Optimisation of Container Process
at
Multimodal Container Terminals

Andy King-sing Wong
MInfoTech, GradDipCompSc (Queensland University of Technology)
BSc, CertEd (University of Hong Kong)

A thesis submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy
April 2008

Principal Supervisor: Professor Erhan Kozan
Associate Supervisor: Professor Luis Ferreira

School of Mathematical Sciences
Faculty of Science
Queensland University of Technology
Brisbane, Queensland 4001, Australia
Keywords

Container process

Scheduling

Storage allocation

Ship delays

Meta-heuristics
Abstract

Multimodal container terminals are an important part of the logistics systems in international trade. Any improvement in the terminal efficiency is likely to reduce the costs of transporting goods, and to strengthen the trading position of the nation. During the import process, containers flow from ships to the storage yard for temporary storage and then are later moved to the hinterland by rail, or by road. The export process is the reverse of the import process. From the marshalling area, it is possible for a yard machine to carry an inbound container to the storage area and back with an inbound container in one round trip. This thesis investigates the inbound and outbound container process of multimodal container terminals in a multi-ship and multi-berth environment. The aim is to develop mathematical models and analytical tools for yard operation and planning. This study concerns the yard-layout, storage locations, operation strategies as well as the sequencing and scheduling of container process. Several models are developed for the scheduling of container process, taking account of planned and unplanned disruptions, and the intermediate buffer at the marshalling area. The problem is NP-hard and real-life problems often involve large number of containers. In addition, many schedules may not be feasible due to deadlock or violation of precedence-constraints. Good results were achieved on benchmark problems using the proposed innovative. In dealing with unplanned disruptions, reactive scheduling approach was found to give the results similar to as if the disruptions were planned in advance. Numerical investigations are also presented on various factors affecting the efficiency of seaport container terminals including the number of yard machines, and the number of quay crane. As with the various yard-layouts studied, it was found that containers are best stored in rows perpendicular to the quay-line with about 10 to 14 bays in each row.

For a shorter ship service time, ideally the containers should be stored as close as possible to the ship. The best storage locations, however, are scarce resources and are not always available. Another model is developed for the best storage location as well as the best schedule for the container process. From an initial best schedule
with predefined storage locations, the problem is solved by iterating through the refinement of storage scheme and re-scheduling.

At a seaport terminal, ships are planned to arrive and leave within a scheduled time window. Nevertheless, a ship may arrive late due to poor weather conditions or disruptions at the previous port. Such delay may also affect its departure to the subsequent port. To minimise the impact of ship delays, port operators must consider alternate arrangements including re-assignment of berths, re-sequencing of ships and rescheduling of the container process. A ship delay model is developed and the problem is solved by combining branching and Tabu Search.

The models developed in this thesis establish the relationship between significant factors and the options for increasing throughput by discovering the bottlenecks. The models are applicable as decision tools for operation planning, yard layout, and cost and benefit analysis for investment in infrastructures.
# Table of Contents

Keywords .................................................................................................................... i  
Abstract .................................................................................................................... ii  
List of Figures .......................................................................................................... vii  
List of Tables ............................................................................................................ ix  
Glossary ...................................................................................................................... x  
Publications arising from the thesis ...................................................................... xii  
Statement of Original Authorship ......................................................................... xiii  
Acknowledgement .................................................................................................. xiv  

1. **Introduction** ....................................................................................................1  
   1.1 Research aims and objectives ................................................................. 2  
   1.2 Outline of the thesis ............................................................................... 4  

2. **Container terminals and literature review** ..............................................5  
   2.1 Seaports .................................................................................................. 6  
   2.2 Unloading and unloading of containers ................................................ 9  
   2.3 Yard operation and planning ............................................................... 11  
   2.4 Multimodal terminals ........................................................................... 19  
   2.5 Ship Delays ........................................................................................... 21  
   2.6 Information technology and telecommunications ................................ 22  
   2.7 Modelling container transport and solution techniques .................... 24  
   2.8 Scheduling in environment with uncertainty ........................................ 26  
   2.9 Conclusions ........................................................................................... 28  

3. **The operating environment** ....................................................................29  
   3.1 Waterside operations ......................................................................... 29  
   3.2 Marshalling area ................................................................................... 31  
   3.3 Yard operations .................................................................................... 31  
   3.4 Storage yard ......................................................................................... 34
4. Modelling of the problem with conventional methodologies .......... 38
   4.1 Objective functions ........................................................................................................38
   4.2 Container process as parallel machine problem ..........................................................40
   4.3 Parallel machine problem with release times and delivery times ..........................42
   4.4 Resource-constraint project scheduling problem (RCPSP) ..................................44

5. Modelling of the problem with a new approach ........................................... 49
   5.1 Two-stage flexible flowshop .........................................................................................49
   5.2 Container process model .............................................................................................51
   5.3 Availability of machines .............................................................................................55
   5.4 Capacity at the marshalling area .................................................................................56
       5.4.1 Marshalling area with limited capacity .................................................................57
       5.4.2 No capacity at the marshalling area ....................................................................57

6. Solution methodologies for the container process model .......................... 59
   6.1 Heuristic algorithms ......................................................................................................59
       6.1.1 Priority rule ............................................................................................................60
       6.1.2 List scheduling ......................................................................................................60
   6.2 Deadlocks and violation of precedence constraints .............................................62
   6.3 Meta-heuristics for solving large-scale problems .................................................65
       6.3.1 Local Search .........................................................................................................66
   6.4 Tabu Search ................................................................................................................67
       6.4.1 Simulated Annealing ............................................................................................69
       6.4.2 Genetic Algorithms ..............................................................................................70
   6.5 Reactive scheduling approach ......................................................................................75

7. Numerical investigations ................................................................................. 77
   7.1 Implementation of CPM and validation ....................................................................77
   7.2 Construction of trial data set .......................................................................................77
7.2.1 Loading/unloading times ........................................................................78
7.2.2 Transfer and handling times in the storage yard .................................79
7.3 Numerical investigations ...........................................................................80
7.4 Yard layout ...............................................................................................82
7.5 Reactive scheduling approach .................................................................84
7.6 Capacity at the marshalling area ..............................................................85

8.1 Multi-berth and multi-ship environment ...............................................87
8.2 Resolving violation of precedence-constraints and deadlocks .............91
8.3 Storage Allocation Model (SAM) ..............................................................93
8.4 Storage allocation scheme and container process sequence .................97
8.5 CPM-SAM ...............................................................................................99
8.6 Numerical investigations .........................................................................101

9. Ship delays .................................................................................................104
9.1 Ship delay model ......................................................................................104
9.2 Lower bound of total weighted ship delay .............................................107
9.3 Solution techniques for SDM .................................................................110
9.4 A case study ............................................................................................113

10. Conclusions and directions for future research .......................................115
10.1 Conclusions ............................................................................................115
10.2 Directions for future research .................................................................117
Appendix A. Solving CPM using LINGO 8 ..................................................119
Appendix B. Selected Computer Code for CPM ..........................................124
Appendix C. Selected Computer Code for validating CPM .........................137
Appendix D. An example on the finding lower-bound of ship delays .........145
References ....................................................................................................150
List of Figures

Figure 2.1 Import and export process ................................................................. 6
Figure 2.2 Unloading of containers .................................................................. 10
Figure 2.3 Gantry crane and terminal truck ..................................................... 13
Figure 2.4 Layout of storage area using gantry cranes ..................................... 13
Figure 2.5 Layout of storage yard using straddle carriers ................................ 14
Figure 2.6 Retrieving a container from the bottom of a stack ....................... 14
Figure 2.7 Stacking of containers - avoiding re-handling ............................... 15
Figure 2.8 Retrieving a container from the bottom of a stack ....................... 15
Figure 2.9 Layout of a multimodal ................................................................... 20
Figure 2.10 GPS technology for transport logistics .......................................... 24
Figure 3.1 Loading and unloading at the same ship bay .................................. 30
Figure 3.2 Storage of containers on a ship and in the yard ............................. 32
Figure 3.3 Import process with and without pooling ...................................... 32
Figure 3.4 Change in travel distance of yard machine due to different storage
assignment plans, (a) and (b) ......................................................................... 33
Figure 3.5 (a) Moving an inbound container to the yard then returning empty. (b)
Moving an outbound container to the marshalling area. (c) Moving an
inbound container and then an outbound one. (d) Unproductive move
between QCs .................................................................................................... 34
Figure 3.6 Containers moved in and out of yard with time for single ship ....... 35
Figure 3.7 (a) Storage scheme for the first ship of a new terminal. (b) Storage of
containers from the second ship and the left-over from first ship. (c)
Storage of containers from the n-th ship and the left-over from
previous ships .................................................................................................. 36
Figure 4.1 $P_m \parallel C_{\text{max}}$ (a) Initial schedule (b) Optimal schedule ............... 40
Figure 4.2 Container process involving one QC and two YMs ....................... 42
Figure 4.3 An activity network (AON) of a container process ....................... 45
Figure 5.1 Make-span of import process ......................................................... 51
Figure 5.2 Setup and processing of containers by yard machines ................. 52
Figure 6.1  (a) Best schedule generated using standard list scheduling algorithms. (b) Optimal schedule ................................................................................................................................. 62

Figure 6.2  Solution space with multiple local minima ................................................................................ 67

Figure 6.3  Neighbourhood search ................................................................................................................ 69

Figure 6.4  Flow-chart for GA ......................................................................................................................... 71

Figure 6.5  New generations from cross-over ................................................................................................ 73

Figure 6.6  New generation from scrambling sub-list ..................................................................................... 73

Figure 6.7  Chromosome representation with machine boundaries .............................................................. 74

Figure 6.8  New chromosome from shifting dummy job .................................................................................. 74

Figure 7.1  Movement of trolley when unloading a container ......................................................................... 78

Figure 7.2  Screenshot from PowerStow (courtesy Navis) .............................................................................. 79

Figure 7.3  Effect of the number of QCs and YMs when the yard is half full ............................................... 82

Figure 7.4  Rows of containers (a) perpendicular to and (b) along-side the quay line ....................................... 83

Figure 7.5  Ship service time against the number of bays per row in the yard for the benchmark problem ......................................................................................................................... 84

Figure 7.6  Effect of capacity at marshalling area on ship time ......................................................................... 86

