sum_{j=1}^{B} X_{ji(t-1)} \quad \text{for} \quad i = 1, 2, .., B, \quad t = 1, 2, ..., T \tag{4.31}$$

$$\sum_{j=1}^{B} X_{jit} \leq 2 \quad \text{for} \quad i = 1, 2, ..., B, \quad t = 1, 2, ..., T \tag{4.32}$$

$$W_{i(t-1)} + B_{it} - W_{it} - \sum_{j=1}^{B} (C - \tau_{ij}) X_{jit} + W_{it}^{+} = 0$$
$$\text{for} \quad i = 1, 2, ..., B, \quad t = 1, 2, ..., T \tag{4.33}$$

$$(C - \tau_{ij}) X_{jit} \geq 0 \quad \text{for} \quad i, j = 1, 2, ..., B, \quad t = 1, 2, ..., T \tag{4.34}$$

$$\sum_{i=1}^{B} \sum_{j=1}^{B} X_{jit} = \sum_{i=1}^{B} X_{ii0} \quad \text{for} \quad t = 1, 2, ..., T \tag{4.35}$$

$$X_{ii0} = 1, \quad \text{for} \quad i = 1, 2, ..., B \tag{4.36}$$

$$X_{ij0} = 0 \quad \text{for} \quad i, j = 1, 2, ..., B \quad \text{and} \quad i \neq j \tag{4.37}$$

$$W_{i0} = 0 \quad \text{for} \quad i = 1, 2, ..., B \tag{4.38}$$

$$W_{it}^{+} \geq 0, \quad W_{it} \geq 0 \quad \text{for} \quad i = 1, 2, ..., B, \quad t = 1, 2, ..., T \tag{4.39}$$

$$X_{ijt} \geq 0, \text{integer for} \quad i, j = 1, 2, ..., B, \quad t = 1, 2, ..., T \tag{4.40}$$

Equation (4.30) is the objective function which is the total unfinished work over the whole deployments. The RTG conservation flow is represented by equation (4.31), which ensures that the number of RTGs used per deployment is the same from period to period. The maximum number of RTGs that can work in a block in a given deployment period is presented by equation (4.32). Equation (4.33) maintains the balance between demand in terms of workload and the supply in terms of the net-crane capacity. Rearranging the terms we have $W_{i(t-1)} + B_{it} - W_{it} = \sum_{j=1}^{B}(C - \tau_{ij})X_{jit} - W_{it}^{+}$. The left hand side represents the actual work done, while the right hand side represents the actual time spent on the job. The fact that RTGs cannot move from one block to another where they cannot reach is depicted by equation (4.34). The total RTG used in all the blocks per period should remain same and is indicated by equation (4.35). Equation (4.36) represents the number of RTGs assigned per block at the start of the deployment. No RTG can move from one block to another at the start of the deployment policy is enforced by equation (4.37). Equation (4.38) indicates that there is no unfinished workload at the start of the planning period. Equations (4.39) and (4.40) are condition on the decision variables.

4.4.1 Model Modification

The number of constraints increases by T because of constraint (4.35) which will influence the solution space by increasing the enumerative searches of the feasible region in an effort to find the optimal integer programming solutions. This process is speeded up when we impose as many reasonable constraints as

possible for defining the feasible and optimal region. Reasonable means that these constraints are not redundant, each uniquely helping define and reduce the size of the feasible solution space.

In integer programming, it is often desirable to introduce constraints which, while appearing unnecessary, can greatly decrease solution time. This has been considered in [76] and [105]. Hence, the improved model can obtain optimal solution in shorter time as demonstrated in Sec 4.4.3. The model formulation in Gnu Mathematical Programming Language (GMPL) is given in Appendix B.

4.4.2 Illustration 1: Yard with 10 blocks and 4 deployments

Here, we solve an instance of the YCSP model1. Consider a yard with 10 blocks, 10 RTGs each having 15 minutes capacity 4 times. One RTG is initially positioned in each block. The workload ranges from 0 to 28 minutes. The vertical rectangles are the blocks and the small horizontal rectangles are the RTGs. The initial positions of the RTGs are as shown in Figure 4.2.

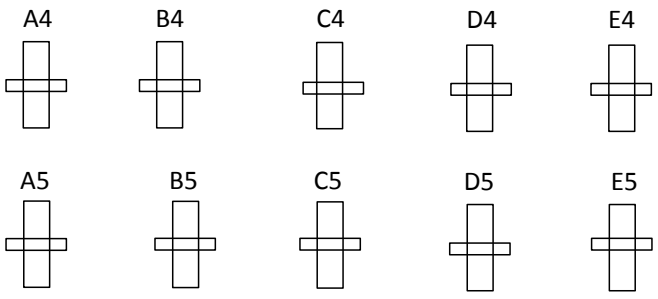


Figure 4.2: Initial RTG allocation to ten blocks in the yard.

After submitting the problem of Figure 4.2 to the solver, the final RTG positions after three deployments are as shown in Figure 4.3.

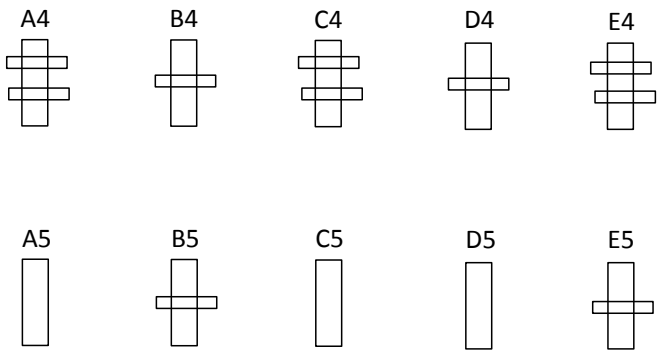


Figure 4.3: Final RTG position for ten blocks in the yard.

Note that the problem involves 761 variables, 590 constraints, 2974 non-zeros and the optimal solution for the LP relaxation is 11.94 minutes. The MILP solution for ten blocks is 36.5 minutes with 7.5 minutes attributable to works in the last deployment. The processing time of the solver is less than one second on a computer with specification given in Section 4.5. Three changes were made to the movement of RTGs in period 1 (from block C4 to block E4), (from block D4 to block E5) and (from block E4 to block C5). Five changes were made in period 2. In period 3, there were nine changes and five changes were made in the last period. The movements of the RTGs are shown in Appendix C.

4.4.3 Computational Experiments

Experiments were conducted to investigate the performance of the proposed model on different problem instances arising at the Port of Felixstowe. The

model is coded in Gnu Mathematical Programming Language (GMPL) and executed on an Intel Core i7-4790 3.60 GHz CPU with 16GB RAM. Each of these problems has been solved using GLPSOL and the results are given in Table 4.2, 4.3 and 4.4. The first two tables consist of six columns with column 1 been the planning period in Table 4.2 and the number of blocks in Table 4.3. Column 2 is the value of the objective function which is in terms of unfinished work (UW) in minutes, Columns 3 and 5 are the number of constraints, Column 4 and 6 are the CPU time in seconds.

4.4.3.1 Computation for different planning periods

The computation was done for six different planning period having ten blocks, ten RTGs and fifteen minutes deployment period. The choice of fifteen minutes is dictated by what is common in the real world and certainty at PoF. Indeed having a deployment period less than fifteen minutes is very uncommon because it restricts the movement of the RTG due to constraints (4.34). In view of this, we have evaluated the performance between the two models keeping the planning period between 60 minutes and 135 minutes. Both models were not able to return solution in less than 15 minutes for planning periods beyond 135 minutes. The results are presented in Table 4.2.

Table 4.2: Computational Results for a Yard having 10 Blocks

Planning	Unfinished	Model1		L and Z	
period	work	const	CPU time(s)	const	CPU time(s)
60	95.75	725	0.10	721	0.10
75	140.25	876	0.30	871	1.00
90	183.00	1027	0.80	1021	2.40
105	235.00	1178	11.80	1171	15.50
120	282.25	1329	15.30	1321	32.70
135	331.50	1480	60.40	1471	112.60

4.4.3.2 Comparing models on different yard sizes

Here, we compare the two models under small yard sizes but large planning period of 240 minutes. We use four block sizes and four RTGs, five blocks and five RTGs as well six blocks with six RTGs in the experiment. The yard size has been kept small due to the computational complexity arising from solving a large problem using the models. The workload distribution follows a uniform distribution U(0, 212). While both models returns the same unfinished work for each yard size, model1 return results in faster times.

Table 4.3: Computational Results for a Yard having 15 Blocks

	Unfinished	Model1		L and Z	
Blocks	work	const	CPU time(s)	const	CPU time(s)
4	49.50	617	0.00	601	0.10
5	71.50	852	0.20	836	1.00
6	98.75	1121	3.40	1105	4.80

4.4.3.3 Comparing models on large yard sizes

In this section we compare our model with that of L&Z model [72] on block sizes ranging from 12 to 30. The workloads are generated from a uniform

distribution of U(0,82). While both models return the same unfinished work (UW) in all cases, our alternative model return results between (1.5 - 6) times faster as presented in Table 4.4.

Table 4.4: Computational Results for large Block sizes

Blocks	12	14	15	16	18	20	22	24	25	26	28	30
UW	122.75	147.5	147.5	150	180.5	193.5	139.5	142	144	144.75	144.75	148.25
Model1	0.2	1.1	2.4	1.9	4.9	25.9	11.3	8.7	12.6	20.8	38.2	28.6
L&Z	1.2	2.5	3.6	5.3	15.6	33.5	16.6	22.4	19.6	33.6	82.8	32
L&Z / Model1	6	2.3	1.5	2.8	3.2	1.3	1.5	2.6	1.6	1.6	2.2	1.1

4.5 YCSP model2

In this section, we modify the model in [72] by changing the objective function and adding a new set of constraints. In practice, there is a trade off between finding the minimum unfinished work and also minimising the surplus capacity in the yard operation. Unfinished work arises when the RTG capacity is less than the workload in that block while there is surplus capacity when the available RTG capatity is more than the workload in the block. To address this imbalance, RTGs are expected to move from a block where there is less work to another block where there is more work. However, RTG movements are constrained by the travel time between blocks and limited to one movement in each deployment period. This results in loss of productivity. In view of this, we have included the surplus capacity in the objective function. The parameters and decision variables remain same as the two previous model discussed so far. The model is as follows;

YCSP Model2:

$$\min \qquad w1 \sum_{t=1}^{T} \sum_{i=1}^{B} W_{it} + w2 \sum_{t=1}^{T} \sum_{i=1}^{B} W_{it}^{+} \tag{4.41}$$

subject to :

$$\sum_{j=1}^{B} X_{ijt} = \sum_{j=1}^{B} X_{ji(t-1)} \quad \text{for} \quad i = 1, 2, .., B, \quad t = 1, 2, ..., T \tag{4.42}$$

$$\sum_{j=1}^{B} X_{jit} \leq 2 \quad \text{for} \quad i = 1, 2, ..., B, \quad t = 0, 1, ..., T \tag{4.43}$$

$$W_{i(t-1)} + B_{it} - W_{it} - \sum_{j=1}^{B} (C - \tau_{ij}) X_{jit} + W_{it}^{+} = 0$$
$$\text{for} \quad i = 1, 2, ..., B, \quad t = 1, 2, ..., T \tag{4.44}$$

$$(C - \tau_{ij}) X_{jit} \geq 0 \quad \text{for} \quad i, j = 1, 2, ..., B, \quad t = 1, 2, ..., T \tag{4.45}$$

$$\sum_{i=1}^{B} \sum_{j=1}^{B} X_{jit} = \sum_{i=1}^{B} X_{ii0} \quad \text{for} \quad t = 1, 2, ..., T \tag{4.46}$$

$$X_{ij0} = 0 \quad \text{for} \quad i, j = 1, 2, ..., B \quad \text{and} \quad i \neq j \tag{4.47}$$

$$W_{i0} = 0 \quad \text{for} \quad i = 1, 2, ..., B \tag{4.48}$$

$$X_{ii0} = 1, \quad \text{for} \quad i = 1, 2, ..., B \tag{4.49}$$

$$W_{it}^{+} \geq 0, \quad W_{it} \geq 0 \quad \text{for} \quad i = 1, 2, ..., B, \quad t = 1, 2, ..., T \tag{4.50}$$

$$X_{ijt} \geq 0, \text{integer for} \quad i, j = 1, 2, ..., B, \quad t = 1, 2, ..., T \tag{4.51}$$

4.5.1 Model Modification

There are two adjustments made to the model in [72]. The objective function has been changed to include the total surplus capacity at the end of the planning period. The weights $w1$ and $w2$ are attached to the unfinished work and the

surplus capacity indicating their relative importance in the objective function. Generally speaking, the weight $w1$ attached to unfinished work is expected to be higher than $w2$ the weight attached to surplus capacity. The two weights add up to one i.e., $w1 + w2 = 1$. Constraint (4.46) is the total RTG available to work in each deployment period.

4.5.2 Computational Experiments

Experiments were conducted to investigate the performance of the proposed model on different problem instances arising at the Port of Felixstowe. The model is coded in Gnu Mathematical Programming Language (GMPL) and executed on an Intel Core i7-4790 3.60 GHz CPU with 16GB RAM. Each of these problems has been solved using GLPSOL and the results are given in Table 4.5, 4.6 and 4.7. The tables consist of fifteen columns with column 1 been the number of blocks. Columns 2 and 3 are objective values (OV) which is the total unfinished work and the surplus capacity (SC) in minutes for L&Z model at the end of a planning period, Columns 4 and 5 are OV which is the unfinished work and the SC in minutes for Model 1 in the same period, Columns 6 to 15 are OV and the SC in minutes for Model 2.

Table 4.5: Computational Results for 15 mins deployment period

	L&Z		Model1		Model2 (w1 / w2=weight for SC)									
	OV	SC	OV	SC	OV	SC	OV	SC	OV	SC	OV	SC	OV	SC
Block						0.5		0.4		0.3		0.2		0.1
10	95.75	23	95.75	23	59.13	15	66.65	23	73.92	23	81.2	23	88.48	23
12	122.75	29.5	122.75	29.5	75.88	21.5	85.45	29.5	94.78	29.5	104.1	29.5	113.43	29.5
14	147.5	36.5	147.5	36.5	91.75	26	103.1	36.5	114.2	36.5	125.3	36.5	136.4	36.5
15	147.5	41.4	147.5	41.5	94.25	33.5	105.1	41.5	115.7	41.5	126.3	41.5	136.9	41.5
16	150	44	150	44	96.75	33.5	107.6	44	118.2	44	128.8	44	139.4	44
18	180.5	64.75	180.5	64.75	112.88	24	130.6	30.25	143.48	54.75	156.15	54.75	168.83	54.75
20	193.5	86.75	193.5	86.75	120.38	33.5	137.75	33.5	154.78	44	168.95	66.75	181.73	66.75

Here we compare the three models for fifteen minutes deployment period over 1-hour planning period. The workload distribution follows a uniform distribution U(0, 29). OV is the weighted sum of the total unfinished work and SC in model2. Since the main goal is to minimise unfinished work, w1 is set to be higher than w2. If w1 is large compared to w2, the optimal decision will be to avoid as much unfinished work as possible. These weighted factors will be set by the port supervisor based on experience. It will be assumed for the purpose of analysis that policy of the port is to give unfinished work a weight of 90% and surplus capacity is given the weight 10%.

We observed that when unfinished work and surplus capacity are given equal weights, the results are not optimal for all block sizes. This shows that such policy should not be adopted. As block sizes increases from 16 to 18 and 20, the model2 still return suboptimal values. The deployments in these cases were more than in the other two models for the same problems. Other cases are optimal for all block sizes and they have the same deployments as L&Z and Model 1.