Figure 8.1  Pooling of yard machines in a multi-berth and multi-ship environment ........................................ 88

Figure 8.2  Generating new storage allocation scheme by (a) swapping and (b) insertion ........................................................ .............................................................................................................. 93

Figure 8.3  Occupancy of storage blocks ......................................................................................................... 94

Figure 8.4  Relationship between process order and storage locations ......................................................... 98

Figure 8.5  Relationship between storage locations on the ship and those in the yard ........................................ 99

Figure 9.1  Schedule of multi-ship on multi-berth represented by partial sequences ......................................... 107

Figure 9.2  Tree representation of two berths and four ships ........................................................................ 111
List of Tables

Table 6.1  Setup times of yard machines ................................................................. 62
Table 7.1  Results from sample data set ................................................................. 81
Table 7.2  Ship service time of benchmark problem with yard originally half full . 84
Table 7.3  Ship service time with machine break downs ......................................... 85
Table 8.1  Sharable storage locations ................................................................... 95
Table 8-2  Results from CPM in multi-berth and multi-ship environment and CPM-  
            SAM ......................................................................................................... 103
Table 9.1  Sample data from four ships ................................................................. 110
Table 9.2  Service time windows with no ship delays ............................................ 113
Table 9.3  Weighted ship delay of nodes with lower bound less than or equal to the  
            initial objective value ............................................................................. 114
# Glossary

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGV</td>
<td>Automatic Guided Vehicle</td>
</tr>
<tr>
<td>AOA</td>
<td>Activity on Arc</td>
</tr>
<tr>
<td>AON</td>
<td>Activity on Node</td>
</tr>
<tr>
<td>ASC</td>
<td>Automated Straddle-carrier</td>
</tr>
<tr>
<td>ATA</td>
<td>Actual Time of Arrival</td>
</tr>
<tr>
<td>ATD</td>
<td>Actual Time of Departure</td>
</tr>
<tr>
<td>CPM</td>
<td>Container Process Model</td>
</tr>
<tr>
<td>CPM-SAM</td>
<td>Model integrating CPM and SAM</td>
</tr>
<tr>
<td>CPM-SD</td>
<td>Model adapted from CPM with weighted ship delay as objective</td>
</tr>
<tr>
<td>DP</td>
<td>Dynamic Programming</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>GA-LIS</td>
<td>Genetic Algorithm - feasible solutions generated from list-scheduling</td>
</tr>
<tr>
<td>GA-PM</td>
<td>Genetic Algorithm - feasible solutions generated from set of list-sequences on parallel machines</td>
</tr>
<tr>
<td>GPS</td>
<td>geographical positioning systems</td>
</tr>
<tr>
<td>IP</td>
<td>Integer Programming Problem</td>
</tr>
<tr>
<td>MIP</td>
<td>Mixed Integer programming problem</td>
</tr>
<tr>
<td>NP Hard</td>
<td>Nondeterministic Polynomial-time Hard</td>
</tr>
<tr>
<td>PDA</td>
<td>Personal Digital Assistant</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>QC</td>
<td>Quay Crane</td>
</tr>
<tr>
<td>RCPSP</td>
<td>Resource-constrained Project Scheduling Problem</td>
</tr>
<tr>
<td>SA</td>
<td>Simulated Annealing</td>
</tr>
<tr>
<td>SA-LIS</td>
<td>Simulated Annealing - feasible solutions generated from list-scheduling</td>
</tr>
<tr>
<td>SAM</td>
<td>Storage Allocation Model</td>
</tr>
<tr>
<td>SA-PM</td>
<td>Simulated Annealing - feasible solutions generated from set of list-sequences on parallel machines</td>
</tr>
<tr>
<td>SDM</td>
<td>Ship Delay Model</td>
</tr>
<tr>
<td>STA</td>
<td>Scheduled Time of Arrival</td>
</tr>
<tr>
<td>STD</td>
<td>Scheduled Time of Departure</td>
</tr>
<tr>
<td>TEU</td>
<td>Twenty-foot Equivalent Unit</td>
</tr>
<tr>
<td>TS</td>
<td>Tabu Search</td>
</tr>
<tr>
<td>TS-LIS</td>
<td>Tabu Search - feasible solutions generated from list-scheduling</td>
</tr>
<tr>
<td>TS-PM</td>
<td>Tabu Search - feasible solutions generated from set of list-sequences on parallel machines</td>
</tr>
<tr>
<td>VBS</td>
<td>Vehicle Booking System</td>
</tr>
<tr>
<td>YM</td>
<td>Yard Machine</td>
</tr>
</tbody>
</table>
Publications arising from the thesis

Wong A. and E. Kozan. 2008. Container process scheduling and storage locations at seaport terminals. (under review)

Wong A. and E. Kozan. 2007. Optimisation of container process at seaport terminals. (under review)


Statement of Original Authorship

The work contained in this thesis has not been previously submitted to meet requirements for an award at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Signature:

Date: 15 April 2008
Acknowledgement

I am particularly indebted to my principal supervisor Professor Erhan Kozan for his excellent guidance throughout, and his tremendous patience in correcting every report I submitted to him. I would also like to thank my associate supervisor Professor Luis Ferreira, the examiners and other members of the Committee for review of the final draft.

I am grateful to Dr Paul Corry, Dr Robert Burdett and other members of the Operations Research Group within Queensland University of Technology for their invaluable ideas for the research.

I would like to thank Port of Brisbane Authority, Patrick Terminals, and Hongkong International Container Terminals for the guided port visits; and Pacific National for the visits to their rail-intermodal terminals. These visits provided a better understanding of container traffic network.

In addition, I would like to thank Kristine Wellings, my sister Christina and my daughter Michelle for their help in the writing process. Special thanks to my wife, Natalie, for her support, patience and understanding throughout the years of my PhD journey.
Chapter 1

Introduction

Containerisation of ship cargo was first introduced in 1956 (Levinson (2006)), aiming to cut down the costs of maritime transport through reducing cargo handling costs. Instead of loading/unloading each piece of transport item to or from a ship in a labour-intensive manner, containerisation increases the efficiency and speed of transport by reducing the packing requirements and handling processes at all transfer points - between port, rail and road. At the end of 2005, the world container fleet was expected to have increased to 21.6 million TEUs (twenty-foot equivalent units) (UNCTAD (2006)). Countries without adequate unitized transport services will be disadvantaged in their international trade (Castro (1999)).

In an attempt to obtain economies of scale, new ships are built with much greater capacity. To date, the largest container vessel can carry 11,000 TEUs. However, the deployment of larger ships demands huge investment in providing greater depth alongside the berth of the calling ports, and more powerful quay cranes with long outreach and lift height. The super-post-Panamax cranes, for example, have an outreach of about 66 m (23 containers wide), costing the Port of Oakland US $7 million each to build (Oakland (2005)). For efficient operation, the ports also require a larger storage yard, and a better road and rail infrastructure. To satisfy the growing demand of container berths, ESCAP (2005) estimates that US$27 billion is needed between 2002 to 2015 for 569 new container berths in the Asia and Pacific region.

In recent years, more and more ports are deregulated from government control. This situation leads to greater opportunity for port operators, but also introduces greater competition between ports. For example, options are available for consignment from Singapore to Sydney by direct shipping; and by shipping to Brisbane and then to Sydney by rail or by road.
Under "cash on delivery" (COD) arrangement, the consignor would obviously prefer a quicker and less expensive route. For this reason, port operators are constantly under pressure to increase the port throughput and reliability, and to keep the costs at a minimum. Investing in a fast, powerful shore crane and other yard equipment may or may not be a solution to the problem. Such investment would incur costs that will eventually have to be reimbursed by shippers. There is also a possibility of shifting the bottleneck from a part of the port system to other parts of the system. An alternative is to investigate the main factors influencing terminal operating performance – operating strategies, physical layout, ship plans, management/work practice, pick-up-delivery cycle times as well as lifting equipment and customer requirements.

Over the last decade, container terminal logistics have gained increasing attention from researchers. Many simulation and analytical models have been proposed on ship berthing, stowing and restowing, container handling and transfers, routing of yard equipment and so on. However, with advancements in technology, new operating environment and infrastructure in ports, new approaches to problems are necessary. For instance, improvement in telecommunications now allows shippers and terminal operators to exchange detailed information for each shipment. Online booking systems have also been implemented in many ports. This allows truck drivers to book the time for retrieving and delivering containers at their most convenient time. Another advantage is that terminal operators will be able to perform resource and operation planning in advance. For researchers and consultants, faster computer systems and improvement in scheduling techniques enable the solving of complicated real-life problems within a reasonable amount of time.

1.1 Research aims and objectives

Multimodal container terminals are an important part of the logistics systems in international trade. At each terminal, inbound containers are unloaded from ships and stored in the yard before being further transferred to the hinterland by rail or by
The yard is also a storage area for outbound containers which are moved into the yard in preparation for loading onto ships. The performance of the container process affects the amount of time a ship is required to stay at the port. The efficiency of the container process is affected by the slack and blocking times between machine operations. To address these important issues, this thesis investigates the scheduling and sequencing of container process in a multi-ship and multi-berth environment, and their relationship with the storage location of containers. The main goal of this research is to develop new models and optimisation techniques, which are applicable or readily adaptable to complicated real-life environment. This is to improve the efficiency and throughput of multimodal container terminals, reduce the turnaround time of ships, and to develop an analytical tool for yard operation planning. Ultimately, the improvement in transport efficiency will reduce the costs of international trade and be beneficial to the economy of the nation.

This study is concerned with the ship sequencing, loading and unloading of containers, yard-layout, storage locations, and the handling and transfers of containers. The real-life problems are complex and are very large in size. The proposed model cannot be solved in reasonable time using exact solution techniques like branch-and-bound as well as dynamic programming. For this reason, effective heuristic and meta-heuristic solution techniques are developed and implemented as computer software to solve the formidable problems in a reasonable amount of time.

Similar to other industrial environments, ports can face disruptions due to machine breakdowns. When such an event occurs, an initially optimised schedule may become inefficient and even infeasible. The port operators must be well-prepared to deal with such uncertainty in order to improve the reliability of their services. This thesis aims at developing a model responsive to the disruptions on the availability of machines.
Ships are expected to arrive and leave within a scheduled time window. However, they may arrive late due to poor weather conditions, or congestions at the previous port. Even though the port will normally receive the notice of late arrival twenty four hours in advance, it is still a great challenge to re-arrange the schedule without affecting the other ships. As cost of a ship delay is very high, this thesis also aims to develop a model for minimising ship delays.

1.2 Outline of the thesis

This thesis consists of ten chapters. In the next chapter, an introduction of container logistics and a review of related literature are presented. Chapter 3 outlines the operating environment of multimodal container terminals. Chapter 4 investigates conventional mathematical models for the scheduling of the container process in a single-berth and single-ship environment, and discusses the choice of objective functions and their implications. Modelling the problem with new approach is presented in Chapter 5. To solve this complex problem, Chapter 6 investigates heuristic and meta-heuristic solution techniques suitable for solving the problem. In applying meta-heuristic techniques, a new alternative schedule may not be feasible due to deadlocks or violation of precedence-constraints. This chapter proposes algorithms for resolving the infeasibility and making the new schedule feasible. In Chapter 7, numerical investigations are presented. The effect of an intermediate buffer at the marshalling area is investigated in Chapter 8. This chapter also studies the relationship between loading/unloading sequence, the transfer of containers to/from yard, and the storage location of containers. A model for improving the scheduling of container process by iterating through the refinement of storage locations and re-scheduling is proposed. Chapter 9 investigates the ship delays at ports. Models are developed as decision tools for analysing alternate berth allocation and ship sequencing. Finally, Chapter 10 will summarise the thesis, and make concluding remarks as well as recommendations for future research.
Chapter 2

Container terminals and literature review

Before the introduction of containerisation, break bulk cargo was transported piece-by-piece. This handling method made sea freight very labour intensive and costly. Containerisation helps reduce freight charges, and increase the security of the goods carried. Cargo can now be transported via different means without the need of packing and re-packing in-between. The lowering of transport costs not only makes containerisation an attractive option, it also induces the increment of the prosperity level of international trade. According to ESCAP (2005), the world trade growth was roughly 1.5 times the world economic growth between 1950 and 1990. The ratio increased to 2.5 times in the period 1990-2004. This pattern indicates that trade has become an ever more crucial component of global economic activity.

Containers were originally designed so that they could be removed from their chassis for easy transport via rail or other chassis. The idea was to avoid the need of re-packing of cargo when changing trains, or changing between train and truck. This method led to a greater security at less transportation costs, and enabled door-to-door delivery of goods. The concept of containerisation was quickly adopted by the shipping industry. Over the years, the world fleet of fully cellular containerships continued to expand and at the beginning of 2006, there were 3494 ships with a total capacity of over 8 million TEUs (UNCTAD (2006)).

Although containerisation gradually gained popularity in the transportation world since 1950s, its initial growth was hindered by lack of standards. Containers of a transport company might not fit on the trucks, ships or railcars of another company. The size of containers from Europe might not fit the size of trucks and railroads in the USA. For this reason, the International Standards Organisation (ISO) was asked to develop standards for containers throughout the world. The development of these standards was by no means an easy task, as the design of containers affected shipping as well as rail and road transport. By 1970, the draft by ISO Technical
Committee 104 was gradually accepted by the vast majority of the concerned parties. Of all the nominal standard lengths of containers, 20 feet and 40 feet were the most popular.

2.1 Seaports

A seaport container terminal is an interface between container vessels and the hinterland. Figure 2.1 shows the flow of containers in the import and export process.

![Diagram](image)

**Figure 2.1** Import and export process

In the past, ship arrivals were modelled using queuing theory (for example, Noritake and Kimura (1983), Noritake and Kimura (1990) and Sabria (1986)). At present, most container vessels are operated as linear and ship schedules are known in advance. It is essential for the ship to adhere to the time window. A delay in ship arrival affects not only its current shipment, but also other calling ships and their subsequent voyages as well. Some terminal operators may re-assign a late arriving ship to the end of the queue, consequently causing further delay to the ship for the next port of call.
With the advancement of information technology and telecommunications, the port and the calling ship can exchange important information much earlier in advance including:

1. the number of containers on the ships;

2. the details of each container, for example, its weight, location on the ship, whether this is a reefer (refrigerated container, requiring electric power) or contains dangerous goods; and

3. the final destination of each container

The above information is important for developing the loading/unloading plan, as well as yard planning. A heavily loaded ship requires deeper water. Before a ship arrives at a port of multiple berths, a decision has to be made on which berth is to be used, and how many quay-crane (QCs) and shifts of workers are to be assigned. Obviously, the berth assigned should not be too far away from storage area allocated for the containers in the yard. Otherwise, the containers will need a longer time to process.

Literature review related to seaport and berth allocation

Atkins (1983) gives comprehensive descriptions on how a container port operates. When a container arrives at the port by road or by rail, it is transferred to an assigned location in the yard. Yard storage areas are divided by vessel, port of destination, container size, and, at times, by weight category. Loading of export containers to a ship must be well planned, taking into account their destination and the weight distribution of the ship. Steenken et al. (2004) provide classification of problems surrounding terminal operations and suggestions for future research. The paper also points out that stacking and storage logistics are becoming increasingly important as a result of growth in container traffic, and are also becoming more complex and sophisticated. Psaraftis (1998) recommends research into a list of problems with the container terminal: scheduling berthing priorities; ship booking by “rendezvous”;
berth and cranes allocation problems; yard management problems in minimising movements of straddle carriers; and route and schedule consolidations.

Noritake and Kimura (1983) use queuing theory models, $M/M/S(\infty)$ and $M/E_k/S(\infty)$, to analyse the average waiting time of general cargo ships. The paper suggests that the total cost of port during the period of port operation is equal to the sum of cost related to berths and the cost related to ships. The optimum number of berths is determined using the marginal cost of berths. The optimal berth capacity for each queuing model queuing under consideration is then obtained by a capacity curve. In the authors’ later study, Noritake and Kimura (1990) extend their previous work to include cost comparison with inland transport. Kozan (1994) presents a model to analyse the economic effects of alternate investment decisions for seaport systems. The author emphasises that the best strategy in investment decisions for expansion of the seaport system is one which provides either the maximum present value of net benefits or the minimum present value of the total costs over the planning perspective.

Cordeau et al. (2005) investigate the sequence of ports visited by vessels, called services. A model is developed to solve the problem for allocating areas of the yard and the quay to services in a planning horizon. Time-based Tabu Search and space-based Tabu Search are proposed to solve the discrete case and the continuous case of the problem. Lim (1998) states that one of the planning problems encountered at the port is to decide if a given set of ships can be berthed in a section of the port given by certain berthing constraints. Ships may be of different lengths and arrive at the port at different times. Every ship has a different expected duration of stay. When two ships are berthed side by side, there is a minimum clearance distance between them. There is also a minimum distance from the end of a section. The ship berthing problem is formulated as a restricted form of the two-dimensional packing problem. The problem is solved using graph theory representation. Fagerholt (2001) formulates the ship scheduling problem as a set partition problem. Different types of cost functions (including linear and quadratic) are analysed and the problem is solved using network-flow and dynamic programming techniques.
Imai et al. (2001) suggest that low usage of berths coupled with small amount of cargo handled results in relatively high port charges per container. The paper studies the problem of berth assignment to ships in the public berth system and formulates the problem as Mixed Integer Programming Problem (MIP). Two models are proposed - a model for static berth allocation where all the ships are already in port; and a model for dynamic allocation where some ships may not arrive at port before berth assignment. Lagrangian relaxation was applied to find optimal berth assignment in relation to the minimum total time for ships staying in port within a planning horizon. In a later study, Imai et al. (2003) extend the dynamic berth allocation problem to include the water-depth and the length of the quay. The problem is solved using genetic algorithms (GA). Legato and Mazza (2001) apply simulation in optimising berth allocation. In Brown et al. (1994) ship berthing position is restricted by a number of factors including the length of the ships, the pier-side depth and the number of shore power cables. The authors conclude that optimisation of berth scheduling can be achieved using simulation. Based on the finding that the vessel inter-arrival time is exponentially distributed, Lai and Shih (1992) apply simulation to analyse different container berth allocation policies. They conclude that the existing "Home Berth" policy is not as good as the three other proposed new policies. Suh and Lee (1998) propose an expert system for ship berthing and material discharging.

2.2 Unloading and unloading of containers

The inside of a containership is constructed in the form of compartments. In each opening, the ship may hold one 40-foot or two 20-foot containers. Containers are stacked sequentially on top of each other (see Figure 2.2). Additional movable support is provided to the upper containers by the ship to achieve higher stacking capability. For this reason, extra time is needed in handling the movable support during loading and unloading activities. Furthermore, the time required to load/unload a container to/from a ship depends on the location where it is stored within the ship as well as the speed of the QC.
A ship often carries containers for several destinations. If containers of the current port are stored below those for other ports, QC must unload those on top first. After the inbound containers are unloaded, containers for other ports have to be loaded back onto the ship. This process is called re-stowing. Re-stowing takes up additional process time of QC. In developing the load/unload plan, the weight and balance of the ship must be taken into consideration. If two or more QCs are assigned to a ship, these QCs cannot cross each other.

When the work on a ship bay is complete, a QC can move along-side the berth to work on another ship bay. The marshalling area is where the QC can pick-up/drop-off containers to/from the ship. Depending on the type of QC in use, this area is normally six lanes wide. At each unloading position, import containers are unloaded to one of the lanes and stacking is to be avoided. Similarly, export containers are moved to one of the lanes before loading to the ship by the QC during the export stage.

**Literature review related to loading and unloading of containers**

Avriel et al. (2000) investigate container ship stowage problem: complexity and connection to the coloring of circle graphs. Kim et al. (2004) suggest a model for optimising the loading sequence by taking into consideration the weight category, overweight, re-handling, and the minimum distance between transfer cranes. Wilson and Roach (1999) studies a stowage planning and develops a model for minimising the number of hatch-lid moves, the number of over-stows, as well as the number of "cargo blocks" occupied. A model is proposed by regarding the total number of
holds as a queuing system. Bish (2003) studies a special case when import containers are being loaded to a ship and export containers are being unloaded to another ship at the same time. The author claims that there may be time saving with combined trips. Peterkofsky and Daganzo (1990) study the amount of work required for a ship as holds (or hatches). When cranes are shared only within a small part of a port, the crane scheduling problem can be formulated as parallel machine problem. The paper also suggests that a crane scheduling problem can be considered as machine scheduling or project scheduling.