Table 4.6: Computational Results for 20 mins deployment period

	L&Z		Model1		Model2 (w1 / w2=weight for SC)									
	OV	SC	OV	SC	OV	SC	OV	SC	OV	SC	OV	SC	OV	SC
Block						0.5		0.4		0.3		0.2		0.1
10	40.75	30.25	40.75	30.25	34	16	36.75	30.25	37.6	30.25	38.65	30.25	39.7	30.25
12	45.25	36.25	45.25	36.25	39.25	22	41.65	36.25	42.55	36.25	43.45	36.25	44.35	36.25
14	55.25	37.5	55.25	37.5	44.88	22.25	48.15	37.5	49.93	37.5	51.75	37.5	53.48	37.5
15	73.25	38.25	73.25	38.25	54.25	24	59.25	38.25	62.75	38.25	66.25	38.25	69.75	38.25
16	119	38	119	38	77	23.75	86.6	38	94.7	38	102.8	38	110.9	38
18	122	42.05	122	42.05	80.69	27.8	90.22	42.05	98.25	42.05	106.27	42.05	114.3	42.05
20	128.75	43.97	128.75	43.97	84.86	29.72	94.84	43.97	103.32	43.97	111.79	43.97	120.27	43.97

In Table 4.6, we compare the three models for twenty minutes deployment period over 1-hour planning period. The workload distribution follows a

uniform distribution U(0, 29). We observed that model2 return optimal results and same deployments as the two other models except when there are equal weights for unfinished work and surplus capacity. Once again, this indicates that equal weights should not be adopted as a policy.

Table 4.7: Computational Results for 30 mins deployment period

	L&Z		Model1		Model2 (w1 / w2=weight for SC)									
	OV	SC	OV	SC	OV	SC	OV	SC	OV	SC	OV	SC	OV	SC
Block						0.5		0.4		0.3		0.2		0.1
10	22	17	22	17	19.5	17	20	17	20.5	17	21	17	21.5	17
12	24.5	20.5	24.5	20.5	22.5	20.5	22.9	20.5	23.3	20.5	23.7	20.5	24.1	20.5
14	27.5	22.75	27.5	22.75	25.13	22.75	25.60	22.75	26.08	22.75	26.55	22.75	27.03	22.75
15	27.75	25	27.75	25	26.37	25	26.65	25	26.93	25	27.2	25	27.48	25
16	35.5	26.5	35.5	26.5	31	26.5	31.9	26.5	32.8	26.5	33.7	26.5	34.6	26.5
18	44.5	28.75	44.5	28.75	36.62	28.75	38.2	28.75	39.78	28.75	41.35	28.75	42.93	28.75
20	50	32.25	50	32.25	41.13	32.25	42.9	32.25	44.68	32.25	46.45	32.25	48.23	32.25

In Table 4.7, we compare the three models for thirty minutes deployment period over 1-hour planning period. The workload distribution follows a uniform distribution U(0, 29). Here, we observe that regardless of the weightings, model2 returns optimal values for all blocks considered. Hence, as the deployment period increases, model2 returns the same result and deployments as the two other models.

We can infer from the analysis above that the shorter the deployment period, the more likely is surplus capacity as the RTGs do not have enough time to move between blocks. The longer the deployment period however, the less likely is surplus capacity as there is enough time for the RTGs to move between blocks.

4.6 Computational aspects of YCSP and other combinatorial problems

We have mentioned that YCSP instances of practical dimensions are solved to optimality in realistic times, using an exact approach. This is good news given that it allows for frequent solutions over the horizon of the scheduling of RTGs to mitigate any uncertainty and, in particular, that due to unforseen changes in the arrival of trucks to the storage bays. This rather unexpected "easiness" in the solution of the different models of YCSP considered here prompts us to consider if there are no underlying properties of the problem that make it so.

It is well-known that some combinatorial optimisation problems, which from the outset are difficult, turn out to be easy. These are the transportation and the assignment problems of course and their variants as well as network problems formulated as linear programs. More striking cases, however, are found among the class of problems with 0, 1, -1; these are easy to solve when these matrix are Totally Unimodular or TUM [34]. A sufficient condition for total unimodularity for a given 0,1,-1 matrix is the so called P property which can be stated as follows [108]:

1. Each element of M is 0, 1 or -1.

2. Each column contains at most two non-zero elements.

3. The rows of M can be partitioned into two subsets P_1 and P_2 such that:

    - if a column contains two non-zero elements of the same sign, one

element is in each of the subsets.

- if a column contains two non-zero elements of opposite sign, both elements are in the same subset.

Definition 4.6.1 Given a 0, 1, -1 matrix M, it is TUM if every square submatrix of M has determinant equal to 0 or ± 1

Let us stress that when this property is not present then nothing can be said about the easiness of solution of the problem with that matrix. In other words, it is not necessary. Let us also define easiness of solution as the predisposition an MIP or MILP has to be solved as an LP and still yield integer optimum values where required. This means that for these problems the integrality constraints are redundant. Another class of integer problems which is easy to solve is that with (0, 1) matrices the columns of which can be permuted in such way that all ones in every row are blocked together. This property is known as the Consecutive Ones Property, or C1P [78].

Moreover, it is now also known that pure IP problems with (0, 1) matrices, submatrices of which have the C1P can be treated as MILP's, thereby exploiting the columns of the "nice" submatrix. In other words, the variables whose columns form the submatrix can be relaxed to be continuous. Such problems are quicker to solve than their pure IP versions, [40].

4.6.1 Conjecture

Our YCSP programs do not have 0,-1, +1 matrices and therefore the above rules do not apply. However, from the practical point of view, some of the instances we have solved, when we relaxed them to LP status, yield integer solutions. It can be understood that, when solved as MIPs, using say B&B, then there is no need to branch on the variables, which take readily integer values, thus potentially reducing the size of the search tree, and by the same token reducing the amount of work required to solve the problem to optimality. We, therefore, conjecture the following.

The observed relative "easiness" of solution of the YCSP instances we have solved is due to an underlying property which has similar effects to those of the above properties namely, the P Property and the C1P.

Note that although we do not yet have a formal proof of the above, we can suggest a sketch for a potential proof plan based on the fact that, MILP's are cheaper to solve than pure IP's of the same size. This, can be established as follows. Consider the scenario where all n variables of the pure IP have to be branched upon before an integer solution is found. If we further assume that the integer solutions are found after that, then we would have solved 2n LP's to obtain the optimum integer solution. Now, if at least one of the variables of the MILP were to be integer and not requiring branching, then the number of required LP's to solve cannot be more than 2n-1.

Note that the concept of matrices almost satisfying the C1P has been considered

before as in [87]. However, that was in the context of (0, 1) matrices of the Set Covering Problem. This problem is of course strongly NP-Hard [33].

4.7 Summary

RTGs are important tools widely used in container ports to manage the storage yard. The efficiency with which the yard is managed is largely dependent on how efficient is their scheduling and deployment. There are a number of attempts at solving this problem; we have reviewed most of them in this thesis. However, they make some unrealistic assumptions such as forecasting the entire workload for the planning period. These assumptions limit seriously the usability of these models particularly when we know that facilities and the demands on them are continually expanding. Workloads arise dynamically and thus unpredictable especially for import containers where the external trucks can have a delay.

We have therefore introduced two alternative models which does without these assumptions. We have illustrated these models and experimented with them on realistic problems as they arise in the Port of Felixstowe, the largest container port in the UK and one of the largest in Europe. The results show that our models are viable. It is also interesting to note that the solver returns results in realistic times, which means the models can be solved frequently enough to mitigate any uncertainty due to the random distribution of vessel arrival times and delay by external trucks.

The choice of a valid inequality/constraint greatly influence the performance of the solver. However, adding the right constraint can speed up the solution time since it reduces the solution space. It is expected that the closer the convex hull of the MILP is close to its LP relaxation, the faster B&C algorithm can get an optimal solution.

Chapter 5

The Container Reshuffle Problem

5.1 Introduction

Containers are often stored temporarily in the yard between the time they arrive at the port either through vessels in the case of import and through external trucks in the case of export. In practice, containers for loading are placed in the export area and those unloaded from ships are placed in the import area [41]. Some ports stack export containers close to the quay and import containers close to the landside gate. Others have a dedicated area for marshalling containers just unloaded from or to be loaded onto vessels. Fast access to stored containers is a major concern in container terminals. Choosing an appropriate location for a container that is to be relocated is essential in reducing the subsequent reshuffles. There are many ways of tackling this problem.

When containers arrive there is need to have a storage strategy to stack them in order to make retrieval efficient. Container types are stored in different places

(for instance 20-foot containers are stored separate from 40-foot containers). Where they are stored together, the 20-foot containers are stacked beneath the 40-foot because the latter have a higher priority when loading onto a vessel. The storage location could also be based on whether the container is full or empty, the destination of vessel and even their weight. After storage, containers can be reshuffled/pre-marshalled in advance of retrieval. The aim here is to change the initial layout of the block to a desired layout to facilitate retrieval [50].

If the containers are placed exactly where they can quickly be accessed, the retrieval process will be efficient. However, this is hardly the case due to many reasons especially inaccurate information as to when the containers will be retrieved when storing them [66]. Other reasons include change in vessels arrival time due to delay, external trucks arriving late due to traffic and the vessels stowage plan amongst others. Since only containers at the top of a stack are accessible, there is usually need to reshuffle containers in order to retrieve the desired container beneath them. This process takes time and thus hampers the operation of yard cranes YCs which will delay the trucks and/or the vessels.

We are interested in retrieving all the containers in a bay at minimal reshuffle. However, decision on where to place a reshuffled container is not as easy as it looks because it affects subsequent retrievals. Even in a static case where there is no arrival of new containers while stacking, the problem is still dynamic due to the fact that the configuration of the bay changes each time there is a retrieval. In [55], it was suggested that in order to minimise reshuffles,

the storage location of incoming container should be well assigned and the location of a reshuffled container should also be determined. For the purpose, YC drivers are given work orders in the form of a movement sequence. The movement sequence contains the order of container movements, instructing the driver which container to move, where and when to move them [64].

Containers can be classified using different attributes such as weight, port of destination, length, being inbound or outbound and full or empty. This determines where they are stored for easy retrieval. For instance, a container destined for a farther port has a higher priority when loading onto a vessel since it will be unloaded later than a container destined for a port that is nearer. Heavier containers are stored in higher tiers in the yard since they are loaded in lower parts of vessels and are thus retrieved earlier. Containers that are stored earlier are likely to need earlier retrieval but would be buried under later arriving containers. Hence, there is need for reshuffle.

5.2 Model Assumptions

1. Each container can only be assessed from the top. Since containers are stacked on top of each other, a container can only be moved if container(s) above it has been moved.

2. The number of column is bounded by C. The total number of columns is known and defined as C.

3. The number of tier is bounded by T. The total number of tiers is known

and defined as T.

4. There is no anticipated reshuffle, a container is moved only when necessary. At each stage of retrieval, a container is moved if and only if it is blocking a desired container.

5. For each retrieval, a container is moved only once. Once a container has been moved from a column at any stage, it cannot be moved again at that stage.

5.3 Problem Description

Due to the limited space in a container yard, containers are stacked on top of each other. However, the higher the stack the higher the probability for reshuffle. Hence, there is a trade off between stacking containers on top of each other and using more space. The time lag between when these containers are stored and when they are retrieved results in improper location. The result is containers that are needed earlier are underneath containers that are needed later. In order to access the desired container, we need to reshuffle the container(s) on top.

The aim is to reduce the total number of reshuffles in the future by picking the best place for a container reshuffled. Reshuffles are only done within bays for safety and operational reasons. When a crane picks a container with its hoist/spreader either for retrieval, storing or reshuffle, it only moves the

containers vertical or horizontal while frame of the crane is kept still [102]. The container reshuffle problem is made up of three basic decisions listed below;

1. **Which** container to move.
2. **When** to move it.
3. **Where** to move it to.

The configuration of a bay can be defined by the number of columns, the number of tiers, the number of containers and their position in the bay. Let S be the number of containers in a bay with C columns and T tiers. The maximum number of container S that should be in the bay for reshuffle to be feasible in the worst case scenario can be defined as follows;

$$S \leq CT - (T - 1) \tag{5.6}$$

where CT is the number of containers that can be in the bay when it is full and $(T-1)$ is the number of empty slot to allow for reshuffle. Figure 5.1, shows a bay with six columns, four tiers and twenty-one containers with their positions.

tier 4	4	3	10		16	
tier 3	14	18	19	20	13	
tier 2	2	17	11	1	6	9
tier 1	21	5	8	7	15	12
	col 1	col 2	col 3	col 4	col 5	col 6

Figure 5.1: A Typical Bay Configuration

Containers are ranked from 1 to S, indicating the priority order for retrieval. Where containers are not stored while retrieval is going on, we have a static state i.e. the number of containers only decreases. We have a dynamic state when containers are stored while retrieval is in process. In this study, we consider both static and dynamic states. Each time a container is retrieved, the configuration of the bay changes due to reshuffles. This dynamic nature of the problem even for a static case shows complexity of the problem. Even for a small problem, the number of state grows exponentially with the number of container reshuffles.

5.4 Mathematical Formulation

According to [95], let $\{x_1, x_2, x_3..., x_{s-1}\}$ be the stages (updated state of the bay) for retrieval of all containers. At each stage, the incremental reshuffles due to retrieval of container s is y_{si}.

$$y_{si} = \begin{cases} 1, & \text{if container } i \text{ is reshuffled in retrieving container } s \\ 0, & \text{otherwise} \end{cases}$$

The number of reshuffles necessary to retrieve container s in stage x_s can be defined as follows;

$$x_s = \sum_{i}^{S} y_{si} \tag{5.7}$$

The total number of reshuffles in retrieving all the containers is thus;

$$\sum_{s=1}^{S-1} \sum_{i=s+1}^{S} y_{si} \tag{5.8}$$

The problem therefore is to minimise the total number of reshuffles necessary in retrieving all containers from a given bay.