2.3 Yard operation and planning

After the ship is berthed, inbound containers are unloaded and are moved from the marshalling area to the storage yard. Containers of the same length are normally stacked on top of each other, and are allowed to stay there for a few days free of charge. The export process is the reverse of the import process. Several days before the scheduled arrival of the ship, the port accepts outbound containers to the yard until a certain cut-off time. Prior to loading containers to the ship, a loading plan is prepared by taking into account the final destination and the type of each container, the weight category, and the maximum allowable weight for each stack, as well as maintaining the overall balance of the ship. Outbound containers taken into the port by rail have a predictable arrival schedule. On the other hand, arrivals by road have always been considered as a random process. In recent years, sophisticated information technology has been adopted in most Australian ports. A vehicle booking system (VBS) is implemented to allow truck operators to book the time for container delivery as well as the time for retrieval in advance. In addition, geographical positioning systems (GPS) and personal digital assistants (PDA) are becoming more affordable. An increasing number of truck drivers are able to communicate with central control office of their present location. The arrival and departure of containers are therefore more predictable.
If trucks and/or automatic guided vehicles (AGV) are used to move containers from the storage yard to the marshalling area, these machines must wait at the marshalling area until the containers are picked up by QCs. During the unloading operation, QCs must also wait for the trucks/AGVs to drop off the containers before the next move. Similar situations apply to the loading operation. Trucks and/or AGV are normally used in conjunction with the expensive powerful gantry crane in the storage yard, which possesses fast moving and high stacking capability. Other yard machines, like fork-lifts, reach-stackers and straddle carriers have handling capability to lift up the containers being dropped off to the marshalling area by the QCs. During the export stage, these yard machines can also drop off the containers to the marshalling area for loading to the ship. Note that at each loading/unloading position, the marshalling area forms a small buffer. However, congestion occurs when the marshalling area is full and a QC unloads more containers than the yard machines can handle. The QC must then slow down or cease processing until space is available again. Similarly, yard machines must slow down or stop during the export stage if they operate faster than the QC can handle.

AGVs are the first kind of yard equipment built with automation. However, AGVs must follow definite paths. On the other hand, automated straddle-carriers (ASCs) are more flexible in their operations. ASCs are straddle-carriers with built-in automation. Located by microwaves and controlled by advanced robotics technology, they can move along any path, pick-up and drop-off containers. After an initial successful trial, ASCs are now in full operation at Port of Brisbane. By using ASCs, the terminal has the potential of saving not only labour costs, but also the costs in line-marking and lighting. Technology for robotic controls also minimises the damage to the ground by assigning slightly different paths for ASCs each time when they travel along the similar route in the yard.

The storage yard is a large area divided into sections. Each section consists of multiple rows and each row consists of multiple bays for containers to be stacked together. The rows in each section may be arranged along-side or perpendicular to the water-line. Gantry cranes can only move containers with their width (see Figure
2.3), and most gantry cranes are designed to move forwards and backwards only. For this reason, rows of containers are stacked close together and the layout of the yard must follow the characteristics of the operation of the gantry cranes (see Figure 2.4).

![Figure 2.3 Gantry crane and terminal truck](image)

Straddle-carriers, fork-lifts and reach-stackers are multi-purpose machineries which are commonly used in small and medium size seaport terminals. They are capable of picking up, dropping down, and carrying containers to other locations. However, they are usually only capable of stacking containers to three or four levels and
require more space between rows for accessing the containers. Hence the yard layout is different from the one for gantry crane (see Figure 2.5).

Figure 2.5 Layout of storage yard using straddle carriers

Depending on the type of yard equipment, containers may be lifted from the top corners, the bottom corners, or by other methods. In any case, pick-up time is needed to secure a container before further transfer. At the final destination of the container, drop-off time is again needed to place the container on the ground. To retrieve a container from the bottom of a stack, extra re-handling time is necessary as shown in Figure 2.6.

Figure 2.6 Retrieving a container from the bottom of a stack

Re-handling increases the processing time and also the risks of damaging containers. Import containers arrive in batches along with the ship. They can be unloaded and stored in the yard from ground level up. On the other hand, export containers arrive at the port mostly in a random fashion. When loading export containers to a ship, lighter containers are always placed on top of the heavier ones. If space allows, the
amount of rehandling can be minimised by careful planning of storage locations. To avoid unproductive re-handling of export containers in the yard, heavier containers should be stacked on top of lighter ones as shown in Figure 2.7.

![Figure 2.7 Stacking of containers - avoiding re-handling](image)

Figure 2.8 shows an inbound container being retrieved from a stack. Container C must be placed somewhere else temporarily before Container D can be collected. With a vehicle booking system, rehandling of containers can potentially be reduced. For example,

1. if there is no firm booking on when Container A and Container C will be collected, the pick-up time for Container C should be arranged before Container A; and
2. if Container A will be collected before Container C, Container C should temporarily be reallocated to somewhere else.

![Figure 2.8 Retrieving a container from the bottom of a stack](image)

Literature review relating to yard operations
Vis and Koster (2003) give a comprehensive review of literature on container terminal planning and operations. The paper discusses the process at the container terminal, including arrival of ship, loading and unloading of the ship, movement of containers between ship and stack, stacking of containers, and inter-terminal transport. There is also a review on the complete container terminal operations. The authors conclude that the literature is in general considered to be necessary to simplify the problem before it can be solved by analytical models. Analytical models that are used most often are mathematical programming models, branch and bound models, queuing models, network models and assignment problems. On the other hand, simulation can be used. The authors state that it is a time-consuming job to develop and validate these models. They also recommend extending models for simple cases to more realistic situations, and more attention should be paid to the combination of various kinds of equipment in a terminal.

Lee and Hsu (2006) investigate the pre-marshalling problem in container terminals. With an initial layout, the problem is to have an efficient way to shuffle containers around to the final layout of the yard. An integer programming model is proposed, with a multi-commodity network flow problem embedded within. Grunow et al. (2006) study the operations of AGVs, taking account of their multi-load capability. The paper considers event-based logic of the logistic control system and presents simulation investigations of the dispatching strategies of AGVs in both on-line and off-line environment.

Taleb-Ibrahimi et al. (1993) consider ships operating on a schedule. Assuming containers must travel on specific ships and each ship is scheduled to carry the same number of containers, the number of import containers can be determined. In a static storage space allocation, a container’s position does not vary during its stay at the terminal. The paper shows that storage space required is related to the number of total number of import containers to be received. In a dynamic storage allocation, the assignment of a container to yard can happen any time after it has arrived before cut-off – containers arriving before this assignment are temporarily stored in a “rough-pile” for later transfer. The paper claims that the procedures developed can
be applied either to minimise the amount of space needed to accommodate traffic or to minimise the number of containers rough-piled per unit time, given a fixed amount of space.

Assuming all containers in the storage area have the same probability of retrieval, Castilho and Daganzo (1993) show that the expected number of moves is related to the height of each stack. Kim (1997) studies the expected number of rehandles on block level and concludes that this expected number depends on the number of rows, bays and tiers (levels) of the block. With the expected number of rehandlings, Kim and Kim (1999b) analyse the optimal height of stacks under different arrival patterns of import containers. In Kim and Kim (1998), a cost model is developed for the costs of space, transfer cranes and trucks. This model is applied to determine the optimal amount of resources required. Kim and Bae (1998) discuss optimising re-marshal export containers in container terminals. Bay matching and movement plan are first solved in the first stage. Task sequencing problem is solved during the second stage. With the model in determining re-marshalling export containers, Kim and Kim (2001) develops a different approach to determine the optimal amount of storage space required, and the authors also discuss the optimal number of transfer cranes for handling import containers. In their model, the estimated cost of terminal operation is related to space cost, travel time cost of the transfer crane, and the pick-up cost. Queuing system is applied to model the transfer operation of inbound containers by assuming that the arrival process of outside trucks follows Poisson process. The results indicate that the optimal number of containers in yard-bay increases as space cost increases. The optimal number of transfer cranes is insensitive to the space cost. Kozan (1997) outlines the analytical and the simulation planning models for container terminals. The paper also gives comparison of the different models as application to seaports.

Kozan and Preston (1999) discuss the handling and transfer of containers at seaport terminals. When containers in the storage yard can be stacked on top of one another, the time taken to get a container from the lower levels needs to be incorporated in the loading times. Assuming the transfer time in accessing lower containers is negligible, additional time is required to pickup or “lock” onto the container and
drop-off the container. The time required to move a container between the storage yard and the marshalling area depends on the number of locks and the time of the journey. The Container Location Problem (CLM) is proposed and the problem is solved using Genetic Algorithms (GA). The paper also presents how the model can be applied to analyse storage policy, maximum heights of container stack and the degree of fullness in the yard storage area. In the authors’ later studies, Preston and Kozan (2001) develop the Container Transfer Model (CTM) for the optimal schedule; and Kozan and Preston (2006) suggest an integrated model in combining both CTM and CLM. The integrated model is designed to find the best storage arrangement and handling schedule to minimise the ship turnaround time.

In Zhang et al. (2002), the authors study the movement of rubber tyred gantry cranes (RTGCs). The authors observe that there can only be at most two gantry cranes in each block, and inter-zone RTGC movements are carried out only once per day at midnight. The problem is formulated as MIP and is solved by Lagrangian relaxation to minimise the total delayed workload in the yard. Lai and Lam (1994) applies queuing theory and simulation to study the throughput and utilisation of yard equipment under various allocation strategies. Kim et al. (2000) propose a dynamic programming model to determine the optimal storage location of an arriving export container. Kim and Kim (1999a) discuss a routing algorithm for a single straddle carrier to load export containers to a ship according to a load profile. The optimal route is obtained by dynamic programming. Kozan and Wong (2002) propose a model in scheduling container handling and transfers process at a multimodel terminal. Before a container ship arrives at a port, the terminal operator would have already received the list of import containers, and their location on the ship. An unloading sequence is worked out according to the location of containers, the number of cranes assigned and the balance of the ship during the unloading. After all import containers have been unloaded to the marshalling area, they are handled and transferred to the storage area by yard machines (forklifts, reach-stackers, etc). Different types of yard machines have different handling and transfer speeds. The complete time for a particular container to be moved to the storage area depends on when and what type of yard machine is available, as well as the location for storing
the containers. The reverse process applies to export containers. The problem is formulated as MIP.