Decision Variables:

$$x_{sict} = \begin{cases} 1, & \text{if container } i \text{ is at tier } t \text{ of column } c \text{ at the beginning of stage } s \\ 0, & \text{otherwise} \end{cases}$$

$$y_{si} = \begin{cases} 1, & \text{if container } i \text{ is reshuffled in retrieving container } s \\ 0, & \text{otherwise} \end{cases}$$

$$w_{sij} = \begin{cases} 1, & \text{if container } i \text{ and } j \text{ are reshuffled during stage } s \text{ and container } j \\ & \text{is at a higher tier than container } i \text{ before reshuffling} \\ 0, & \text{otherwise} \end{cases}$$

Parameters:

S	The total number of containers initially stored in the bay
T	The total number of tiers in the bay
C	The total number of columns in the bay
s	Index of the container to be retrieved and also the stage for retrieving the container $1 \leq s \leq S$
i, j	Indexes for the containers under consideration, $1 \leq i, j \leq S$
t	Index for tiers in a column, counting from the lowest tier, $1 \leq t \leq T$
c	Index for the columns in the bay, $1 \leq c \leq C$
A_{1ict}	The initial location of the containers in the bay. If container i is stored in tier t of column c, $A_{1ict} = 1$; otherwise, $A_{1ict} = 0$

The Model:

$$\min \sum_{s=1}^{S-1} \sum_{i=s+1}^{S} y_{si} \tag{5.9}$$

subject to :

$$\left(1 - \sum_{t=1}^{T} x_{ssct}\right)T + y_{si} \geq \left(\sum_{t=1}^{T} tx_{sict} - \sum_{t=1}^{T} tx_{ssct}\right)/T$$
$$1 \leq s < i \leq S, 1 \leq c \leq C; \tag{5.10}$$

$$\left(\sum_{t=1}^{T} tx_{ssct} - \sum_{t=1}^{T} tx_{sict}\right)/T \leq 1 - y_{si}$$
$$1 \leq s < i \leq S, 1 \leq c \leq C; \tag{5.11}$$

$$\sum_{c=1}^{C}\sum_{t=1}^{T} x_{sict} = 1, \qquad 1 \leq s \leq i \leq S; \tag{5.12}$$

$$\sum_{i=s}^{S} x_{sict} \leq 1, \qquad 1 \leq s \leq S, 1 \leq c \leq C, 1 \leq t \leq T; \tag{5.13}$$

$$\sum_{i=s}^{S} x_{sict} \leq \sum_{i=s}^{S} x_{sic,t-1}, \quad 1 \leq s \leq S, 1 \leq c \leq C, 2 \leq t \leq T; \tag{5.14}$$

$$\sum_{t=1}^{T} x_{s+1,ict} + \sum_{t=1}^{T} x_{ssct} \leq 2 - y_{si}, \quad 1 \leq s < i \leq S, 1 \leq c \leq C; \tag{5.15}$$

$$2 - y_{si} - y_{sj} + w_{sij} \geq \left(\sum_{c=1}^{C}\sum_{t=1}^{T} tx_{sjct} - \sum_{c=1}^{C}\sum_{t=1}^{T} tx_{sict}\right)/T$$
$$1 \leq s < i \leq S, 1 \leq s < j \leq S, i \neq j; \tag{5.16}$$

$$y_{si} + y_{sj} + w_{sij} \leq 3 + \left(\sum_{c=1}^{C}\sum_{t=1}^{T} tx_{sjct} - \sum_{c=1}^{C}\sum_{t=1}^{T} tx_{sict}\right)/T$$
$$1 \leq s < i \leq S, 1 \leq s < j \leq S, i \neq j; \tag{5.17}$$

$$w_{sij} \leq y_{si}, \qquad 1 \leq s < i \leq S, 1 \leq s < j \leq S, i \neq j; \tag{5.18}$$

$$w_{sij} \leq y_{sj}, \qquad 1 \leq s < i \leq S, 1 \leq s < j \leq S, i \neq j; \tag{5.19}$$

$$\sum_{t=1}^{T} tx_{s+1,ict} - \sum_{t=1}^{T} tx_{s+1,jct} \geq -T(1 - w_{sij}) - T(1 - y_{si})$$
$$-T(1 - y_{sj}) - T\left(1 - \sum_{t=1}^{T} x_{s+1,ict}\right)$$
$$1 \leq s < i \leq S, 1 \leq s < j \leq S, i \neq j, 1 \leq c \leq C; \tag{5.20}$$

$$x_{s+1,ict} - x_{sict} \geq -y_{si}, \quad 1 \leq s < i \leq S, 1 \leq c \leq C, 1 \leq t \leq T; \tag{5.21}$$

$$x_{sict} - x_{s+1,ict} \geq -y_{si}, \quad 1 \leq s < i \leq S, 1 \leq c \leq C, 1 \leq t \leq T; \tag{5.22}$$

$$x_{1ict} = A_{1ict}, \quad 1 < i \leq S, 1 \leq c \leq C, 1 \leq t \leq T; \tag{5.23}$$

$$x_{sict} = A_{1ict}, \qquad 2 \leq s \leq min\{i, s_i\}$$

$$s \leq i \leq S, 1 \leq c \leq C, 1 \leq t \leq T; \tag{5.24}$$

$$y_{si} \in \{0,1\}, \qquad 1 \leq s < i \leq S; \tag{5.25}$$

$$w_{sij} \in \{0,1\}, \quad 1 \leq s < i \leq S, 1 \leq s < j \leq S, i \neq j; \tag{5.26}$$

$$x_{sict} \in \{0,1\}, \quad 1 \leq s \leq i \leq S, 1 \leq c \leq C, 1 \leq t \leq T; \tag{5.27}$$

Constraints (5.10) and (5.11) determine the reshuffle variable y_{si}. $\sum_{t=1}^{T} tx_{sict}$ and $\sum_{t=1}^{T} tx_{ssct}$ represent the position of containers i and j in column c tier t, while $\sum_{t=1}^{T} x_{ssct}$ is one if container s is in column c tier t at stage s. If container i is above container s, the RHS is a value less than one and the expression in bracket in LHS is zero. This forces y_{si} to be one i.e., container i is reshuffled in retrieving container s in Constraints (5.10). On the other hand, if container s is above container i in constraint (5.11), there will be no need for reshuffle hence y_{si}=0 and the LHS of the equation is less than one. Constraints (5.12) ensure that each container occupies only one spot. At each stage of retrieval, each container i, $i \geq s$ can be traced to one slot. Constraints (5.13) implies not more than one container can be at a slot i.e., a slot is either empty or has a container. The fact that containers cannot float is defined in Constraints (5.14). This means, if in column c, there is a container in tier t>1 then, there must be a container below it.

Constraints (5.15) imply a reshuffled container cannot be in the same column after reshuffle. If container i is above container s and both are in the same column then, container i must be reshuffled i.e., y_{si} and $\sum_{t=1}^{T} x_{ssct}$ are both one.

This forces $\sum_{t=1}^{T} x_{s+1,ict}$ to be zero i.e., container i cannot be in column c at the next stage $s + 1$. Suppose container i, j and s are in the same column and container i and j are above container s. If container j is above container i before reshuffle therefore, the expression in bracket on RHS of constraints (5.16) and (5.17) will be a value less than one. Since y_{si} and y_{sj} are one, this forces w_{sij} to be one. In constraints (5.18) and (5.19), if either container i or j is not reshuffled or container j is not above container i then w_{sij} =0.

Constraints (5.20) address the relative positions of two containers before and after both are reshuffled. Suppose container i, j and s are in the same column and container j is above container i before reshuffle. If containers i and j are reshuffled in retrieving container s then, y_{si}, y_{sj} and w_{sij} are all one. If both containers i and j are reshuffled to same new column therefore, container i will be above container j in the new column. Constraints (5.21) and (5.22) ensure that a container not reshuffled keep its position. If container i is not reshuffled at stage s therefore $y_{si} = 0$. This means $x_{s+1,ct} = x_{sict}$. If the container is reshuffled, it is stored in a position higher than those already in the column. Constraints (18) assign known values of x_{1ict} to slots in the initial configuration and this is updated at each stage of retrieval i.e., the lowest ranked container is retrieved at each stage. Constraints (5.23) to (5.24) are self explanatory. The model formulation in GMPL is given in Appendix D.

5.4.1 Model Modification

Constraint (5.23) and (5.24) force the containers to keep the same location they are at the first stage of retrieval. In view of this, we have removed them and introduced a bay configuration constraint to reflect the change arising from each container reshuffled. Constraints (5.23) and (5.24) are now replaced by constraints (5.28) below.

$$x_{1ict} = A_{ict}, \quad 1 < i \leq S, 1 \leq c \leq C, 1 \leq t \leq T \tag{5.28}$$

The modified model reduces the number of constraints by $(S-1)SCT$.

5.4.2 Illustration

Consider a bay with seven containers, three columns and three tiers. The initial position is as shown in stage (a) of Figure 5.2. The first reshuffle is done in stage (b) where container 7 moves from top of container 1 to top of container 2. At stage (c), container 1 is retrieved. Container 7 moves again from top of container 2 to the slot vacated by container 1; this is the second reshuffle at stage (d). At stage (e), container 2 is then retrieved. The third reshuffle is done in stage (f) where container 6 moves to top of container 7 and that is the last reshuffle. If container 6 had been moved to the top of container 4 that would have necessitated another reshuffle. This would have led to four reshuffles instead of three. Subsequently all the remaining containers can be retrieved without need for reshuffle.

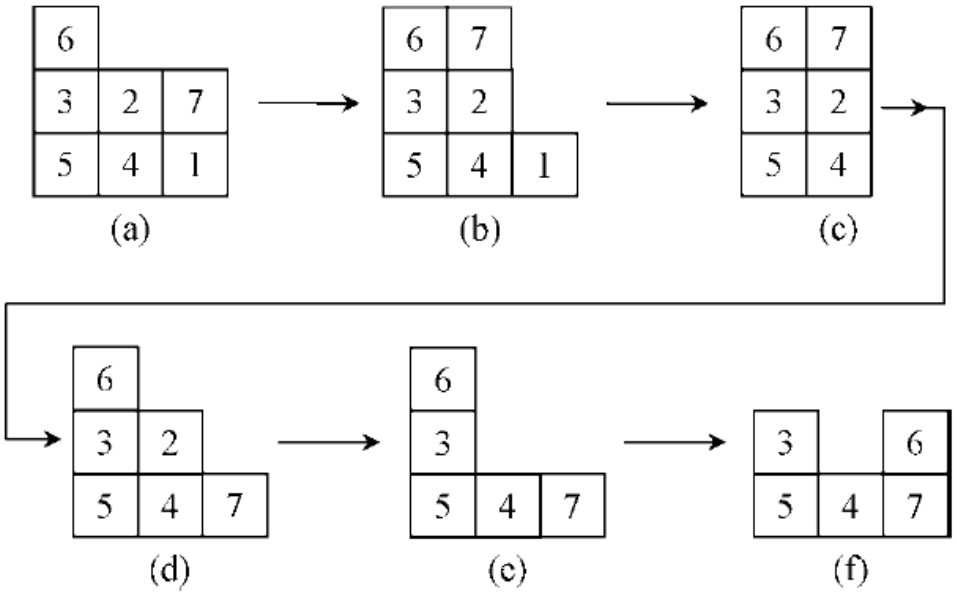


Figure 5.2: Reshuffle process

The way this instance can be solved by invoking GLPSOL after representing the instance in GMPL can be found in Appendices E. The results are as follows:

The problem involves 376 variables, 911 constraints and 7696 non-zeros. The optimal solution is three reshuffles. The processing time for the solver is less than one second on a computer with specification given in § 5.6 on the computational experiment.

5.5 A Heuristic Approach

Due to the computational complexity of the reshuffle problem, the exact approach is only able to solve small scale problems. This is at variance with real world situations where medium to large scale instances are encountered and fast results are required. There is thus need to seek an approximate approach solution. In this section, we review some heuristics found in literature and propose four new ones based on the least priority rule. They are referred to as LPH1, LPH2, LPH3 and LPH4.

5.5.1 Some existing heuristics

Consider a bay with S containers to be retrieved. The number of containers in the bay is initially numbered from 1 to S. The first container to be picked is container 1, once that is done, the remaining container is renumbered from 1 to S-1. At each stage, the container to pick is always container 1. The reshuffle process for the three heuristics as described in [95] is summarised in Algorithm 5.1. The algorithmn starts by copying initial matrix $\Lambda(tier, col)$ for evaluation. It then sets counter for the number of iteration and reshuffle. Thereafter, the algorithm searches for the minimum index number in the matrix and increases the counter by one each time it does the search. This minimum index number is refer to as the desired number (DC). Subsequently, $[index_r\ index_c]$ identifies the location of DC. The heuristic principle determine how the reshuffle process is undertaken until all containers are retrieved.

5.5.2 Pseudocode of Framework for Implementing RI, H1 and H2

Algorithm 5.1 RI, H1 and H2 Framework

Generate initial bay matrix elements $\Lambda(tier, col)$ ▷ Input matrix
$\Sigma(tier, col) \leftarrow \Lambda(tier, col)$ ▷ copy initial matrix for evaluation
$Iteration \leftarrow 0$ & $NumResh \leftarrow 0$ ▷ variables initialization
$n \leftarrow input(Enter\ a\ number :)$ ▷ RI = 1; H1 = 2; and H2 = 3
while $\min(\Lambda(\Lambda > 0)) <= \max(\Lambda(\Lambda > 0))$ **do** ▷ searches for min container
    $Iteration \leftarrow Iteration + 1$ ▷ Increases counter by one after each move
    $DesiredContainer\ (DC) \leftarrow \min(\Lambda > 0)$ ▷ Identify min. container
    $[index_r\ index_c] = \text{find}(\Lambda == DC)$ ▷ determine location of DC
    if $n == 1$ **then** ▷ apply Reshuffle Index heuristics
        $\left[\hat{\Lambda},\quad NumResh\right] \leftarrow$ **ComputeTypeRI**($tier$, $index_c$, Σ, DC, col, $NumResh$)
    else if (**then** $n == 2$) ▷ apply H1 heuristics
        $\left[\hat{\Lambda},\quad NumResh\right] \leftarrow$ **ComputeTypeH1**($tier$, $index_c$, Σ, DC, col, $NumResh$)
    else if (**then** $n == 3$) ▷ apply H2 heuristics
        $\left[\hat{\Lambda},\quad NumResh\right] \leftarrow$ **ComputeTypeH2**($tier$, $index_c$, Σ, DC, col, $NumResh$)
    else
        Terminate iteration wrong input
        Enter correct input number: RI = 1; H1 = 2; and H2 = 3
    end if
    Print number of reshuffle after each iteration $NumResh$
    $\Sigma(tier, col) \leftarrow \Lambda(tier, col)$ ▷ update matrix $\Sigma(tier, col)$ for next iteration
end while
Return Σ

5.5.3 Reshuffle Index Heuristic, (RI)

This is based on the rule that a blocking container should be moved to a column having the least number of containers that have to be retrieved earlier than the blocking container as described in [81]. Identify the blocking container and compare it to the minimum numbered container in each column that has empty space. Put the blocking container in column that have the least number of container to be retrieved earlier than the blocking container. Where there is

a tie, the container is put in the column with higher tier and arbitrary choice for a further tie. The pseudocode of RI is summarised in Algorithm 5.2 and Algorithm 5.3 determines where to put the blocking container using Rule1:

5.5.3.1 Function for Computing ComputeTypeRI

Algorithm 5.2 Function RI

Func($\hat{\Lambda}$, $NumResh$) ← **ComputeTypeRI**($tier$, $index_c$, Σ, DC, col, $NumResh$)
for $j = 1 : tier$ **do** ▷ check for DC in the Col from top
  if $\Sigma(j, index_c) == DC$ **then** ▷ where the cont is the DC
    $\Sigma(j, index_c) = 0$ ▷ retrieve the DC
    break
  else if $\Sigma(j, index_c) \neq DC$ && $\Sigma(j, index_c) \neq 0$ **then** ▷ if not DC
    $k \leftarrow \Sigma(j, index_c)$ ▷ cont is a BC
    $Col_j \leftarrow$ **Rule1** (Σ, k, col) ▷ choose Col to put the BC
    $K \leftarrow \Sigma(:, Col_j)$ ▷ Col to put the BC
    $\Sigma \leftarrow$ **Replacer** $(Col_j, k, K, \Sigma, tier)$ ▷ det where to put BC in Col
    $\Sigma\,(j, index_c) \leftarrow 0$ ▷ move the BC
    $NumResh \leftarrow NumResh + 1$ ▷ increase number of reshuffle by one
    break
  else if $\Sigma(j, index_c) == 0$ **then** ▷ where there is no cont at the top
  end if
end for
$\hat{\Lambda} \leftarrow \Sigma$
Return

Algorithm 5.3 Rule1

Rule1 (Σ, k, col)
if $k > n_c(i)$ **then** ▷ Condition for empty Col.
  move k into empty Col
end if
if $Len(holder) == 1$ **then** ▷ Condition for one Col.
  move k into only Col satisfying condition
  return
else if $Len(holder) > 1$ **then** ▷ Condition for more than one Col.
  move k into Col with highest number of containers
  return
end if
$\Sigma(:, col) \leftarrow k$ ▷ Assign BC to Col.
$\hat{\Sigma} \leftarrow \Sigma$
Return

5.5.4 H1 Heuristic

H1 is based on two conditions as described in [95]. The first condition is to identify the blocking container referred to as k and compare it to the minimum numbered container referred to as n_c in each column c that has empty space. Put k in column c that satisfies $k < n_c$ and arbitrarily decide on which is the closest column where there is a tie. The total number of containers in the bay is given the value S. A column with all empty space n_c is given value $S + 1$ thereby having the highest possible value in the bay and the blocking container will always move there, if it exists. Where there is no column that satisfies the above condition which means there is no column c where $k < n_c$.