2.4 Multimodal terminals

Outbound containers are brought into the port and inbound containers are brought out of the port by road or by rail. For this reason, a good road and rail network plays an important role in the efficiency of the port. These two means of transport have different characteristics in the way they transfer containers into and out of the port. Trains arrive and leave the port in a pre-arranged schedule. The arrival of each train will bring in over a hundred TEUs, and takes away a similar amount of containers on its departure. On the other hand, trucks arrive and leave the port more frequently but they carry a maximum of two TEUs each time.

Figure 2.9 shows the layout of a multimodal terminal. The overall efficiency of this complicated system is affected by its sub-systems, for example, ship scheduling, berth allocation, performance of loading/unloading, storage yard, as well as road and rail network. Investing in fast, powerful QC and other yard equipment may or may not be a solution to the problem. Such investment will incur costs that eventually have to be reimbursed by shipping companies. There is also a possibility of shifting the bottleneck from a part of the port system to other parts of the system.
Figure 2.9 Layout of a multimodal

Literature Review relating to multimodal terminals

Macharis and Bontekoning (2004) give a review of problem definitions and models on recent research on intermodal freight transport. Kozan (2000) proposes a network model in studying container transfer at multimodal terminals. The problem is formulated as MIP. The expected number of containers moving into each section of the storage area by all yard machines is the same as the expected number of
containers moving out of the section. With trial data from Port of Brisbane, the model is successfully applied in sensitivity of yard equipment. In the context of port planning this model would be useful in the decision process of port facilities. Kozan and Casey (2006) develop a model for minimising ship-delays in multimodal container terminals. Large-scale sample problems are solved using various meta-heuristics. Corry and Kozan (2006) investigate the load planning of intermodal trains. Their study concerns the double handling of containers and the mass distribution of trains, and a model for load planning assignment is developed for the dynamic environment.

2.5 Ship Delays

A newly built container vessel of 2750 TEUs is about US $40 million, and its average revenue is about US $35,000 a day (see SSY (2004)). At such high dollar value, a container vessel is expected to operate to their full capacity. Normally, the owner and the port operator work closely together for a scheduled arrival and departure time of the ship. If the ship does arrive at the scheduled time, the port is expected to complete loading and unloading of the ship so it can leave before the scheduled departure time. However, a ship delay may occur if the port cannot complete the work on the ship before the scheduled time due to disruptions like congestions, labour dispute and machine breakdowns. The delay at the port may also cause further delay at subsequent ports as well.

The scheduled arrival and departure times of a ship are determining factors on berth allocation and other resource planning. Regardless of whether it is the result of poor climate conditions or delay at the previous port of call, a late-arriving ship poses a challenge to the port operator. The service time window for the ship is to be adjusted accordingly, and perhaps, the allocation of berths is to be revised as well. Some operators will penalise the late-arriving ship to a much later time window,
while others will try to complete the work on the ship according to original scheduled time of departure.

Literature Review relating to Ship Delays

Daganzo (1989) studies the crane allocation problem and develops models for maximising the cost savings by having ships to depart earlier. The proposed static case assumes that all ships are ready for processing at the start of the planning horizon by a number of identical quay cranes. This assumption is relaxed in the dynamic case, allowing for ship arrival at different intervals within the planning horizon. To some extent, the problem is formulated as parallel machines problem with pre-emption. However, the paper does not take into account the fact that cranes can not cross each over. Also as the author points out, the time in moving cranes between holds must be small when compared with the hold handling times. In another study, Daganzo (1990) considers the number of holds of the berthing ships as randomly distributed and that there is an infinite ship queue. The author suggests applying the formula for an expected number of busy cranes and the expected number of idle cranes to calculate the berth throughput. The paper also investigates the relationship between ship delay and crane operations.

2.6 Information technology and telecommunications

In the last decades, many ports have invested heavily in information technology and telecommunications to cope with the increasing work-load. Information on the arrival/departure time of ships and the goods they carry is now more readily available to ports; and in case of ship delays, alternative arrangement can be planned with early advice.

In the past, load/unloading plans were done by hand which took many hours to complete. With the help of in-house or commercial software, terminal operators can complete these plans in a much shorter time. For example, Navis has a product
called PowerStow for automated vessel planning. Also, tracking containers on yard used to be a difficult task. Now with geographic positioning system (GPS) and/or micro-waves technology, the movement of the transfer machines can be tracked and the location of containers are then stored in database. This enables further improvement of yard operation. For example, PrimeRoute from Navis is designed for increasing the productivity of straddle carriers and terminal tractors. The idea is to have a better pooling of yard equipment in use. However no further detail is available on how the system is optimised.

GPS technology is also more commonly used in ships, trucks and trains. This enables a centralised operational planning and control of the transport network. With growing popularity of personal digital assistant (PDA), truck drivers can get immediate notification when containers are available for picking up. They can also make bookings of retrieval and delivery of containers while on the road. When a disruption occurs within the transport network, it is possible for terminal operators and the truck drivers to reschedule the pick-up and delivery of containers to minimise the impact of the disruption.
To assist researchers, consultants and port operators, there are many simulation software packages available - for example, Arena, Extend, and Simul8. For optimisation tools, GAMS, CPLEX, SOLVER and LINGO are example of commercial software available. On the other hand, the performance of computers continues an upward exponential trend. At the time of writing, the fastest computer system can do 280 trillions of calculations per second (Top500 (2006)). These powerful technologies lead to research opportunity on real-life complicated problems.

2.7 Modelling container transport and solution techniques

Modelling techniques used in container transport are broadly classified into two main-streams - analytical and simulation models. Simulation is a technique which imitates the more complicated real-life environment and gives an insight of the interaction between various sub-systems. After a model is developed, a simulation
process would run the model through time in order to generate representative samples. (mostly on a computer). By inputting different parameters, operation strategies, or design, one can select the best option from the results of the simulation. However, it is not always possible to generate all possible scenarios to run through the simulation. Usually simulation is not designed to solve optimisation problems. For optimisation problems, popular analytical models are mixed integer programming (MIP), network, queuing theory and parallel machine problem. When working with complicated systems in container transport, it is necessary to simplify the problem in a way so that the problem can be solved within reasonable time.

In solving a problem using analytical models, exact solution techniques like branch-and-bound, dynamic programming, and linear programming, are only applicable to small size optimisation problems. For NP hard problem of large size, exact solution may not be obtainable within a reasonable amount of time. The alternative goal is to look for a "good solution" instead. Empirical techniques found to perform well (for example, the despatching rule shortest-processing-time-first) are referred to as heuristic algorithms. These algorithms may not be able to provide the optimal solution, but they can provide a mechanism for obtaining a good feasible solution quickly. Meta-heuristics are search strategies combining with heuristic algorithms, in hoping for obtaining the best solution. Tabu Search (TS), Simulation Annealing (SA) and Genetic Algorithms are meta-heuristics which are widely accepted frameworks for solving NP hard problems (see Chapter 6 for more details).

**Literature review relating to modelling and solution techniques**

A good introduction of branch-and-bound technique, and mixed integer programming problem can be found in Winston (1994). Brucker (1995) and Pinedo (1995) explain concepts in scheduling, including parallel-machine problem, flow-shop problem as well as job-shop problem. They also discuss the complexity of various scheduling problems, and algorithms in solving these problems. Chen and Lee (2002) study a parallel machine scheduling problem where there are penalties in earliness as well as tardiness. The problem is formulated as a set partition problem
and is solved using branch-and-bound. Given multiple resource restrictions, minimising the project cost is Resource-constrained Project Scheduling Problem (RCPSP) with time-resource tradeoffs. Elmaghraby (1977) explains the concepts of activity network and its application to project scheduling under limited resources. A project can be viewed as a collection of activities and events. An activity is any undertaking that consumes time and resources. The concept of precedence is represented by the modelling of projects via “activity network”. In general an activity representation can either be activity-on-arc or activity-on-node. To reduce the time of completion of a project, study on critical path may be an answer. Another approach is to formulate the problem as an integer programming problem (IP). The author presents a model which also takes into account renewable resources, allowing for time-cost trade-offs.

Ahuja et al. (1993)) discuss how network-flow can be applied to various optimisation problems. Bellman and Dreyfus (1962)) outline the general principles of dynamic programming (DP) techniques. It also explains how the Travelling Salesman Problem can be solved using DP. Fagerholt (2001)) and Kim et al. (2000)) both apply DP techniques to solve different problems for container terminal. Peterkofsky and Daganzo (1990)) apply branch-and-bound method to find the optimal solution for the crane scheduling problem. Because of the complexity of the port system, many researchers develop simulation models in solving the problems. Kondratowicz (1990)), Lai and Shih (1992)) and Legato and Mazza (2001)) are examples of such research. In other studies meta-heuristic techniques are applied for solving optimisation problems, for example, Wong and Kozan (2004)), Nishimura et al. (2001a)), Cordeau et al. (2005)) and Kim et al. (2004)).

### 2.8 Scheduling in environment with uncertainty

Production scheduling in real life is exposed to uncertainty environment. Of these uncertainties, some are of stochastic nature (for example, machine processing times) whereas others are less predictable (for example, industrial accident and sudden
breakdown of machines). Scheduling techniques are classified as deterministic, robust and on-line scheduling (see Gan and Wirth (2005)). Models built on the assumption that all problem data are pre-determined and are fixed during the execution of the schedule, are said to be deterministic. Robust scheduling takes a more realistic approach and allows for change in program data. In an environment with a higher level of uncertainty, on-line scheduling techniques aim to produce only good partial schedules since scheduling information may only be available over time.

Uncertainty is often the norm in a production environment. Upon a change in problem data, a feasible schedule generated from a deterministic model may render the schedule infeasible. For this reason, it is important to include uncertainty into machine scheduling so as to improve the quality of the scheduling system (see Floudas and Lin (2004)).

**Literature review relating to uncertainty in scheduling environment**

Floudas and Lin (2004)) review the scheduling of batch and continuous processes, as well as scheduling under uncertainty. The paper classifies existing approaches for scheduling under uncertainty as reactive scheduling and stochastic scheduling. With reactive scheduling, the system adjusts the schedule upon the awareness of uncertain scheduling information or occurrence of unexpected events. With stochastic scheduling, the system takes into account the uncertainty at the original scheduling stage. The aim is to create an optimal and reliable schedule in the presence of uncertainty. Yang and Yu (2002)) prove that the robust version of the single machine scheduling problem with sum of completion times is NP-complete. An algorithm is for the problem using dynamic programming. Lin et al. (2004)) study machine scheduling under bounded uncertainty. From generic MIP, they consider uncertainty in the coefficients and the right-hand side parameters of constraints. Though uncertain at the scheduling stage, it is assumed that the upper and lower bound of the parameters are known in advance. A model is developed for the generic MIP, which is applicable to machine scheduling.
2.9 Conclusions

Over the past decade, there has been a growing interest in the research of container terminals. As Vis and Koster (2003)) conclude, however, the majority of the papers only address the single types of handling equipment; or simplified cases like routing a single straddle carriers. Container scheduling under an uncertain environment has real-life applicability but has been largely ignored. Also, it is generally considered that the ship delay will be minimised by minimising service time, but without taking into account of the time windows for ship arrival and departure. This confirms a gap in the literature that needs to be filled as outlined in the aims and objectives in Chapter 1. This research is motivated in developing models concerning multiple quay cranes, yard machines, and yard-layouts in a multi-berth and multi-ship environment. Uncertainty in the terminal operations including machine breakdowns and ship delays are also investigated. The advancement of information technology and telecommunications provides Operations Research practitioners new research opportunities for solving more complex problems of larger size.
Chapter 3

The operating environment

In a seaport terminal, inbound containers flow from ship to storage yard and then to hinterland. The reverse process applies to outbound containers. This chapter investigates the container process in more detail and analyses the problems surrounding seaport operation and planning.

3.1 Waterside operations

Quay cranes have evolved with the progress of containerisation. They are characterised by their lifting capacity, and the size of the container ships they can handle. A Panamax crane is said to be able to load and unload containers from a container ship capable of passing through the Panama Canal. A Post-Panamax crane can operate on bigger ships that can not pass through the Panama Canal. The newer type Super-Post Panamax cranes are even bigger and more powerful. They can handle ships of 10,000 to 12,000 TEUs, beam over 50m (21 containers), and even have twin-lift capability, and can load/unload up to 35 containers an hour.

When loading/unloading a container to/from a ship, the time required depends on the speed of the QC as well as the storage location of the container on the ship. Furthermore, movable support is provided to the upper containers by the ship to achieve higher stacking capability. For this reason, extra time is needed in handling the movable support during loading and unloading activities. From a pre-determined loading/unloading plan, a database can be developed for the loading/unloading time for each container.

In reality, QC does not lift containers straight-up nor drop them in a straight-down fashion. As the ship is loaded and unloaded, the deck of the ship from sea-level varies and so does the distance each container has to travel. Since it is more difficult
to move QCs along the berth-line, the movement of QCs are to be avoided. To load and unload a ship, QCs may have at least three options:

1. Unload all inbound containers and then load all outbound containers;

2. QCs unload all inbound containers from a ship bay and then load all outbound containers from a ship bay before working on the next one; or

3. Similar to 2, but QCs starts loading before unloading complete on a ship bay (see Figure 3.1).

Option 1 is simple to implement. With this option, there will be more movement of the QCs and more demand on the yard storage area. Besides, the depth of ship varies significantly from heavy loaded to empty loaded. With Option 2, storage area originally occupied by outbound containers may be re-used by inbound containers from the same ship. The depth of ship will not change much. Option 3 is more complicated to implement, but the demand of yard storage area is even less. Besides, QCs may process two containers in one round trip - but an unloaded trolley moves twice as a loaded one.

![Figure 3.1 Loading and unloading at the same ship bay](image-url)
3.2 Marshalling area

To enable fast loading and unloading, the four to six lanes within the reach of QCs are set aside as a marshalling area. The marshalling area is an interface between the land-side and the water-side operation. If containers are taken away from the marshalling area by trucks with no lifting capability, the QC must drop off the containers directly to the trucks. In the terminology of machine scheduling, there is no buffer between the unloading and the transfer operation. If these two operations are not synchronised, one machine must wait for the availability of the other. A similar situation occurs for the loading of containers.

Straddle-carriers and fork-lifts are general purpose equipment. They can pick-up and drop-off containers without the aid of the QC. In fact, QCs must drop-off containers on the ground for this yard equipment to pick-up. Effectively, a buffer is available between the loading and transfer operation. Many busy ports even stack containers in the marshalling area. However, the capacity at the marshalling area is limited and the stacking of containers leads to rehandling and safety issues. On the other hand, stacking containers in the marshalling area complicates the loading operation as the heavier containers must be loaded to the bottom of the ship.

3.3 Yard operations

Not only the loading/unloading time of a container varies as its storage location inside the ship, the transfer time of container also depends on the loading/unloading position and its storage location in the yard. To increase the efficiency of the container process, the loading/unloading operation should synchronise with transfer operation in the yard. For example while a QC is working on a container with longer unloading time, perhaps the YM servicing this QC should be working on a transfer action of longer journey too instead of waiting for unloading to complete.
Figure 3.2 Storage of containers on a ship and in the yard

At the port, each QC is supported by several yard machines. If the two or more QCs are assigned to a ship, pooling or no-pooling policy may be adopted. Under no-pooling policy, each YM works only on designated QC (see Figure 3.3(a)) and is simple to implement. If pooling is allowed, yard machines may work on any QC for load-balancing (see Figure 3.3(b)).

Figure 3.3 Import process with and without pooling
Before a ship can leave the port, all inbound containers must be unloaded and outbound containers loaded to the ship. Under no-pooling policy, the time taken to work on a ship can be estimated from the list of containers to be moved between the yard and the marshalling area. If there is any imbalance in work-load for yard-machines, those containers stored further away can be re-assigned to yard machines that are less busy. However, the travel distance of a yard machine depends on the location of the container in the yard as well as the location of the QC (see Figure 3.4). It is possible to minimise the total travel distance of yard machines by optimising the storage allocation of containers in the yard.

**Figure 3.4** Change in travel distance of yard machine due to different storage assignment plans, (a) and (b)

Assume that the container locations are pre-determined. The transfer time of containers is sequence-dependent when pooling of yard machines is allowed. Figure 3.5(a) and (b) show inbound and outbound containers moved to and from the storage yard. If the yard machine (YM) is to move an inbound container followed by an outbound container, only one round-trip is needed (see Figure 3.5(c)). If the YM is to travel between QCs to collect another container, such movement is non-productive (see Figure 3.5(d)).
Figure 3.5 (a) Moving an inbound container to the yard then returning empty. (b) Moving an outbound container to the marshalling area. (c) Moving an inbound container and then an outbound one. (d) Unproductive move between QCs

Also considering the variation of unloading/unloading time, the water-side and land-side operation may be better synchronised by assigning a task of long processing time to yard machine when a QC will need a longer time to work a container.

3.4 Storage yard

Ideally containers should be stored at the most suitable locations in the yard. However, such locations are scarce resources and may not always be available. In a congested port, containers are stored wherever location is available in the yard. As a result, not only YM need a longer time to transfer containers but fuel costs also increase. Before the ship arrives, the gate is opened a few days in advance for the outbound containers. After the inbound containers are unloaded, they have a few
days to stay in the yard free of charge. Figure 3.6 shows the containers moved in and out of the yard with time for a single ship. Ships arrive and leave, so do containers. After the containers have left the yard, the storage area is then released.

Figure 3.6  Containers moved in and out of yard with time for single ship.

Figure 3.7(a) shows a simplified storage scheme for the first ship of a container terminal. By the time the n-th ship \((s_n)\) is berthed, the storage yard will have containers of \(s_n\) as well as import containers from the previous ship \((s_{n-1})\) and export containers for the next ship \((s_{n+1})\). Figure 3.7(b) shows an example of the new storage scheme after a number of ship arrivals and departures.
Figure 3.7 (a) Storage scheme for the first ship of a new terminal. (b) Storage of containers from the second ship and the left-over from first ship. (c) Storage of containers from the n-th ship and the left-over from previous ships

3.5 Uncertainty in the operation environment

For a scheduling model of container process to be applicable to real life environment, the model must consider the uncertainty in the parameters or scheduling information during the execution of the schedule. The uncertainty may arise from any of the following sources:

1. uncertainty in transfer and handling time of YMs;

2. uncertainty in loading/unloading time due to weather and other changing conditions;
3. late arrival of trucks or trains (which bring in outbound containers and release the yard storage by taking away inbound containers);

4. late or early arrival of ships

5. change in service priority of ship;

6. machine breaks down;

7. accident in the work-place;

8. human resource issues (for example, strike and sudden sick-leave)

If the actual processing times are not much different from the stipulated value, there may not be needed any change to the original schedule if the schedule can be maintained by speeding up or slowing down of the machines. With advancement in technology, some of the above become more predictable. When automated straddle-carriers are in use, the storage location, the route and the speed in processing a container are all generated by computers. VBS and GPS give earlier notification of the arrival of ships, trucks and trains. However, other sources of uncertainty may still need to be addressed for a more robust, practical scheduling system.
Chapter 4

Modelling of the problem with conventional methodologies

In this chapter, several conventional methodologies for solving the problem are investigated. For adopting a conventional approach to the problem, the following assumptions are made:

1. only one ship is considered;

2. all machines operate at the same speeds;

3. the ship length is negligible (the storage location in the yard has the same distance from any of the QCs);

4. re-handling time of containers is negligible;

5. loading cannot start until unloading of the ship is complete;

6. loading/unloading sequence and the storage location in the yard are pre-determined;

7. loading/unloading times are negligible; and

8. the capacity at the marshalling area is sufficiently large

4.1 Objective functions

From the port operators’ perspective, the container process problem is to increase the port through-put by minimising ship service time, while maintaining the quality of
service. To reduce the cost of operation, the number of shifts and fuel usage should be kept at a minimum. On the other hand, the ship operators would expect minimum or no delay in ship schedule. In a competitive environment, the wish of the ship operators will have direct influence on the decision process of the port operators. The following is a list of objective functions that could be considered:

\[ F_1: \] Weighted ship time. Bigger ship may have higher opportunity costs and should be assigned with a higher weighting.