For the second condition, H1 will put k in the column with the minimum RI. This is the total number of container that has a lower number than k in each column that has empty space. Where there is a tie, put k in the closest column. The decision on where to put the container is done arbitrary. This is summarised in Algorithm 5.4 as Function H1. Where to put the blocking container is determine in Algorithm 5.5 using Rule2:

5.5.4.1 Function for Computing ComputeTypeH1

Algorithm 5.4 Function H1

Func($\hat{\Lambda}$, $NumResh$) ← **ComputeTypeH1**($tier$, $index_c$, Σ, DC, col, $NumResh$)
for $j = 1 : tier$ **do** ▷ check for DC in the Col from top
  if $\Sigma(j, index_c) == DC$ **then** ▷ where the cont is the DC
    $\Sigma(j, index_c) = 0$ ▷ retrieve the DC
    break
  else if $\Sigma(j, index_c) \neq DC$ && $\Sigma(j, index_c) \neq 0$ **then** ▷ if not DC
    $k \leftarrow \Sigma(j, index_c)$ ▷ cont is a BC
    $Col_j \leftarrow$ **Rule2** (Σ, k, col) ▷ choose Col to put the BC
    $K \leftarrow \Sigma(:, Col_j)$ ▷ Col to put the BC
    $\Sigma \leftarrow$ **Replacer** $(Col_j, k, K, \Sigma, tier)$ ▷ det where to put BC in Col
    $\Sigma\,(j, index_c) \leftarrow 0$ ▷ move the BC
    $NumResh \leftarrow NumResh + 1$ ▷ increase number of reshuffle by one
    break
  else if $\Sigma(j, index_c) == 0$ **then** ▷ where there is no cont at the top
  end if
end for
$\hat{\Lambda} \leftarrow \Sigma$
Return

Algorithm 5.5 Rule2

Rule2(Σ, k, col)
if $k > n_c(i)$ **then** ▷ Condition for empty Col.
  move k into empty Col
end if
if $Len(holder) == 1$ **then** ▷ Condition for one Col.
  move k into only Col satisfying condition
  return
else if $Len(holder) > 1$ **then** ▷ Condition for more than one Col.
  move k into nearest Col with least RI
  return
end if
$\Sigma(:, col) \leftarrow K$ ▷ Assign BC to Col.
$\hat{\Sigma} \leftarrow \Sigma$
Return

5.5.5 H2 Heuristic

H2 is also based on two conditions as described in [95]. The first condition is to identify the blocking container referred to as k and compare it to the minimum numbered container referred to as n_c in each column c that has empty space. Put k in column c that satisfies $k < n_c$ and arbitrarily decide on which is the closest column where there is a tie. The total number of containers in the bay is given the value S. A column with all empty space n_c is given value $S + 1$ thereby having the highest possible value in the bay and the blocking container will always move there, if it exists. Where there is no column that satisfies the above condition which means there is no column c where $k < n_c$.

For the second condition, H2 puts k in the column with the minimum BI. This is the column that has lower number of container that will block the minimum number container in a column if k is moved to that column. Where there is a tie, the decision on where to put the container is done arbitrarily. This is summarised in Algorithm 5.6 as Function H2 and the rule that determines where to put the blocking container using Rule3 is in Algorithm 5.7:

5.5.5.1 Function for Computing ComputeTypeH2

Algorithm 5.6 Function H2

Func($\hat{\Lambda}$, *NumResh*) ← **ComputeTypeH2**(*tier*, $index_c$, Σ, *DC*, *col*, *NumResh*)
for $j = 1 : tier$ **do** ▷ check for DC in the Col from top
  if $\Sigma(j, index_c) == DC$ **then** ▷ where the cont is the DC
    $\Sigma(j, index_c) = 0$ ▷ retrieve the DC
    break
  else if $\Sigma(j, index_c) \neq DC$ && $\Sigma(j, index_c) \neq 0$ **then** ▷ if not DC
    $k \leftarrow \Sigma(j, index_c)$ ▷ cont is a BC
    $Col_j \leftarrow$ **Rule3** (Σ, k, col) ▷ choose Col to put the BC
    $K \leftarrow \Sigma(:, Col_j)$ ▷ Col to put the BC
    $\Sigma \leftarrow$ **Replacer** $(Col_j, k, K, \Sigma, tier)$ ▷ det where to put BC in Col
    $\Sigma\,(j, index_c) \leftarrow 0$ ▷ move the BC
    $NumResh \leftarrow NumResh + 1$ ▷ increase number of reshuffle by one
    break
  else if $\Sigma(j, index_c) == 0$ **then** ▷ where there is no cont at the top
  end if
end for
$\hat{\Lambda} \leftarrow \Sigma$
Return

Algorithm 5.7 Rule3

Rule3(Σ, *k*, *col*)
if $k > n_c(i)$ **then** ▷ Condition for empty Col.
  move k into empty Col
end if
if $Len(holder) == 1$ **then** ▷ Condition for one Col.
  move k into only Col satisfying condition
  return
else if $Len(holder) > 1$ **then** ▷ Condition for more than one Col.
  move k into nearest least BI Col
  return
end if
$\Sigma(:, Col) \leftarrow k$ ▷ Assign BC to Col.
$\hat{\Sigma} \leftarrow \Sigma$ ▷ updated the array
Return

Once a blocking container have been identified and the column to put it is defined, Algorithm 5.8 summarise the pseudo-code that moves the container to a new position. The slot to put the container is checked from the bottom for

empty space.

Algorithm 5.8 Replacer

1: **Func**($\hat{\Sigma}$) ← **Replacer**(Col_j, k, K, Σ, $tier$)
2: **for** $i = tier : -1 : 1$ **do** ▷ Check from bottom where there is an empty slot
3: **if** $K(i) == 0$ **then** ▷ where slot is empty
4: $K(i) \leftarrow k$ ▷ assign BC to the slot
5: **break**
6: **end if**
7: $\Sigma(:, Col_j) \leftarrow K$ ▷ Col where the BC has been assigned
8: $\hat{\Sigma} \leftarrow \Sigma$ ▷ updated the array
9: **end for**
10: **Return**

5.5.6 The Least Priority Heuristic (LPH1)

The heuristic is based on the fact that highest numbered container should be retrieved last hence, it is given the least priority. The algorithm starts by imputing a matrix of dimension t by c representing the initial state of the bay with t being the number of tiers and c the number of columns. Empty slot is represented by zero which can be randomly generated or input manually. The mimimum numbered container is always the one to be retrieved at each stage hence, this indicates the priority order i.e retrieval is done in ascending order. At each stage, the minimum numbered container is identified and retrieved if there is no container blocking it. If there is a container blocking it, the container(s) has to be moved to another column.

The decision on where to put the blocking container is determined by calculating the sum of the reciprocal for containers in those columns where there is empty spaces. These will be compared, and the column that returns the lowest

sum of reciprocal is chosen as the column where the blocking container should be moved to. Where a column has all slots empty, the blocking container is placed in this column. This is done for all the containers blocking the desired container until there is no container blocking it at which stage such container can now be retrieved. Since at each stage the minimum numbered container is always the one to be retrieved, the algorithm updates the configuration and the process is repeated until all containers are retrieved. The pseudocode of LPH is summarised in Algorithm 5.9.

5.5.7 The Least Priority Heuristic (LPH2)

The blocking container k is compared to the minimum numbered container n_c in each column having empty spaces. Put k in a column that satisfies the condition $k < n_c$ and put k in the closest column where more than one column satisfies the condition. Where no column satisfies the condition ie $k > n_c$ for those columns, apply the least priority principle.

5.5.8 The Least Priority Heuristic (LPH3)

The blocking container k is compared to the minimum numbered container n_c in each column having empty spaces. Put k in a column that satisfies the condition $k < n_c$ by applying the least priority principle. Where no column satisfies the condition ie $k > n_c$ for those columns, apply the reshuffle index principle as in condition 2 of H1 heuristics.

5.5.9 The Least Priority Heuristic (LPH4)

Apply the first condition in LPH3 where $k < n_c$ and the blocking index principle where $k > n_c$.

5.5.10 Pseudocode for the Least Priority Heuristic

Algorithm 5.9 Least Priority Heuristic

```
1: Generate initial bay matrix Λ(tier, col)                    ▷ Input matrix
2: Σ(tier, col) ← Λ(tier, col)
3: Iteration ← 0  &  NumResh ← 0                               ▷ variables initialization
4: while min(Λ(Λ > 0)) <= max(Λ(Λ > 0)) do
5:     Iteration ← Iteration + 1
6:     DesiredContainer (DC) ← min (Λ > 0)                     ▷ Finds min. container
7:     [index_r index_c] = find(Λ == DC)                       ▷ Determine location of DC
8:     yCol ← Reshuffle(Λ, index_c, col)                       ▷ Col with min. sum of reciprocal
9:     Γ ← Λ(:, yCol)
10:    [Λ, NumResh] ← CountReshuffle(tier, index_c, Λ, DC, yCol, Γ, NumResh)
11: end while
12: Return
```

The illustration in Figure 5.3 shows how to retrieve thirteen containers from a bay in order of priority. The first container to be retrieved is number 1, second container number 2 until the last container number 13 is retrieved. Container 1 can be retrieved without the need for reshuffle since there is no blocking container at stage *a*. Container 2 however is blocked by container 7 at stage *b*. The decision as to where to put container 7 is determined by calculating the sum of reciprocal for the containers in column 1, 3, 4 and 6. The column that returns the least value is chosen as the column to put container 7 which is column 1. Once container 7 has been moved on top of container 13 at stage *c*, container 2 can now be retrieved at stage *d*. The process continues until all

containers are retrieved. All containers can be retrieved at stage i without need for further reshuffle.

Algorithm 5.10 Reshuffle

Func($yCol$) $\leftarrow$ **Reshuffle**($\hat{\Lambda}$, $DesireCol$, Col)
for $j = 1 : Col$ **do** ▷ check through the whole Cols
  if $DesireCol == j$ **then** ▷ Col of DC
    return
  else if $\hat{\Lambda}(1, j) == 0$ **then** ▷ Cols with empty spaces
    $\hat{\Lambda}(:, j) \leftarrow \frac{1}{\hat{\Lambda}(:, j)}$ ▷ cal the reciprocal of each entry
  end if
end for
$[NRow, NCol] \leftarrow find(\hat{\Lambda} == \infty)$ ▷ check for slot returning ∞
for $i = 1 : NRow$ **do**
  for $j = 1 : NCol$ **do**
    if $\hat{\Lambda}(NRow(i), NCol(j)) == \infty$ **then** ▷ where slot return ∞
      $\hat{\Lambda}(NRow(i), NCol(j)) = 0$ ▷ assign zero to such slot
    end if
  end for
end for
$yCol \leftarrow \min \sum(\hat{\Lambda})$ ▷ summing reciprocal of each Col and return min. value
Return

Algorithm 5.11 CountReshuffle

Func($Narray$, $NumResh$) $\leftarrow$ **CountReshuffle**($tier$, $index_c$ $\hat{\Lambda}$, DC, $yCol$, Γ, $NumResh$)
for $i = 1 : tier$ **do** ▷ check for DC in the Col from top
  if $\hat{\Lambda}(i, index_c) == DC$ **then** ▷ where the cont is the DC
    $\hat{\Lambda}(i, index_c) \leftarrow 0$ ▷ retrieve the DC
    break
  else if $\hat{\Lambda}(i, index_c) \neq DC$ && $\hat{\Lambda}(i, index_c) \neq 0$ **then** ▷ if not DC
    $k \leftarrow \hat{\Lambda}(i, index_c)$ ▷ cont is a BC
    $\hat{\Lambda}(i, index_c) \leftarrow 0$ ▷ mogve cont
    $\hat{\Lambda} \leftarrow$ **Replacer**($yCol$, k, Γ, $\hat{\Lambda}$, $tier$) ▷ det where to put BC in Col
    $NumResh \leftarrow NumResh + 1$ ▷ increase number of reshuffle by one
    break
  else if $\hat{\Lambda}(i, index_c) = 0$ **then** ▷ where there is no cont at the top
  end if
end for
$Narray \leftarrow \hat{\Lambda}$ ▷ Update the array
Return

The associated functions for calculating the least reciprocal for columns that have some zeros and determining the column to put the reshuffled container is shown in Algorithm 5.10. The total number of reshuffle after every iteration and where to put the blocking container is shown in Algorithm 5.11.

Algorithm 5.12 Bay Configuration Matrix Generator

Populate initial bay matrix with zeros
Determine number of empty container spaces (Δ)
Generate a row vector of size ($\Sigma(tier * col)$ with unique integers selected randomly
Initialise counter i
for $q = 1 : col$ **do**
    Generate temporary empty array (Ω)
    for $k = 1 : tier$ **do**
        $\Omega(k) \leftarrow \Sigma(i)$
        $i \leftarrow i + 1$
    end for
    $\Gamma(:, :, q) \leftarrow \Omega$ ▷ Assign values of row vector Ω
end for
Flip & Reshape $\Gamma(:, :, q)$ matrix to *Tier by Col matrix*
Generate random empty **slots** within *Tier by Col matrix*
for $j = 1 : \Delta$ **do**
    if $\Gamma(1, tier,$ **slots (j)**$) \neq 0$ **then**
        $\Gamma(1, tier,$ **slots (j)**$) \leftarrow 0$
    else
        $mat \leftarrow \Gamma(1, tier,$ **slots (j)**$)$
        $index \leftarrow find(mat == 0)$
        $\Gamma\,(1, (index - 1),$ **slots (j)**$) \leftarrow 0$
    end if
    $\Gamma(1, tier,$ **slots (j)**$) \leftarrow 0$
end for
Tier by Col matrix $\leftarrow$ *Flip* & *Reshape* $(\Gamma,\ tier,\ col)$
Return Σ

Algorithm 5.12 shows the procedure for generating the initial matrix input as bay configuration. Slots having no container are represented by zeros and those with containers are numbered from 1 to S indicating their priority order. The number of zeros for any given bay configuration can be determined based on the utilisation capacity desired. The algorithm will randomly generate a

different matrix for any bay with tier (t) as the number of rows and columns (c) as the number of columns.

5.5.11 Illustration

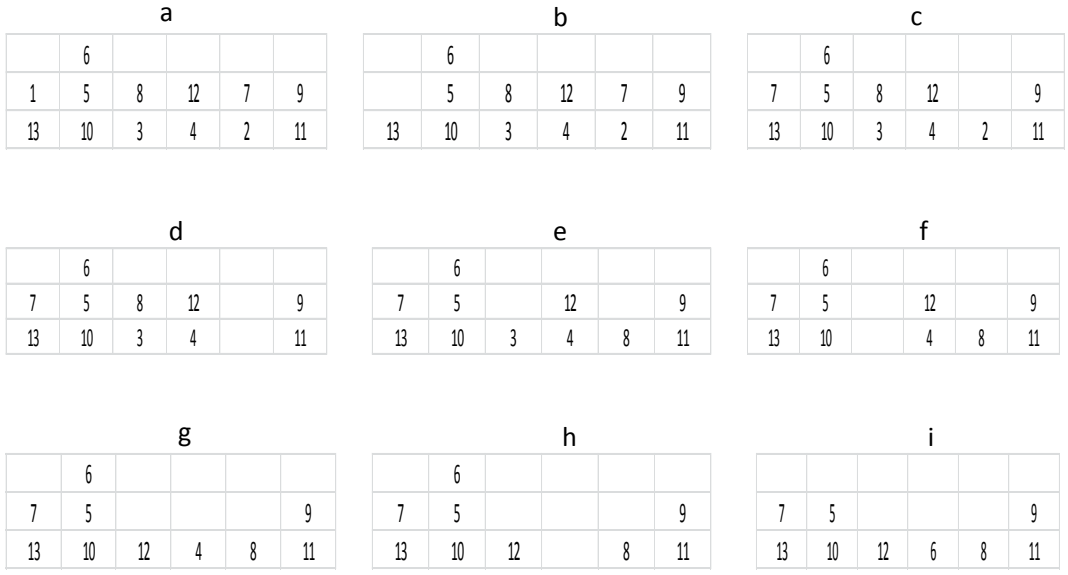


Figure 5.3: Heuristic Process

5.6 Computational Experiments

Experiments were conducted to investigate the performance of the model on different problem instances randomly generated. The model is coded in GMPL and executed on an Intel Core i7-4790 3.60 GHz CPU with 16 GB RAM. For an exact solution, each of the problems has been solved using GLPSOL while the heuristic is coded in MATLAB R2018b and the results are given in Tables 5.1 to 5.6 for the static case, 600 instances were randomly generated for 12 problem classes each having 50 cases. The problem class is represented as "the number

of columns - the number of tiers - the number of containers". 1-hour was set as the benchmark for a solution to be generated for each instance of the problems.

5.7 The Static case

Here we compare the results from the model solved with B&C and the heuristics for a small size problem defined as a bay with 50% utilisation capacity. We observe that they all return the same results for problem classes $(6-2-6)$ and $(6-3-8)$. However, as the problem size increases for $(6-4-11)$ and $(6-5-13)$ we begin to observe differences in results between the methods as the tier and number of container increase. The utilisation capacity is calculated as:

$$\frac{\text{Number of containers in the bay}}{\text{Total number of possible container reshuffles}} \tag{5.29}$$

Table 5.1: Performance between the Model and Heuristics for Small Size Problems

Classes	Model	H1	H2	RI	LPH1	LPH2	LPH3	LPH4
$6-2-6$	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84
$6-3-8$	1.52	1.52	1.52	1.52	1.52	1.52	1.52	1.52
$6-4-11$	3.32	3.46	3.46	3.46	3.48	3.42	3.56	3.58
$6-5-13$	4.32	4.64	4.68	4.68	4.64	4.52	4.82	4.82

Table 5.2: Performance between the Model and Heuristics for Small Size Problems

Classes	Model	H1_E	H2_E	RI_E	LPH1_E	LPH2_E	LPH3_E	LPH4_E
$6-2-6$	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84
$6-3-8$	1.52	1.52	1.52	1.52	1.52	1.52	1.52	1.52
$6-4-11$	**3.32**	**3.38**	3.42	3.40	3.48	**3.38**	3.42	3.48
$6-5-13$	**4.32**	4.52	4.54	4.62	4.64	**4.52**	4.64	4.70

In Table 5.1, we compare the model with original version of the heuristics for

small size problems. While Table 5.2 shows comparison with the extended versions. The extended version of each heuristic as defined in [102] is obtained by considering each possible slot for a reshuffled container based on the original rule of the heuristic and the one that gives the minimum possible reshuffle to empty the bay is chosen. They are denoted by *E* for each of the heuristic. For problem class $(6-2-6)$ and $(6-3-8)$, all the methods return the same average reshuffle. This is due to the fact that the number of tiers is still reasonably small; hence, reshuffle can still be achieved with relative ease. When the size increases to $(6-4-11)$ and $(6-5-13)$ we observe a difference in the average reshuffle between the methods because of the increase in tiers and number of containers. Note that all extended heuristics performed better than the original heuristic; LPH4_E has the least performance. H1_E and LPH2_E have the best performance among the heuristics for problem class $(6-4-11)$. LPH2_E has the best performance for problem class $(6-5-13)$.

Table 5.3: Performance between the Model and Heuristics for Medium Size Problems

Classes	Model	H1	H2	RI	LPH1	LPH2	LPH3	LPH4
$6-2-9$	1.52	1.52	1.52	1.52	1.52	1.52	1.52	1.52
$6-3-13$	3.98	4.16	4.14	4.18	4.12	4.10	4.16	4.16
$6-4-17$	–	6.94	7.02	6.98	7.0	6.80	7.20	7.30
$6-5-21$	–	13.4	13.28	13.34	13.26	12.92	14.02	13.94

Table 5.4: Performance between the Model and Heuristics for Medium Size Problems

Classes	Model	H1_E	H2_E	RI_E	LPH1_E	LPH2_E	LPH3_E	LPH4_E
$6-2-9$	1.52	1.52	1.52	1.52	1.52	1.52	1.52	1.52
$6-3-13$	**3.98**	**4.06**	**4.06**	4.08	4.12	**4.06**	4.12	4.12
$6-4-17$	–	**6.74**	6.76	6.82	7.0	6.78	7.0	6.98
$6-5-21$	–	12.52	12.52	12.64	13.26	**12.32**	13.34	13.18

' - ': Out of time.

Medium size problems are considered in Tables 5.3 and 5.4. We compared the model with the heuristics for 80% utilisation capacity. H1_E, H2_E and LPH2_E performed better than other heuristics for problem classes $(6-3-13)$. While H1_E and LPH_E have the best performance for problem classes $(6-4-17)$ and $(6-5-21)$ respectively. The model was not able to return results for all the 50 instances for problem classes $(6-4-17)$ and $(6-5-21)$ within 1-hour.

Table 5.5: Performance between the Heuristics for large Size Problems

Classes	H1	H2	RI	LPH1	LPH2	LPH3	LPH4
$6-2-11$	2.72	2.72	2.72	2.72	2.72	2.72	2.72
$6-3-16$	7.16	7.04	7.14	7.10	7.02	7.26	7.10
$6-4-21$	11.88	11.70	11.94	11.72	11.54	12.26	11.96
$6-5-26$	22.80	22.36	23.08	22.60	21.74	23.88	23.56

Table 5.6: Performance between the Heuristics for large Size Problems

Classes	H1_E	H2_E	RI_E	LPH1_E	LPH2_E	LPH3_E	LPH4_E
$6-2-11$	2.72	2.72	2.72	2.72	2.72	2.72	2.72
$6-3-16$	**6.92**	6.96	6.96	7.10	6.94	7.08	7.08
$6-4-21$	11.44	11.28	11.46	11.72	**11.20**	11.72	11.72
$6-5-26$	22.18	21.72	22.42	22.60	**21.06**	23.12	22.90

In Tables 5.5 and 5.6, we compare all heuristics for large size problems defined as bay with 100% capacity. As expected, all the heuristics return the same average reshuffle for problem class $(6-2-11)$ due to the fact that all the 50 instances in the class have the same result. As we increase the problem class by increasing both the number of containers and the number of tiers, we begin to notice a difference in results. All the heuristics return the same results for some of the instances but there are instances where we observe differences depending on the configuration of the bay. HI_E has the best performance for problem class $(6-3-16)$ while LPH2_E has the best performance for problem

classes (6 – 4 – 21) and (6 – 5 – 26).

5.8 Dynamic case

The true test of a reshuffle process is the application to a dynamic situation. In container terminal, reshuffle is done while both retrieval and storage is taking place. 50 instances were generated for four problem classes with 80% utilisation capacity. 1000 containers were retrieved and 650 containers stored for each problem class. The retrieval of containers is the same as static case except that an incoming container will affect the priority order based on the expected departure time of the new container. In view of this, every container in the bay will be re-assigned a new index each time there is an incoming container.

The simulated time and the job type is given in appendix F. The heuristic rule determines where the container will be stored similar to how reshuffles are treated. The RTG is assumed to be positioned close to the sixth column hence, the incoming container is stored from right to left of the bay. Since incoming container is treated as a reshuffle and stored based on the reshuffle rule, it is important to determine where the RTG is positioned especially when we have to resolve the location of a container where there is tie.

Table 5.7: Dynamic

	Average number of reshuffle						Stdev x 10^{-4}			Average CPU time(s)		
Heuristics	(6 – 2 – 9)			(6 – 3 – 13)			(6 – 4 – 17)			(6 – 5 – 21)		
H1	1.68	2.59	0.0272	8.02	2.49	0.0346	13.28	5.13	0.0478	19.26	3.06	0.0446
H2	1.70	1.94	0.0289	8.40	2.05	0.0346	13.84	2.30	0.0401	20.86	3.29	0.0472
RI	1.70	1.70	0.0280	8.22	2.86	0.0369	13.28	2.53	0.0440	18.74	3.80	0.0490
LPH1	**1.66**	2.62	0.0297	7.16	1.70	0.0345	13.10	1.98	0.0415	20.10	3.21	0.0461
LPH2	1.70	1.41	0.0304	**6.88**	1.53	0.0342	**11.76**	2.48	0.0386	**17.18**	3.41	0.0443
LPH3	1.68	2.69	0.0287	8.32	1.62	0.0338	14.14	2.38	0.0414	21.94	3.79	0.0477
LPH4	1.70	1.35	0.0259	8.46	1.53	0.0337	15.38	2.54	0.0436	22.94	3.22	0.0499

The average number of reshuffle for the 50 instances and the processing time is shown in Table 5.7. It can be observed that LPH1 has the best performance for problem class $(6 - 2 - 9)$. Though H1 and LPH3 have the same average number of reshuffle in this class, however H1 has a better performance than LPH3 because it has a lower spread. H2, RI, LPH2 and LPH4 have the same performance in terms of their average number of reshuffle for problem class $(6 - 2 - 9)$, LPH4 outperformed the three other heuristics since it has the lowest spread. LPH2 is the best performing heuristics for other problem classes.

5.9 Compatibility

Here we check how well the different heuristics work together when they are combined to solve a given problem. Since heuristics are known to be instance dependent, it is not surprising to observe some heuristics performing well on some instance and very poorly on some others. In view of this, it is worthwhile investigating how well the heuristics perform individually compare to when they are combined to solve an instance of a problem. We have chosen the

problem class (6 – 5 – 26) since it is the most difficult problem as seen in previous section. 50 instances were generated different from those in Section 5.7 for a static scenario.

Table 5.8: Compatibility

Heuristics	H1	H2	RI	LPH1	LPH2	LPH3	LPH4
H1	**21.98**	21.66	21.73	22.25	21.19	22.42	22.34
H2	21.66	**21.42**	21.48	21.99	20.94	22.18	22.13
RI	21.73	21.48	**21.58**	22.01	21.00	22.28	22.22
LPH1	22.25	21.99	22.01	**22.38**	21.45	22.70	22.57
LPH2	**21.19**	**20.94**	**21.00**	**21.45**	**20.44**	**21.77**	**21.72**
LPH3	22.42	22.18	22.28	22.70	21.77	**22.96**	22.80
LPH4	22.34	22.13	22.22	22.57	21.72	22.80	**22.74**

Table 5.8 show the average reshuffle for 50 instances when the heuristics are used individually as well as when they are combined to solve the problems. Entries along the main diagonal are results for the individual heuristics and other entries indicate pairing of the heuristics. Once again we observe LPH2 returning the best result similar to what we have in Sections 5.7 and 5.8 either when used alone or combined with other heuristics.

5.10 Summary

Container reshuffle is a potent way to measure the efficiency of operations in container ports. Basically, the fewer the reshuffles made to retrieve needed containers, the better since a reshuffle, when unnecessary, is a waste of effort. The problem of minimising the number of reshuffles has therefore, been recognised for some time now and has been approached by many. Integer programming

models of the binary type have been constructed for it and solved both exactly and approximately using heuristics and meta-heuristics. The issue is that these models are less than satisfactory for the following reasons.

1. The models are too big; they have too many unnecessary variables and/or constraints.

2. The heuristics use arbitrary rules to resolve issues of ties to determine storage positions of reshuffled containers.

We have, thus addressed these issues by considering an alternative model to fit our requirements. Alternative model has been given and described extensively. Given the computational complexity of the problem (NP-hard, since it is represented as an ILP), studies relying on exact methods are not realistic. Those using heuristics are more realistic in that sense although some of the heuristics are crude. We have, therefore carried out experiments on realistic instances using the model as well as some of the prominent heuristics found in the literature and used on the reshuffle problem namely H1, H2 and RI. Moreover, we have designed four novel heuristics LPH1, LPH2, LPH3 and LPH4 which has been tested and compared with others on static and dynamic cases. We also proposed a compatibility test to determine how well the different heuristics work together. The results show that LPH2 is superior to other heuristics.

Chapter 6

Conclusions and Future Work

6.1 Summary

Yard management is a core activity in container terminal operations. It concerns the efficient use of the limited space available for temporarily storing containers and the very expensive equipment such as RTGs. The Port of Felixstowe has a basic yard management system which is on the whole ad hoc. It relies mainly on the vast experience of staff. However, it also has basic system referred to as AOC planner. This is a simulation based system which is run to visualise the movements of RTGs and containers within a period of time. This system has limitations such as having to load the tool every 500 seconds before subsequent re-run and only consider RTG movements. Most importantly, it does not consider the unfinished work at each block and the surplus capacity.

Given the importance of yard management to the whole of the port operations,

we thought that a more systematic approach relying on advanced Operation Research (OR) and optimisation approach must be called upon. Our first attempt was to use mathematical modelling to represent the so called Yard Crane Scheduling Problem (YCSP). Before embarking onto that, we looked at what is available in the open literature on port operations. There are two main aspects to this research. The first aspect relate to YCSP.

We examined the model in [72] which aimed at determining the deployment periods for the RTG. The problem was formulated as a mixed integer linear programming problem. The objective is to minimise the unfinished work at the end of the planning period. To have an idea of the tractability of the problem, we solved the model using MATLAB MILP solver version R2017b, `intlinprog` [29]. This small version of the problem highlights the need for a reformulation of the problem. The model assume workload for the entire planning period to be forecast. This is not always the case due to some circumstances such as the delay by external trucks to collect import containers. We observed that the model can be improved to fit real world instance as they arise in the container terminal. An improved model, model1 has been developed by including more constraints to improve the solution quality and comparison made with the old model. Furthermore, we introduced Model2 which aims at minimising jointly the unfinished work and surplus capacity. This model captures the un-utilised time that is lost in operation. Our model return results in faster time.

The use of a rolling horizon in this thesis as in some other sources in the literature is in line with real world operations in terminal yards as work arises

in a dynamic fashion and are thus undertaken in a piece-meal manner. This helps to mitigate uncertainties arising from random workloads in the blocks at different times. We have chosen a shortest deployment period of fifteen minutes in this thesis as is the case with actual operations in PoF where works are assigned within short periods as opposed to long ones which has inherent uncertainties due to the time lag. This makes it possible to investigate actual workload completed compared to predicted workload.

The second aspect relates to the container reshuffle problem as a refinement of the allocation problem. Once the RTGs have been allocated to a bay, the reshuffle problem is defined and solved. We examined the model in [95] and observed that there are some constraints that make the model unnecessarily big. The problem was formulated as binary integer programming problem. The objective is to minimise the number of reshuffles in retrieving a number of containers. An alternative model was introduced and small instances of the problem was solved for an exact solution using GLPSOL in [75]. For medium to large scale problems as they arise in the port, we introduced four new heuristic LPH1, LPH2, LPH3 and LPH4. Implementation of the new heuristics was discussed extensively and comparison was made between them and some other heuristics found in the literature. Both static and dynamic cases of the reshuffle problems were considered. A compatibility test between the heuristics was also proposed. Our heuristics were found to be competitive with those in the literature.

Real-life optimisation problems such as YCSP and Container reshuffle are com-

plex and difficult to solve. They cannot be solved in an exact manner within reasonable amount of time. The alternative therefore is using approximate algorithms to solve this class of problems [94]. These two optimisation solution approaches have been examined and applied in this thesis.