\[ F_2: \] Weighted ship delay. By minimising \( F_2 \), \( F_1 \) may be minimised as well. In a special case, for example, where the actual time of arrival (ATA) of a ship is the same as the schedule time of departure (STD),

\[
\text{Ship delay} = ATD - STD \\
= ATD - ATA \\
= \text{Ship time}
\]

(where \( ATD \) is the actual time of departure)

Note that \( F_1 \) and \( F_2 \) are not identical. In case STD is much later than ATA and ship delay is unlikely, any schedule is likely to be optimal according to \( F_2 \). However, this schedule may not be necessary leading to a shorter ship time.

\[ F_3: \] Total slack-time of QCs. As QCs are expensive equipment, some terminal operators may like to keep them busy. In special cases when one QC is operating on a ship:

\[
\text{Ship time} = (\text{sum of loading and unloading time}) + (\text{slack-time of the QC})
\]

Since the sum of loading and unloading time is almost constant, minimising \( F_3 \) is the same as minimising \( F_1 \).
$F_4$: Combination of $F_1$, $F_2$, and $F_3$ ($F_4 = a_1F_1 + a_2F_2 + a_3F_3$, where $a_1$, $a_2$ and $a_3$ are non negative constants). In choosing the values for $a_1$, $a_2$ and $a_3$, one has to be aware of the fact that $F_1$, $F_2$, and $F_3$ are not independent of the other.

In this thesis, the objective functions $F_1$ and $F_2$ are used.

### 4.2 Container process as parallel machine problem

In a classical identical parallel machine scheduling problem with the objective of minimising the make-span, all jobs require a single operation and can be processed on anyone of the $m$ identical machines (see Blazewicz et al. (1996) and Brucker (1995)). In the classification of deterministic scheduling, the minimisation of make-span of identical parallel machine problem is denoted as $P_m // C_{\text{max}}$. Figure 4.1 shows the Gantt chart of the initial and the optimal schedule of a $P_2 // C_{\text{max}}$ problem.

![Figure 4.1 P_m // C_{max} (a) Initial schedule (b) Optimal schedule](image)
Many problems relating to container process can be readily adopted as \( P_m \parallel C_{\text{max}} \), though simplification is required. For example, the crane scheduling problem in minimising the unloading time of a single ship, may be modelled as \( P_m \parallel C_{\text{max}} \).

Now consider a ship with only inbound containers and further assume that the unloading time is negligible. The following model minimises the processing time of the ship.

**Parameters and indices**

- \( m \quad 1..M \) yard machines
- \( j, k \quad 1..N \) containers
- \( p_j \) time required to transfer container \( j \) between the marshalling area and the storage yard

**Variables**

- \( X_{mjk} \quad \text{Binary variable representing whether container } j \text{ is the } k\text{-th task processed by machine } m \)

The objective is to minimise the maximum make-span of YMs:

\[
\text{Minimise } \max_m \left( \sum_j \sum_k p_j \times X_{mjk} \right) \quad (4.1)
\]

Constraints 4.2 and 4.3 ensure that each task is performed by one and only one YM:

\[
\sum_m \sum_k X_{mjk} = 1 \quad (4.2)
\]

\[
\sum_j X_{mjk} \leq 1 \quad (4.3)
\]
Even with this simplified approach, $P_m//C_{max}$ is known as NP hard. From a more realistic perspective, container process is a special class of scheduling problem. As shown in Figure 4.2, some jobs flow from sea-side to land-side while other jobs are processed in the reverse direction of flow. Inbound containers must be unloaded before they can be moved to the yard, and outbound containers cannot be loaded until they are moved to the marshalling area. For this reason, the problem may be reformulated as a parallel machine program with release and delivery times.

![Diagram showing container process involving one QC and two YMs](image)

Figure 4.2 Container process involving one QC and two YMs

### 4.3 Parallel machine problem with release times and delivery times

Let $\tau_j$ be the loading/unloading time of the container $j$. In this section, $\tau_j$ is no longer considered negligible. Let $P_j$ be the set of all inbound containers of the same QC preceding $j$; $Q_j$ be the set of all outbound containers of the same QC succeeding $j$. The release time of the container $j$:

$$r_j = \sum_{k \in P_j} \tau_k$$  \hspace{1cm} (4.4)
For an inbound container, the task is complete when it is transferred to the yard by a YM. This is not the case for an outbound container, which requires additional time for loading onto the ship after arriving in the marshalling area. In order to enforce the loading sequence, the delivery time of an outbound container $j$ is taken as the sum of loading time of $j$ and its successor:

$$q_j = \tau_j + \sum_{k \in Q_j} \tau_k$$  \hspace{1cm} (4.5)$$

The container process may be formulated as a parallel machine problem with release times and delivery times ($Pm/r_j,q_j/C_{max}$). This type of scheduling problem is known to be NP-hard in the strong sense (see Garey and Johnson (1979)). However, the Schrage's algorithm can solve the single machine case with a time complexity $O(n \log n)$.

**Algorithm 4.1. Schrage's Algorithm for $1/r_j,q_j/C_{max}$**

Let $p_j$ be the processing time and $q_j$ of task $j \in J$.

**Step 1.** Initialise the sequence of schedule $S \neq \emptyset$

**Step 2.** Let $J$ be the set of all remaining tasks. Set $t = \min_{j \in J}(r_j)$; $C_{max} = t$

**Step 3.** Initialise the sequence of schedule $J \neq \emptyset$

**Step 4.** If $J \neq \emptyset$ then stops

**Step 4.** Obtain the set of tasks ($J'$) with release time not less than $t$

**Step 5.** Obtain the set of tasks $K' \subset J'$ such that

$$K' = \{k \in J' : q_k = \max_h(q_h)\}$$

**Step 6.** Select $j \in K$ such that $p_j = \max_h(p_h)$

**Step 7.** Schedule $j$ at time $t$, and $J = J \setminus \{j\}$

**Step 8.** Set $t = \max\left(t + p_j, \min_{h \in J}(n_h)\right)$

**Step 9.** Set $C_{max} = \max\left(C_{max}, t + p_j + q_j\right)$

**Step 10.** Go to Step 4.
Let $t_j$ be the start processing time of the container $j$. $Pm/r_j,q_j/C_{\text{max}}$ can be formulated as MIP. Since $Pm/r_j,q_j/C_{\text{max}}$ is just a special case of $P_m//C_{\text{max}}$, Constraints 4.2 and 4.3 are still required. The objective is:

$$\text{Minimise } \max_j (t_j + p_j + q_j) \quad (4.6)$$

In addition to, Constraint 4.7 ensures that processing of container $n$ must not start before the release time:

$$t_j \geq r_j \quad (4.7)$$

If $j$, $j'$ are both processed by YM $m$ and $j$ precedes $j'$, Constraint 4.8 enforces such relationship:

$$t_j \geq t_j + p_j - K(2 - X_{mjk} - X_{mjk'}) \quad (4.8)$$

where $K$ is a sufficiently large number.

### 4.4 Resource-constraint project scheduling problem (RCPSP)

In this sub-section, the capacity at the marshalling area is considered to be limited. Suppose that no more than one container can be stored at each loading/unloading position. The transfer of containers to and from the yard must follow the loading/unloading sequence to avoid congestion. Before an inbound container is unloaded to the marshalling, those previously unloaded should have been moved away. Similarly when an outbound container is moved to the marshalling area, the loading of all other containers at the top of the sequence should have started or completed. In scheduling terminology, the tasks have precedence constraints.

In RCPSP, resources can be classified as renewable, non-renewable, and doubly constrained (see Blazewicz et al. (1996)). These resources are considered to be available at any moment within the planning horizon. However, the total usage of any renewable resource (for example, labour) at any particular moment is limited.
For a non-renewable resource, the total usage throughout the whole planning horizon is limited - once it is used up, the resource is no longer available (for example, cash). A doubly constrained resource is one which is limited at any moment as well as throughout the planning horizon.

A project can be viewed as a collection of activities and events (see Elmaghraby (1977)). An activity consumes time and resources. An event is a well-defined occurrence in time. Most activities in project scheduling have precedence constraints because of technological and other considerations. These precedence constraints can be modelled using the activity networks. An activity network is commonly represented by either activity-on-node (AON) or activity-on-arc (AOA). Figure 4.3 shows an activity network (AON) of a container process of one ship:

![Activity Network Diagram](image)

**Figure 4.3** An activity network (AON) of a container process

Based on the formulation of RCPSP, a model for the container process is developed as follows.

**Parameters and indices**
ATA  Actual time of arrival of the ship \( s \)

c  1..C quay cranes

\( n_c^i \)  1.. \( N_c^i \) inbound containers unloaded to the ship by \( c \)

\( n_c^o \)  1.. \( N_c^o \) outbound containers loaded to the ship by \( c \)

t  0.1..T equal time intervals

\( \mu_{c,i} \) (\( \mu_{c,i} \)) Time required for unloading \( n_c^i \) from ship (loading \( n_c^o \) to ship)

\( \lambda_{m,c} \) (\( \lambda_{m,c} \)) Time required by \( m \) in moving \( n_c^i \) (\( n_c^o \)) between the yard and the marshalling area

\( E_{m,c}^i \) (\( L_{m,c}^i \)) Initial calculated earliest (latest) finishing time for \( m \) to complete finishing moving \( n_c^i \) between the yard and the marshalling area.

\( E_{m,c}^o \) (\( L_{m,c}^o \)) Initial calculated earliest (latest) finishing time for \( m \) to complete moving \( n_c^o \) between the yard and the marshalling area.