The contributions include:

- A comprehensive literature review on container terminal operations and related problems was provided in chapter 3. Interested researchers in container terminal operational may find this as a reference point.
- The improvements in the scheduling model are useful since better solutions will bring cost reductions and efficient use of resources, while faster methods will allow us to evaluate the decision problems more often. The planning period can be short enough due to the reduction in computational time by our method. This will help to improve the real-time decision making process.
- Our study on the container reshuffle problem should enhance our understanding of RTG operations. The heuristics developed in this thesis serves not only as a reshuffle strategy but also storage strategy. The compatibility check between the different heuristics has never been done in previous studies. This knowledge may further increase the overall efficiency of port operations when comparing different optimizing techniques.

6.2 Future Work

The problems considered in this thesis are based on deterministic cases. In reality however, there are risks in container terminal operations. These issues can include, machine break-down, staffs calling in sick, severe weather, mechanical problems, and strikes. There is also the possibility of incorrect information relating to when containers are needed at the time they are stored, resulting in container-handling problem. Therefore recognising these risks should improve the optimisation process. The resulting stochastic problems are typically very complex, but can yield significant improvements in container terminals efficiency, productivity and the scheduling of terminal operations.

A possible improvement on the YCSP model will be tracking the movement of each RTG. The model determines the number of RTGs to assign to each block and their movements from one block to another but does not identify the specific RTG. This will require introduction of another variable in the model formulation by giving each RTG some form of identification. This would potentially help to measure the relative efficiency or the work rate of each RTG which could help to manage their maintenance.

Developing better solution algorithm could be enhanced by using other methods. An alternative approach may concern combining our heuristics with other solution methods, such as metaheuristics or some exact methods. The hybrid algorithms can draw the strengths of each solution method in solving large-scale problems and identifying good solutions.

A further possibility is to combine both the YCSP and CRP as one. The size of the combined problem could be challenging to solve but it would offer a better optimum solution rather than a sub-optimum one obtained when solving two separate problems.

Bibliography

[1] Alsoufi, G., Yang, X., and Salhi, A. Combined Quay Crane Assignment and Quay Crane Scheduling with Crane Inter-vessel Movement and Non-interference Constraints. Journal of the Operational Research Society 69, 3 (2018), 372–383.

[2] Ayachi, I., Kammarti, R., Ksouri, M., and Borne, P. Harmony Search to Solve the Container Storage Problem with Different Container ypes. International Journal of Computer Applications 975, 8887.

[3] Bai, D., and Tang, L. Open Shop Scheduling Problem to Minimize Makespan with Release Dates. Applied Mathematical Modelling 37, 4 (2013), 2008–2015.

[4] Bai, D., Zhang, Z., and Zhang, Q. Flexible Open Shop Scheduling Problem to Minimize Makespan. Computers & Operations Research 67 (2016), 207–215.

[5] Bar-Yehuda, R., and Even, S. A Linear-time Approximation Algorithm for the Weighted Vertex Cover Problem. Journal of Algorithms 2, 2 (1981), 198–203.

[6] Beck, J. C., and Wilson, N. Proactive Algorithms for Job Shop Scheduling with Probabilistic Durations. Journal of Artificial Intelligence Research 28 (2007), 183–232.

[7] Bierwirth, C., and Meisel, F. A Follow-up Survey of Berth Allocation and Quay Crane Scheduling Problems in Container Terminals. European Journal of Operational Research 244, 3 (2015), 675–689.

[8] Bish, E. K., Leong, T.-Y., Li, C.-L., Ng, J. W., and Simchi-Levi, D. Analysis of a New Vehicle Scheduling and Location Problem. Naval Research Logistics (NRL) 48, 5 (2001), 303–385.

[9] Borgman, B., Van Asperen, E., and Dekker, R. Online Rules for Container Stacking. OR Spectrum 32, 3 (2010), 687–716.

[10] Bortfeldt, A., and Forster, F. A Tree Search Procedure for the Container Pre-marshalling Problem. European Journal of Operational Research 217, 3 (2012), 531–540.

[11] Brinkmann, B. Operations Systems of Container Terminals: A Compendious Overview. In Handbook of Terminal Planning, vol. 49. Springer, 2011, pp. 25–39.

[12] Brucker, P. Scheduling Algorithms. Springer, 2007.

[13] Buhrkal, K., Zuglian, S., Ropke, S., Larsen, J., and Lusby, R. Models for the Discrete Berth Allocation Problem: A Computational Comparison. Transportation Research Part E: Logistics and Transportation Review 47, 4 (2011), 461–473.

[14] Cao, J. X., Lee, D., Chen, J. H., and Shi, Q. The Integrated Yard Truck and Yard Crane Scheduling Problem: Benders' Decomposition-based Methods. Transportation Research Part E: Logistics and Transportation Review 46, 3 (2010), 344–353.

[15] Carlo, H. J., Vis, I. F., and Roodbergen, K. J. Storage Yard Operations in Container Terminals: Literature Overview, Trends, and Research Directions. European Journal of Operational Research 235, 2 (2014), 412–430.

[16] Carlo, H. J., Vis, I. F., and Roodbergen, K. J. Transport Operations in Container Terminals: Literature Overview, Trends, Research Directions and Classification Scheme. European Journal of Operational Research 236, 1 (2014), 1–13.

[17] Caserta, M., Voss, S., and Sniedovich, M. Applying the Corridor Method to a Blocks Relocation Problem. OR Spectrum 33, 4 (2011), 915–929.

[18] Casey, B., and Kozan, E. Optimising Container Storage Processes at Multimodal Terminals. Journal of the Operational Research Society 63, 8 (2012), 1126–1142.

[19] Chang, D., Jiang, Z., Yan, W., and He, J. Developing a Dynamic Rolling-horizon Decision Strategy for Yard Crane Scheduling. Advanced Engineering Informatics 25, 3 (2011), 485–494.

[20] Chen, L., and Langevin, A. Multiple Yard Cranes Scheduling for Loading Operations in a Container Terminal. Engineering Optimization 43, 11 (2011), 1205–1221.

[21] Chen, P., Fu, Z., Lim, A., and Rodrigues, B. Port Yard Storage Optimization. Automation Science and Engineering, IEEE Transactions on 1, 1 (2004), 26–37.

[22] Chen, T. Yard Operations in the Container Terminal: A Study in the Unproductive Moves. Maritime Policy & Management 26, 1 (1999), 27–38.

[23] Cheung, R. K., Li, C., and Lin, W. Interblock Crane Deployment in Container Terminals. Transportation Science 36, 1 (2002), 79–93.

[24] Choe, R., Park, T., Oh, M., Kang, J., and Ryu, K. R. Generating a Rehandling-free Intra-block Remarshaling Plan for An Automated Container Yard. Journal of Intelligent Manufacturing 22, 2 (2011), 201–217.

[25] Chu, C., and Huang, W. Determining Container Terminal Capacity on the Basis of An Adopted Yard Handling System. Transport Reviews 25, 2 (2005), 181–199.

[26] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. Introduction to Algorithms. MIT press, 2009.

[27] Dekker, R., Voogd, P., and Van Asperen, E. Advanced Methods for Container Stacking. OR Spectrum 28, 4 (2006), 563–586.

[28] Dell, R. F., Royset, J. O., and Zyngirdis, I. Optimizing Container Movements Using One and Two Automated Stacking Cranes. Journal of Industrial and Management Optimization 5, 2 (2009), 285–302.

[29] Etter, D. M., and Kuncicky, D. C. Introduction to MATLAB. Prentice Hall, 2011.

[30] Expósito-Izquierdo, C., Melián-Batista, B., and Moreno-Vega, M. Premarshalling Problem: Heuristic Solution Method and Instances Generator. Expert Systems with Applications 39, 9 (2012), 8337–8349.

[31] Festa, P. A Brief Introduction to Exact, Approximation, and Heuristic Algorithms for Solving Hard Combinatorial Optimization Problems. In Transparent Optical Networks (ICTON), 2014 16th International Conference on (2014), IEEE, pp. 1–20.

[32] Forster, F., and Bortfeldt, A. A Tree Search Procedure for the Container Relocation Problem. Computers & Operations Research 39, 2 (2012), 299–309.

[33] Garey, M. R. A Guide to the Theory of NP-Completeness. Computers and Intractability (1979).

[34] Garfinkel, R., and Nemhauser, G. Integer Programming. 1972. John Wiley&Sons, New York.

[35] Gharehgozl, A. H. Developing New Methods for Efficient Container Stacking Operations. TRAIL Thesis Series, T2012/7 (2012).

[36] Gharehgozli, A. H., Roy, D., and De Koster, M. Sea Container Terminals: New Technologies, OR Models, and Emerging Research Areas. ERIM Report Series Reference No. ERS-2014-009-LIS (2014).

[37] Giallombardo, G., Moccia, L., Salani, M., and Vacca, I. Modeling and Solving the Tactical Berth Allocation Problem. Transportation Research Part B: Methodological 44, 2 (2010), 232–245.

[38] Goodwin, G., Seron, M. M., and De Doná, J. A. Constrained Control and Estimation: An Optimisation Approach. Springer Science & Business Media, 2006.

[39] Guo, X., Huang, S. Y., Hsu, W. J., and Low, M. Y. H. Dynamic Yard Crane Dispatching in Container Terminals with Predicted Vehicle Arrival Information. Advanced Engineering Informatics 25, 3 (2011), 472–484.

[40] Haddadi, S. Benders Decomposition for Set Covering Problems. Journal of Combinatorial Optimization 33, 1 (2017), 60–80.

[41] Hakan Akyüz, M., and Lee, C. A Mathematical Formulation and Efficient Heuristics for the Dynamic Container Relocation Problem. Naval Research Logistics (NRL) 61, 2 (2014), 101–118.

[42] He, J., Chang, D., Mi, W., and Yan, W. A Hybrid Parallel Genetic Algorithm for Yard Crane Scheduling. Transportation Research Part E: Logistics and Transportation Review 46, 1 (2010), 136–155.

[43] He, J., Huang, Y., and Yan, W. Yard Crane Scheduling in a Container Terminal for the Trade-off between Efficiency and Energy Consumption. Advanced Engineering Informatics 29, 1 (2015), 59–75.

[44] He, J., Huang, Y., Yan, W., and Wang, S. Integrated Internal Truck, Yard Crane and Quay Crane Scheduling in a Container Terminal Considering Energy Consumption. Expert Systems with Applications 42, 5 (2015), 2464–2487.

[45] Hirashima, Y. A Q-learning System for Container Transfer Scheduling Based on Shipping Order at Container Terminals. International Journal of Innovative Computing, Information and Control 4, 3 (2008), 547–558.

[46] Hirashima, Y. A Q-learning System for Container Marshalling with Group-based Learning Model at Container Yard Terminals. In Proceedings of the International Multi-Conference of Engineers and Computer Scientists 2009 (IMECS 2009) (2009), vol. 1, Citeseer.

[47] Hirashima, Y., Takeda, K., Harada, S., Deng, M., and Inoue, A. A Q-learning for Group-based Plan of Container Transfer Scheduling. JSME International Journal Series C Mechanical Systems, Machine Elements and Manufacturing 49, 2 (2006), 473–479.

[48] Jarosław, P., Czesław, S., and Dominik, Ż. Optimizing Bicriteria Flow Shop Scheduling Problem by Simulated Annealing Algorithm. Procedia Computer Science 18 (2013), 936–945.

[49] Javanshir, H., and Ganji, S. S. Yard Crane Scheduling in Port Container Terminals Using Genetic Algorithm. Journal of Industrial Engineering International 6, 11 (2010), 39–50.

[50] Jin, B., Zhu, W., and Lim, A. Solving the Container Relocation Problem by An Improved Greedy Look-ahead Heuristic. European Journal of Operational Research 240, 3 (2015), 837–847.

[51] Johnson, D. S. Approximation Algorithms for Combinatorial Problems. Journal of Computer and System Sciences 9, 3 (1974), 256–278.

[52] Kang, J., Ryu, K. R., and Kim, K. H. Deriving Stacking Strategies for Export Containers with Uncertain Weight Information. Journal of Intelligent Manufacturing 17, 4 (2006), 399–410.

[53] Kemme, N. Effects of Storage Block Layout and Automated Yard Crane Systems on the Performance of Seaport Container Terminals. OR Spectrum 34, 3 (2012), 563–591.

[54] Kim, K. H., and Bae, J. W. Re-marshaling Export Containers in Port Container Terminals. Computers & Industrial Engineering 35, 3 (1998), 655–658.

[55] Kim, K. H., and Hong, G. A Heuristic Rule for Relocating Blocks. Computers & Operations Research 33, 4 (2006), 940–954.

[56] Kim, K. H., Park, Y., and Jin, M. An Optimal Layout of Container Yards. OR Spectrum 30, 4 (2008), 675–695.

[57] Kim, K. H., Park, Y. M., and Ryu, K. Deriving Decision Rules to Locate Export Containers in Container Yards. European Journal of Operational Research 124, 1 (2000), 89–101.

[58] Korte, B., and Vygen, J. Combinatorial Optimization: Theory and Algorithms, vol. 2. Springer, 2012.

[59] Kumar, M. M., and Omkar, S. Optimization of Yard Crane Scheduling Using Particle Swarm Optimization with Genetic Algorithm Operators (PSOGAO). Journal of Scientific and Industrial Research 67, 5 (2008), 335.

[60] Lee, B. K., and Kim, K. H. Comparison and Evaluation of Various Cycle-time Models for Yard Cranes in Container Terminals. International Journal of Production Economics 126, 2 (2010), 350–360.

[61] Lee, B. K., and Kim, K. H. Optimizing the Block Size in Container Yards. Transportation Research Part E: Logistics and Transportation Review 46, 1 (2010), 120–135.

[62] Lee, B. K., and Kim, K. H. Optimizing the Yard Layout in Container Terminals. OR Spectrum 35, 2 (2013), 363–398.

[63] Lee, D., Jin, J. G., and Chen, J. H. Terminal and Yard Allocation Problem for a Container Transshipment Hub with Multiple Terminals. Transportation Research Part E: Logistics and Transportation Review 48, 2 (2012), 516–528.

[64] Lee, Y., and Chao, S. A Neighborhood Search Heuristic for Pre-marshalling Export Containers. European Journal of Operational Research 196, 2 (2009), 468–475.

[65] Lee, Y., and Hsu, N. An Optimization Model for the Container Pre-marshalling Problem. Computers & Operations Research 34, 11 (2007), 3295–3313.

[66] Lee, Y., and Lee, Y. A Heuristic for Retrieving Containers from a Yard. Computers & Operations Research 37, 6 (2010), 1139–1147.

[67] Legato, P., Trunfio, R., and Meisel, F. Modeling and Solving Rich Quay Crane Scheduling Problems. Computers & Operations Research 39, 9 (2012), 2063–2078.

[68] Lenstra, J. Job Shop Scheduling. In Combinatorial Optimization. Springer, 1992, pp. 199–207.

[69] Li, J., Leung, S. C., Wu, Y., and Liu, K. Allocation of Empty Containers between Multi-ports. European Journal of Operational Research 182, 1 (2007), 400–412.

[70] Li, W., Goh, M., Wu, Y., Petering, M., De Souza, R., and Wu, Y. A Continuous Time Model for Multiple Yard Crane Scheduling with Last Minute Job Arrivals. International Journal of Production Economics 136, 2 (2012), 332–343.

[71] Lim, A., and Xu, Z. A Critical-Shaking Neighborhood Search for the Yard Allocation Problem. European Journal of Operational Research 174, 2 (2006), 1247–1259.

[72] Linn, R. J., and Zhang, C. A Heuristic for Dynamic Yard Crane Deployment in a Container Terminal. IIE transactions 35, 2 (2003), 161–174.

[73] Liu, C., Jula, H., Vukadinovic, K., and Ioannou, P. Automated Guided Vehicle System for two Container Yard Layouts. Transportation Research Part C: Emerging Technologies 12, 5 (2004), 349–368.

[74] Mak, K., and Sun, D. Scheduling Yard Cranes in a Container Terminal Using a New Genetic Approach. Engineering Letters 17, 4 (2009), 274.

[75] Makhorin, A. The GNU Linear Programming Kit (GLPK). GNU Software Foundation, 2000, 2015.

[76] McCarl, B., and Spreen, T. Applied mathematical programming using algebraic system, department of agricultural economics, texas a&m university, 2002.

[77] McNary, B. S. Real-time Dispatching of Rubber Tired Gantry Cranes in Container Terminals. PhD thesis, Monterrey California. Naval Postgraduate School, 2008.

[78] Meidanis, J., Porto, O., and Telles, G. P. On the Consecutive Ones Property. Discrete Applied Mathematics 88, 1-3 (1998), 325–354.

[79] Mitchell, J. E. Branch-and-Cut Algorithms for Combinatorial Optimization Problems. Handbook of Applied Optimization (2002), 65–77.

[80] Monaco, M. F., and Sammarra, M. The Berth Allocation Problem: A Strong Formulation Solved by a Lagrangean Approach. Transportation Science 41, 2 (2007), 265–280.

[81] Murty, K. G. Yard Crane Pools and Optimum Layouts for Storage Yards of Container Terminals. Journal of Industrial and Systems Engineering 1, 3 (2007), 190–199.

[82] Ng, W. Crane Scheduling in Container Yards with Inter-crane Interference. European Journal of Operational Research 164, 1 (2005), 64–78.

[83] Ng, W., and Mak, K. Yard Crane Scheduling in Port Container Terminals. Applied Mathematical Modelling 29, 3 (2005), 263–276.

[84] Papadimitriou, C. H., and Steiglitz, K. Combinatorial Optimization: Algorithms and Complexity. Courier Corporation, 1998.

[85] Petering, M. E. Effect of Block Width and Storage Yard Layout on Marine Container Terminal Performance. Transportation Research Part E: Logistics and Transportation Review 45, 4 (2009), 591–610.

[86] Ren, T., Guo, M., Lin, L., and Miao, Y. A Local Search Algorithm for the Flow Shop Scheduling Problem with Release Dates. Discrete Dynamics in Nature and Society 2015 (2015).

[87] Ruf, N., and SchöBel, A. Set Covering with Almost Consecutive Ones Property. Discrete Optimization 1, 2 (2004), 215–228.

[88] Saanen, Y. Optimizing Automated Container Terminals to Boost Productivity. Port Technology International 51 (2011), 55–65.

[89] Saanen, Y. A., and Valkengoed, M. V. Comparison of Three Automated Stacking Alternatives by Means of Simulation. In Proceedings of the 37th conference on Winter simulation (2005), Winter Simulation Conference, pp. 1567–1576.

[90] Salido, M. A., Sapena, O., Rodriguez, M., and Barber, F. A Planning Tool for Minimizing Reshuffles in Container Terminals. In 2009 21st IEEE International Conference on Tools with Artificial Intelligence (2009), IEEE, pp. 567–571.

[91] Schrijver, A. Theory of Linear and Integer Programming. John Wiley & Sons, 1998.

[92] Sharif, O. Application of Agent-based Approaches to Enhance Container Terminal Operations. PhD thesis, University of South Carolina, 2013.

[93] Stahlbock, R., and Voβ, S. Vehicle Routing Problems and Container Terminal Operations: An Update of Research. In The Vehicle Routing Problem: Latest Advances and New Challenges. Springer, 2008, pp. 551–589.

[94] Talbi, E. Metaheuristics: from Design to Implementation. John Wiley & Sons, 2009.

[95] Tang, L., Jiang, W., Liu, J., and Dong, Y. Research into Container Reshuffling and Stacking Problems in Container Terminal Yards. IIE Transactions 47 (2015), 751–766.

[96] Türkoğullari, Y. B., Taşkin, Z. C., Aras, N., and Altinel, İ. K. Optimal Berth Allocation, Time-variant Quay Crane Assignment and Schedul-

ing with Crane Setups in Container Terminals. European Journal of Operational Research 254, 3 (2016), 985–1001.

[97] Umang, N., Bierlaire, M., and Vacca, I. Exact and Heuristic Methods to Solve the Berth Allocation Problem in Bulk Ports. Transportation Research Part E: Logistics and Transportation Review 54 (2013), 14–31.

[98] Ursavas, E. Crane Allocation with Stability Considerations. Maritime Economics & Logistics 19, 2 (2017), 379–401.

[99] Van Asperen, E., Borgman, B., and Dekker, R. Evaluating Impact of Truck Announcements on Container Stacking Efficiency. Flexible Services and Manufacturing Journal 25, 4 (2013), 543–556.

[100] Van Dijk, S. Decomposition Methods and Rolling Horizon Approach for the Yard Crane Scheduling Problem. Master's thesis, Dleft University of Technology, Delft, The Netherlands, May 2015.

[101] Vis, I. F. A Comparative Analysis of Storage and Retrieval Equipment at a Container Terminal. International Journal of Production Economics 103, 2 (2006), 680–693.

[102] Wan, Y., Liu, J., and Tsai, P. The Assignment of Storage Locations to Containers for a Container Stack. Naval Research Logistics (NRL) 56, 8 (2009), 699–713.

[103] Weiss, C., Waldherr, S., Knust, S., and Shakhlevich, N. V. Open Shop Scheduling with Synchronization. Journal of Scheduling 20, 6 (2017), 557–581.

[104] Wiese, J., Kliewer, N., and Suhl, L. A Survey of Container Terminal Characteristics and Equipment Types. Working Paper 0901, March 2009. Decision Support & Operations Research Lab. University of Paderborn.

[105] Williams, H. P. Model Building in Mathematical Programming. John Wiley & Sons, 2013.

[106] Williamson, D. P., and Shmoys, D. B. The Design of Approximation Algorithms. Cambridge University Press, 2011.

[107] Wolde, M. G., Berhan, E., and Jilcha, K. Production Improvement with Flow Shop Scheduling Heuristics in Household Utensils Manufacturing Company. Cogent Engineering 5, 1 (2018), 1430007.

[108] Wolsey, L. A. Integer programming. Wiley, 1998.

[109] Yang, J. H., and Kim, K. H. A Grouped Storage Method for Minimizing Relocations in Block Stacking Systems. Journal of Intelligent Manufacturing 17, 4 (2006), 453–463.

[110] Yang, X. Introduction to Mathematical Optimization: From Linear Programming to Metaheuristics. Cambridge International Science Publishing, 2008.

[111] Yang Peng, Sun Junqing, W. Y. A Novel Genetic Algorithm for Multiple Yard Cranes Scheduling. Journal of Computational Information Systems 9, 14 (2013), 5761–5769.

[112] Yi, D., GuoLong, L., and ChengJi, L. Real Time RTG Dispatching Model and Algorithm for Container Terminal with Safe Distance. Journal of Convergence Information Technology 7, 12 (2012), 94–102.

[113] Zhang, C., Wan, Y., Liu, J., and Linn, R. J. Dynamic Crane Deployment in Container Storage Yards. Transportation Research Part B: Methodological 36, 6 (2002), 537–555.

[114] Zhang, Y., Huang, Y., Yan, W., and He, J. A Proactive Approach for Yard Crane Scheduling Problem with Stochastic Arrival and Handling

Time. International Journal of Hybrid Information Technology 9, 2 (2016), 389–406.

[115] Zhao, J., and Tang, L. The Simultaneous Reshuffle and Yard Crane Scheduling Problem in Container Terminals. In International Forum on Shipping, Ports and Airports (IFSPA) 2012: Transport Logistics for Sustainable Growth at a New Level Hong Kong Polytechnic University (2012).

[116] Zhao, W., and Goodchild, A. V. The Impact of Truck Arrival Information on Container Terminal Rehandling. Transportation Research Part E: Logistics and Transportation Review 46, 3 (2010), 327–343.

[117] Zheng, H., Yu, K., and Tu, C. Multiple Yard Cranes Scheduling Model and Algorithm in the Mixture Storage Block. Journal of Software 8, 10 (2013), 2495–2502.

Appendix A

Model solution of L&Z using intlinprog solver

To get an exact solution for a small instance of the problem we used MATLAB [29]. It will be solved using MATLAB MILP solver version R2018b, `intlinprog`.

A.1 intlinprog driver code

```
n = 2; % number of YCs
B = [18.75  3.75  3.50  14.50]'; % workload per block in each deployment
d3 = [15 10 10 15 15 10 10 15]; % net crane capacity per block
function  Output=H_function_1(n,d)
n=2;
r=n^2; c=n^3;
h=zeros(r, c);
d1=-ones (1, n^2*(n-1));
k1=1; x=1;
for    i=n+1:r
```

```
k=x*n;
h (i,k1:k)=d1(k1:k);
k1=k1+n;
x=x+1;
end
for    i=1:(c/n^2)
I= (i-1)*n^2+1;
F=I + (n^2-1);
II= (i-1)*n+1;
FF=II+ (n-1);
h(II:FF,I:F)=repmat (eye(n),1,n);
end
Output=h;
function Output=H_function_2(n, d)
n=2;
r=n^2; c=n^3;
h=zeros(r, c);
d= [ones (1, n^3)];
k1=1; x=1;
for   i=1:r
k=x*n;
h (i,k1:k)=d(k1:k);
k1=k1+n;
x=x+1;
end
Output=h;
function Output=H_function_3(n, d)
n=2;
r=n^2; c=n^3;
```

```
h=zeros(r, c);
d= [15 10 10 15 15 10 10 15];
k1=1; x=1;
for  i=1:r
k=x*n;
h (i,k1:k)=d(k1:k);
k1=k1+n;
x=x+1;
end
Output=h;
d1 = -ones (1, n^2*(n-1));
d2 = ones (1, n^3);
h1 = H_function_1 (n, d1);
h2 = H_function_2 (n, d2);
h3 = H_function_3 (n, d3);
u = ones (n^2, 1);
p = spdiags ([-u  u -u], [-n 0 n^2], n^2, 2*n^2);
A = [zeros (n^2, 2*n^2) h2; zeros (n^2*(n+1), n^2*(n+2))];
Aeq = [zeros (n^2, 2*n^2) h1; p h3; zeros (n^3, n^2*(n+2))];
intcon = (2*n^2)+1:n^2*(n+2);
beq = [ones(n,1);zeros(n*(n-1),1); B ;zeros(n^3,1)];
f = [ones (1, n^2) zeros (1,n^2*(n+1))];
lb = zeros(n^2*(n+2),1);
b = [2*u; zeros (n^2*(n+1), 1)];
[x,fval] = intlinprog (f, intcon, A, b, Aeq, beq, lb, [ ])
```

A.2 The output

```
f =   1   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0

A =   0   0   0   0   0   0   0   0   1   1   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   1   1   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0   1   1   0   0
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   1
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

Aeq = 0   0   0   0   0   0   0   0   1   0   1   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   1   0   1   0   0   0   0
      0   0   0   0   0   0   0   0  -1  -1   0   0   1   0   1   0
      0   0   0   0   0   0   0   0   0   0  -1  -1   0   1   0   1
      1   0   0   0  -1   0   0   0  15  10   0   0   0   0   0   0
      0   1   0   0   0  -1   0   0   0   0  10  15   0   0   0   0
     -1   0   1   0   0   0  -1   0   0   0   0   0  15  10   0   0
      0  -1   0   1   0   0   0  -1   0   0   0   0   0   0  10  15
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

```
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

b' =  [2  2  2  2  0  0  0  0  0  0  0  0  0  0  0]

beq' = [1  1  0  0  18.75  3.75  3.5  14.5  0  0  0  0  0  0  0  0]

lb' = [0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]

[x,fval]= intlinprog (f, 9:16, A, b, Aeq, beq, lb, [])
LP:                Optimal objective value is 0.000000.

Cut Generation:    Applied 1 flow cover cut, and 5 mir cuts.
Lower bound is 3.750000.
Relative gap is 0.00%.

Optimal solution found.

Intlinprog stopped at the root node because the
objective value is within a gap tolerance of the optimal value,
options.TolGapAbs = 0 (the default value). The intcon variables are
integer within tolerance, options.TolInteger = 1e-05 (the default value).
x =   3.75, 0, 0, 0, 0, 11.25, 7.75, 0.5, 1, 0, 0, 1, 1, 0, 0, 1
fval = 3.75
```

Appendix B

GMPL for YCSP model 1

The model formulated in GNU MathProg modelling Language (GMPL) before been submitted to the solver is as follows;

```
param period_count >0;
param rtg_capacity > 0 ;
set BLOCKS;
set PERIODS := 1.. period_count;
param demand { b in BLOCKS, p in PERIODS};
param travel {bf in BLOCKS, bt in BLOCKS};
var rtg_movement {bf in BLOCKS, bt in BLOCKS, p in 0..period_count}
integer >=0;
var remaining_work {b in BLOCKS, 0  .. period_count}  >=0;
var surplus_capacity{b in BLOCKS, p in PERIODS} integer >=0;
minimize goal:
sum{b in BLOCKS, p in PERIODS}(remaining_work[b,p]);
subject to flow_conservation {b in BLOCKS, p in 1 .. period_count}:
sum {b2 in BLOCKS} rtg_movement[b,b2,p] =
sum {b2 in BLOCKS} rtg_movement[b2,b,p-1] ;
```

```
subject to block_max_rtg {bt in BLOCKS, p in PERIODS} :
sum{bf in BLOCKS} rtg_movement[bf,bt,p] <= 2;
subject to remaining_work_balance{b in BLOCKS, p in 1 .. period_count} :
remaining_work[b,p-1] + demand[b,p]
- sum{b2 in BLOCKS} ((rtg_capacity - travel[b2,b]) *rtg_movement[b2,b,p])
+surplus_capacity[b,p]
-remaining_work[b,p] = 0;
subject to total_rtg_period{p in 1 .. period_count}:
sum{{b in BLOCKS, b2 in BLOCKS} rtg_movement[b2,b,p] =
sum{b in BLOCKS} rtg_movement[b,b,0];
subject to can_reach_block {b in BLOCKS, b2 in BLOCKS, p in PERIODS} :
rtg_movement[b,b2,p] * (rtg_capacity - travel[b,b2]) >=0;
subject to no_movements_first_period{ b in BLOCKS, b2 in BLOCKS} :
if b<>b2 then rtg_movement[b,b2,0] = 0 ;
subject to rtg_starting_positions {b in BLOCKS}:
rtg_movement[b,b,0] = 1 ;
subject to c8 {b in BLOCKS} :
remaining_work[b,0]  = 0;
subject to c8_2 {b in BLOCKS, p in 1.. period_count}:
surplus_capacity [b,p] >= 0;
subject to c8_3 {b in BLOCKS, p in 1.. period_count}:
remaining_work[b,p] >=0;
subject to max_movements_in_period {p in 1.. period_count}:
max_movements[p] >= sum {b in BLOCKS, b2 in BLOCKS} if b<>b2
then rtg_movement [b,b2,p] ;
```

Appendix C

Yard with ten blocks

C.1 Output of the MILP Solver of GLPSOL

```
GLPSOL: GLPK LP/MIP Solver, v4.57
Parameter(s) specified in the command line:
--cover --clique --gomory --mir -m L&Z102.mod
Reading model section from L&Z102.mod...
Reading data section from L&Z102.mod...
339 lines were read
Generating goal...
Generating flow_conservation...
Generating block_max_rtg...
Generating remaining_work_balance...
Generating total_rtg_period...
Generating can_reach_block...
Generating no_movements_first_period...
Generating rtg_starting_positions...
Generating c8...
```

```
Generating c8_2...
Generating c8_3...
Model has been successfully generated
GLPK Integer Optimizer, v4.57
761 rows, 590 columns, 2974 non-zeros
500 integer variables, none of which are binary
Preprocessing...
157 rows, 360 columns, 1760 non-zeros
320 integer variables, 80 of which are binary
Scaling...
A: min|aij| =  1.000e+00  max|aij| =  1.500e+01  ratio =  1.500e+01
GM: min|aij| =  7.577e-01  max|aij| =  1.320e+00  ratio =  1.742e+00
EQ: min|aij| =  5.753e-01  max|aij| =  1.000e+00  ratio =  1.738e+00
2N: min|aij| =  5.000e-01  max|aij| =  1.250e+00  ratio =  2.500e+00
Constructing initial basis...
Size of triangular part is 156
Solving LP relaxation...
GLPK Simplex Optimizer, v4.57
157 rows, 360 columns, 1760 non-zeros
0: obj =   1.145000000e+02 inf =   6.091e+01 (22)
72: obj =   2.127916667e+02 inf =   9.548e-15 (0)
*   144: obj =   1.193750000e+01 inf =   0.000e+00 (0) 1
OPTIMAL LP SOLUTION FOUND
Integer optimization begins...
Gomory's cuts enabled
MIR cuts enabled
Cover cuts enabled
Clique cuts enabled
Constructing conflict graph...
```

```
Conflict graph has 80 + 0 = 80 vertices
+   144: mip =     not found yet >=              -inf        (1; 0)
Cuts on level 0: gmi = 16; mir = 23;
Cuts on level 52: gmi = 16; mir = 23;
+  1088: >>>>>   8.925000000e+01 >=   3.351889897e+01  62.4% (53; 0)
Cuts on level 8: gmi = 22; mir = 36;
+  1387: >>>>>   3.650000000e+01 >=   3.573432056e+01   2.1% (59; 1)
+  1561: mip =   3.650000000e+01 >=     tree is empty   0.0% (0; 121)
INTEGER OPTIMAL SOLUTION FOUND
Time used:   0.2 secs
Memory used: 2.3 Mb (2390967 bytes)
SOLVE COMPLETE
Remaining work rollup = 36.5
Remaining work last period = 7.5
Start positions:
A4(1)  A5(1)  B4(1)  B5(1)  C4(1)  C5(1)  D4(1)  D5(1)  E4(1)  E5(1)
1 C4>E4 : 1 D4>E5 : 1 E4>C5 : 1
2 A4>A5 : 1 C5>B5 : 1 D5>C5 : 1 E5>D5 : 1 E5>E4 : 1
3 A5>B4 : 2 B4>A4 : 1 B5>C4 : 2 C5>D4 : 1 C5>D5 : 1 E4>E5 : 2
4   B4>A4 : 1 C4>B5 : 1 D5>C4 : 1 D5>E4 : 1 E5>E4 : 1


       A4(1) A5(1) B4(1) B5(1) C4(0) C5(1) D4(1) D5(0) E4(1) E5(1)
1         1     1     1     1           2           1       1    2
2               2     1     2           2           1       2
3         1           2           2             1   2             2
4         2           1     1     2             1           2    1
Model has been successfully processed
```

Appendix D

GMPL formulation for Container Reshuffle model

The model formulated in GNU MathProg modelling Language (GMPL) before been submitted to the solver is as follows;

```
param container_count integer > 0;
param tier_count integer > 0;
param column_count  integer > 0 ;
set TIERS:= 1..tier_count;
set COLUMNS:= 1..column_count;
set CONTAINERS := 1..container_count;
param physical_relationship {i in CONTAINERS, s in CONTAINERS, c in COLUMNS,
t in TIERS};
param container_float {s in CONTAINERS, c in COLUMNS, t in TIERS} ;
param reshuffle_not_in_same_column {i in CONTAINERS, s in CONTAINERS,
c in COLUMNS};
param bay_configuration {i in CONTAINERS,c in COLUMNS,t in TIERS};
param container_not_reshuffle_retain_position {i in CONTAINERS,
```

```
s in CONTAINERS, c in COLUMNS, t in TIERS};
param c_start_pos {i in CONTAINERS, c in COLUMNS, t in TIERS};
var container_reshuffle {i in 1..container_count, s in 1..container_count}
binary ;
var container_relative_height {i in CONTAINERS, i2 in CONTAINERS,
s in CONTAINERS} binary ;
var container_location {i in CONTAINERS, s in CONTAINERS, c in COLUMNS,
t in TIERS} binary ;
minimize goal:
sum{s in 1..container_count - 1, i in 2..container_count}
(container_reshuffle[s,i]);
subject to c1 {i in CONTAINERS, s in CONTAINERS, c in COLUMNS}:
(1 - sum{t in TIERS} container_location[s,s,c,t])*tier_count
+ container_reshuffle[s,i]
>= (sum{t in TIERS}(t*container_location[s,i,c,t]) - sum{t in TIERS}
(t*container_location[s,s,c,t]))/tier_count;
subject to c1_2 {i in CONTAINERS, s in CONTAINERS, c in COLUMNS}:
(sum{t in TIERS}(t*container_location[s,s,c,t]) - sum{t in TIERS}
(t*container_location[s,i,c,t]))/tier_count
<= 1 - container_reshuffle[s,i];
subject to container_can_only_be_in_a_slot {i in CONTAINERS,
s in CONTAINERS: s<=i}:
sum{c in COLUMNS, t in TIERS} container_location[s,i,c,t]=1;
subject to max_one_container_per_slot {s in CONTAINERS, c in COLUMNS,
t in TIERS}:
sum{i in CONTAINERS}container_location[s,i,c,t]<=1;
subject to container_not_floating{s in CONTAINERS, c in COLUMNS,
t in TIERS: t>=2}:
sum{i in CONTAINERS}container_location[s,i,c,t]<= sum{i in CONTAINERS}
```

```
container_location[s,i,c,t-1];
subject to no_reshuffle_in_same_column{i in CONTAINERS, s in CONTAINERS,
c in COLUMNS: s<i}:
sum{t in TIERS} container_location[s+1,i,c,t] <=2 - container_reshuffle[s,i]
- sum{t in TIERS} container_location[s,s,c,t];
subject to c6 {i in CONTAINERS, s in CONTAINERS, i2 in CONTAINERS: i<>i2}:
2 - container_reshuffle[s,i] - container_reshuffle[s,i2]
+ container_relative_height[s,i,i2]>=(sum{c in COLUMNS, t in TIERS}
(t*container_location[s,i2,c,t]) - sum{c in COLUMNS, t in TIERS}
(t*container_location[s,i,c,t]))/tier_count;
subject to c6_1 {i in CONTAINERS, s in CONTAINERS, i2 in CONTAINERS: i<>i2}:
container_reshuffle[s,i] + container_reshuffle[s,i2]
+ container_relative_height[s,i,i2]<=3 + (sum{c in COLUMNS, t in TIERS}
(t*container_location[s,i2,c,t]) - sum{c in COLUMNS, t in TIERS}
(t*container_location[s,i,c,t]))/tier_count;
subject to c6_2 {i in CONTAINERS, s in CONTAINERS, i2 in CONTAINERS: i<>i2}:
container_relative_height[s,i,i2]<= container_reshuffle[s,i];
subject to c6_3 {i in CONTAINERS, s in CONTAINERS, i2 in CONTAINERS: i<>i2}:
container_relative_height[s,i,i2]<= container_reshuffle[s,i2];
subject to c7 {i in CONTAINERS, s in CONTAINERS, i2 in CONTAINERS,
c in COLUMNS: i<>i2 and s <i}:
sum{t in TIERS}(t*container_location[s+1,i,c,t]) - sum{t in TIERS}
(t*container_location[s+1,i2,c,t])>= -1*tier_count*(1
- container_relative_height[s,i,i2]) - tier_count*
(1-container_reshuffle[s,i])- tier_count*(1-container_reshuffle[s,i2])
-tier_count*(1-sum{t in TIERS}container_location[s+1,i,c,t]);
subject to c8 {i in CONTAINERS, s in CONTAINERS, c in COLUMNS,
t in TIERS: s < i}:container_location[s+1,i,c,t] - container_location[s,i,c,t]
>= -1*container_reshuffle[s,i];
```

```
subject to c8_1 {i in CONTAINERS, s in CONTAINERS, c in COLUMNS,
t in TIERS: s <i}:container_location[s,i,c,t] - container_location[s+1,i,c,t]
>= -1*container_reshuffle[s,i];
subject to c9 {i in CONTAINERS, c in COLUMNS, t in TIERS}:
container_location[1,i,c,t] = bay_configuration[i,c,t];
subject to c10 {i in 1..container_count, s in 1..container_count: s<i}:
container_reshuffle[s,i] >=0;
subject to c10_1 {i in 1..container_count, i2 in 1..container_count,
s in 1..container_count: i<>i2}:container_relative_height[s,i,i2] >=0;
subject to c10_2 {i in 1..container_count, s in 1..container_count,
c in COLUMNS, t in 1..tier_count}:container_location[s,i,c,t] >=0;
```

Appendix E

An instance of the problem for Container Reshuffle

E.1 Input of GLPSOL: the MILP Solver

```
param container_count:= 7;
param tier_count:= 3;
param column_count:= 3;
param bay_configuration:=
[1,*,*] : 3 2 1 :=
1 0 0 0
2 0 0 0
3 0 0 1
[2,*,*] : 3 2 1 :=
1 0 0 0
2 0 1 0
3 0 0 0
[3,*,*] : 3 2 1 :=
```

```
1 0 1 0

2 0 0 0

3 0 0 0

[4,*,*] : 3 2 1:=

1 0 0 0

2 0 0 1

3 0 0 0

[5,*,*] : 3 2 1:=

1 0 0 1

2 0 0 0

3 0 0 0

[6,*,*] : 3 2 1:=

1 1 0 0

2 0 0 0

3 0 0 0

[7,*,*] : 3 2 1:=

1 0 0 0

2 0 0 0

3 0 1 0
```

In this problem, there are seven containers with three columns and three tiers. The initial position of the containers are shown in the matrix param bay configuration. $[i, c, t]$ means container i is in column c and tier t at the initial stage. For example, container 1 is in column 3 and tier 1 in the problem input above.

E.2 Output of GLPSOL: the MILP Solver

```
GLPSOL: GLPK LP/MIP Solver, v4.57
```

```
Parameter(s) specified in the command line:
--cover --clique --gomory --mir -m Reshuffle_Alb.mod
Reading model section from Reshuffle_Alb.mod...
Reading data section from Reshuffle_Alb.mod...
1091 lines were read
Generating goal...
Generating c1...
Generating c1_2...
Generating container_can_only_be_in_a_slot...
Generating max_one_container_per_slot...
Generating container_not_floating...
Generating no_reshuffle_in_same_column...
Generating c6...
Generating c6_1...
Generating c6_2...
Generating c6_3...
Generating c7...
Generating c8...
Generating c8_1...
Generating c9...
Generating c10...
Generating c10_1...
Generating c10_2...
Model has been successfully generated
GLPK Integer Optimizer, v4.57
3242 rows, 784 columns, 22128 non-zeros
784 integer variables, all of which are binary
Preprocessing...
43 hidden packing inequaliti(es) were detected
```

```
39 hidden covering inequaliti(es) were detected
22 constraint coefficient(s) were reduced
911 rows, 376 columns, 7696 non-zeros
376 integer variables, all of which are binary
Scaling...
A: min|aij| =  3.333e-01  max|aij| =  3.000e+00  ratio =  9.000e+00
Problem data seem to be well scaled
Constructing initial basis...
Size of triangular part is 911
Solving LP relaxation...
GLPK Simplex Optimizer, v4.57
911 rows, 376 columns, 7696 non-zeros
0: obj =   3.000000000e+00 inf =   1.100e+01 (8)
37: obj =   3.050000000e+00 inf =   0.000e+00 (0)
*    56: obj =   3.000000000e+00 inf =   2.064e-15 (0)
OPTIMAL LP SOLUTION FOUND
Integer optimization begins...
Gomory's cuts enabled
MIR cuts enabled
Cover cuts enabled
Clique cuts enabled
Constructing conflict graph...
Conflict graph has 353 + 45 = 398 vertices
+    56: mip =     not found yet >=              -inf        (1; 0)
Solution found by heuristic: 3
+    84: mip =   3.000000000e+00 >=     tree is empty   0.0% (0; 1)
INTEGER OPTIMAL SOLUTION FOUND
Time used:   0.0 secs
Memory used: 3.9 Mb (4083984 bytes)
```

```
SOLVE COMPLETE

Number of reshuffles = 3

Model has been successfully processed
```

Appendix F

Dynamic case

F.1 Simulated time

Table F.1: Simulated time and job type

Job sequence	Simulated Time (h:m)	Job Type
1	00.00	Retrieval
2	00.03	Retrieval
3	00.07	Storage
4	00.10	Retrieval
5	00.12	Retrieval
6	00.14	Storage
7	00.17	Retrieval
8	00.21	Retrieval
9	00.24	Retrieval
10	00.28	Storage
11	00.31	Storage
12	00.34	Retrieval
13	00.37	Storage

14	00.41	Retrieval
15	00.45	Retrieval
16	00.48	Storage
17	00.52	Retrieval
18	00.55	Storage
19	00.58	Retrieval
20	00.62	Retrieval
21	01.06	Storage
22	01.08	Storage
23	01.11	Retrieval
24	01.14	Storage
25	01.19	Retrieval
26	01.22	Retrieval
27	01.26	Storage
28	01.29	Retrieval
29	01.33	Storage
30	01.37	Retrieval
31	01.40	Retrieval
32	01.44	Storage
33	01.47	Retrieval

Appendix G

List of works

G.1 Paper Published

1. Reshuffle minimization to improve storage yard operations efficiency. Journal of Algorithm and Computational Technology. Manuscript number: ACT-19-0105 [In press].

G.2 Papers submitted to Journals

1. Improving Yard Management in Container Ports using Mathematical Modelling and an Optimisation Approach. Submitted On 8th Nov 2016 to Journal of the Operational Research Society (JORS) with manuscript number: JORS-16-0671-P.

2. Improving Yard Management in Container Ports using Mathematical Modelling and an Optimisation Approach. Submitted on 18th July 2017 to the International Journal of Shipping and Transport Logistics (IJSTL) with manuscript number Ref: IJSTL 186402.

3. Yard crane scheduling in container ports using mathematical modelling and an optimization approach. Submitted on 16th August 2017 to RAIRO Operations Research with manuscript number: ro170320.

4. Yard crane scheduling in container ports using mathematical modelling and an optimization approach. Resubmitted on 5th June 2018 to RAIRO Operations Research with manuscript number: ro170320R1.

5. Reshuffle minimization to improve storage yard operations efficiency. Submitted on 3rd August 2018 to Annals of Operations Research (ANOR) with manuscript number: ANOR-D-18-00831.

G.3 Papers presented at Workshops

1. Improving Yard Management in Container Ports using Mathematical Modelling and Systematic Optimisation Approach. EGL Workshop on Numerical Mathematics, Room 310 Roberts Building, University College London, 29th May 2016.

2. Improving Yard Management in Container Ports using Mathematical Modelling and an Optimisation Approach. EGL Applied and Numerical Mathematics Workshop, KW003 King William Court, Maritime Greenwich Campus, University of Greenwich, 8th-9th June 2017.

3. On a Heuristic to Solve the Reshuffle Problem in the Storage Yard. Meeting to catch up with our partners from the Port of Felixstowe. Room 6.314 Department of Mathematical Sciences, University of Essex. 23rd february, 2018.