Note that \( E_{m,c}^i, L_{m,c}^i, E_{m,c}^o, L_{m,c}^o \) are only for limiting the search space. For example, \( n_c^i \) all containers preceding must be unloaded first before moving to the yard:

\[
E_{m,c}^i = ATA + \sum_{n=1}^{n_c^i} \mu_n + \lambda_{m,c}
\]  \hspace{1cm} (4.9)

The ship must finish loading/unloading and leave within the planning horizon:

\[
L_{m,c}^i = T - \left( \sum_{k=i}^{n_c^i} \mu_k + \sum_{k=i}^{n_c^o} \mu_k \right)
\]  \hspace{1cm} (4.10)

Similarly

\[
E_{m,c}^o = ATA + \left( \sum_{n=1}^{N_c^o} \mu_n \right)
\]  \hspace{1cm} (4.11)
\[ L_{mn} = T - \left( \sum_{k=1}^{n^e} \mu_k \right) \]  
(4.12)

**Decision variables**

\[ t_{mn} \quad (t_{mn}) \quad \text{Time at which yard machine } m \text{ completes moving container } n^e \quad (n^e) \]

between the yard and the marshalling area

\[ X_{mn} \quad (X_{mn}) \quad \text{Binary variable representing whether yard machine } m \text{ completes moving container } n^e \quad (n^e) \text{ between the yard and the marshalling area at time } t \]

**Objective function**

The objective is to minimise the ship time:

\[ \text{Minimise } Z = ATD - ATA \]  
(4.13)

**Constraints**

Constraints 4.14 and 4.15 ensure that all containers are moved to either the yard or the marshalling area:

\[ \sum_{m} \sum_{t=1}^{L_{mn}} X_{mn} = 1 \]  
(4.14)

\[ \sum_{m} \sum_{t=1}^{L_{mn}} X_{mn} = 1 \]  
(4.15)
Constraint 4.16 ensures that all outbound containers must be loaded to the ship first before departure

\[
ATD \geq \mu_{n_c}^i + \sum_{m}^{L_{mn}} \max_{t=0}^T \sum_{m}^{L_{mn}} tX_{mn} \quad \text{for all } m, n_c
\]  

(4.16)

For each QC \( c \), Constraint 4.17 ensures that YMs must process all inbound containers before its outbound containers.

\[
\sum_{m}^{L_{mn}} (tX_{mn}) \geq \sum_{m}^{L_{mn}} (t - \lambda_n)X_{mn} \quad \text{for all } n, n' \in N \ (n \text{ precede } n')
\]  

(4.17)

Constraint 4.18 ensures the limit in capacity of YMs as renewable resources:

\[
\sum_{n_c}^{n_c} \left( \min_{t=0}^T \left( t + \lambda_n \right) \sum_{m}^{L_{mn}} X_{mn} \right) + \sum_{n_c}^{n_c} \left( \min_{t=0}^T \left( t + \lambda_n \right) \sum_{m}^{L_{mn}} X_{mn} \right) \leq m \quad \text{for } h=0,1,2,..,T
\]  

(4.18)

However, such formulation is based on a simplified port environment. A new approach is proposed in the next chapter to deal with a more realistic problem that:

1. from a different loading/unloading position, the distance of a storage location in the yard is different

2. loading of a ship may start even when unloading is not complete
Chapter 5

Modelling of the problem with a new approach

In this chapter, a new approach is investigated for a more realistic problem in scheduling of container process. In addition, this chapter also studies how the model is adapted to the reality when machines may not always be available.

5.1 Two-stage flexible flowshop

There are similarities between an import (export) process and a classical two-stage flexible flowshop (see Pinedo (1995)), where each stage (water-side or land-side operation) consists of multiple identical processors. Now consider an import process of a ship involving two QCs, two YMs and 10 containers (each QC unloads 5 containers). Figure 5.1(a) shows an example of import process under no-pooling policy, and Figure 5.1(b) shows a reduction of maximum make-span of an import process by allowing pooling of YMs. The maximum make-span is further reduced when both pooling and sequence flexibility of YMs are allowed as shown in Figure 5.1(c). A more detailed discussion will be presented in Chapter 8 on how sequence flexibility is achieved. The export process has similar characteristics as the two-stage flexible flowshop.

When a ship arrives, inbound containers are already stored on the ship. There is not much flexibility in the unloading sequence because the ship balance must be observed, and containers from the top of a stack must be unloaded first before those from the bottom are taken out. Loading of containers is also not highly flexible if each outbound container is allocated a specific location on the ship for various reasons. However, the loading/unloading can start from bow to stern, or from stern to bow, or unload-load-unload-load, etc. This flexibility adds more complexity to the problem. In order to develop a manageable model, this chapter assumes that the
loading and the unloading sequences are pre-determined; and the storage location of containers on the ship and in the yard is also pre-determined. Furthermore, it is assumed that, at most, one container is allowed at each loading/unloading position.
The Container Process Model (CPM) is proposed in this section for optimising the import and export process of a single berth system. The multi-berth system will be discussed later in Chapter 8.

In CPM, the processing time of an inbound container is the sum of handling time and the transfer time between the unloading position and the location in the yard. The processing time of an outbound container is the sum of handling time and the transfer time between the location in the yard and the loading position. After a YM has completed a task, it needs a setup time for travelling to a new position for the next task (see Figure 5.2). In scheduling terminology, the setup time is said to be sequence-dependent.

In this chapter, CPM has the following assumptions:

1. only one ship is considered;
2. all machines operate at the same speeds;

3. loading/unloading sequence and the storage location in the yard are predetermined;

4. at most one container is allowed at each loading/unloading position

---

**Figure 5.2** Setup and processing of containers by yard machines

**Parameters and indices**

\[ c: \quad \text{Quay crane } c, \ c \in C \]

\[ m: \quad \text{Yard machine } m, \ m \in M \]

\[ K: \quad \text{Sufficient large number} \]
\( j: \) Container \( j, j \in J \)

\( j_c: \) Container \( j \) of loaded/unloaded by \( c, j_c \in J_c \)

\( j_c^*(j_c^*): \) Import (export) containers of ship \( s \) unloaded (loaded), \( j_c^* \in J_c^* \)

\( j_0(j_i): \) Dummy starting (ending) job of nil processing time for all YMs

\( J^+: \) \( J \cup \{j_0, j_i\} \)

\( \tau_{jm}: \) Loading/unloading time of \( j_m \)

\( p_j: \) Processing time of \( j \) where \( j \in J \)

\( \pi_{jm}: \) The setup time required by a YM to process \( j' \in J \) just after processing \( j \in J \)

\( STA(\ STD): \) Scheduled arrival (departure) time of the ship. If the ship arrives early or late, then the scheduled time should be revised.

**Variables**

\( R_{jm}: \) Start loading/unloading time of \( j_m \)

\( ATA(\ ATD): \) Actual arrival (departure) of the ship

\( X_{mj}: \) Binary variable represents whether \( m \) processes \( j \) then \( j' \), where \( j, j' \in J^* \)

\( t_j: \) Start processing time of \( j \in J \)

\( u_j: \) End processing time of \( j \in J \)

**The Model**

The objective is to minimise the weighted penalty on ship service time:

\[
\text{Minimise} \quad (ATD - ATA) \quad \text{(5.1)}
\]

Constraint 5.2 ensures that the ship berths after its scheduled arrival time:
\[ ATA \geq STA \] (5.2)

Constraint 5.3 ensures that loading and unloading of containers can not start until the
ship berths:

\[ R_{j_i} \geq ATA \] (5.3)

Constraint 5.4 ensures that the ship leaves the berth after completion of loading and
unloading of containers:

\[ ATD \geq R_{j_i} + \tau_{j_i} \] (5.4)

Constraints 5.5 - 5.8 ensure that each container is processed by one and only one
YM:

\[ \sum_{j \in J} X_{mjj} = 1 \] (5.5)

\[ \sum_{j \in J} X_{jmi} = 1 \] (5.6)

\[ \sum_{m} \left( \sum_{j \in J} X_{mjj'} \right) = 1 \quad \forall j \in J \] (5.7)

\[ \sum_{j \in J} X_{mj'} - \sum_{j \in J} X_{mj} = 0 \quad \forall j \in J \] (5.8)

Constraint 5.9 ensures that QC loads and unloads containers one after the other:

\[ R_{j} \geq R_{j} + \tau_{j} \quad \forall j, j' \in J_{e}, \text{ where } j \text{ precedes } j' \] (5.9)

Constraint 5.10 ensures that container \( j \) will be unloaded from the ship before
processing by a YM:

\[ t_{j} \geq R_{j} + \tau_{j} \quad \forall j \in J_{e} \] (5.10)

Constraint 5.11 ensures that export containers will be loaded to the ship after they
are moved to the marshalling area:

\[ R_{j} \geq u_{j} \quad \forall j \in J_{e}' \] (5.11)

Constraints 5.12-5.14 ensures that no more than one container at the marshalling
area for each QC:
\[ R_j + \tau_j \geq t_j \quad \forall j, j' \in J^I, \text{ where } j \text{ is unloaded before } j' \quad (5.12) \]

\[ u_j \geq t_j \quad \forall j \in J^I \cap J^C, \text{ where } j \text{ is unloaded before } j' \text{ is loaded} \quad (5.13) \]

\[ u_j \geq R_j \quad \text{ where } j, j' \in J^C \text{ and } j \text{ is loaded before } j' \quad (5.14) \]

Constraints 5.15-5.16 satisfy the starting and finishing time of \( m \)

\[ u_j \geq t_j + p_j \quad (5.15) \]

\[ t_j \geq u_j + \pi_{mjj'} - (1 - X_{mjj'})K \quad \forall j, j' \in J \quad (5.16) \]

The model has been implemented using LINGO 8 (see Appendix A), and a small sample of 8 containers is solved in less than one minute on a Pentium 4 PC. For a sample of 10 containers, however, the optimal solution was not obtained even after 8 hours or running time. Thus, meta-heuristic techniques are developed in Chapter 6 for solving large-scale real-life problem. Numerical investigations will be presented in Chapter 7.

### 5.3 Availability of machines

Yard machines may be unavailable due to breakdown (unplanned event) or maintenance (planned event). If YMs are manually operated, the availability of machines is also linked with the availability of labour. A new schedule is required for the new operation environment - irrespective of whether pooling or no-pooling was originally implemented.

When pooling is allowed in a port, unplanned disruption due to machines being temporarily unavailable can cause serious problems to the container process. Suppose a YM designated to process an outbound container breaks down. This outbound container can not be loaded to the ship, nor can all the successive containers of the same QC. For this reason, ports implementing pooling policy must be able to detect any disruptions and to re-schedule if necessary. YMs and the control tower must be equipped with communications systems for exchanging
information about the change in operation environment, and any updated schedule. An approach for handling unplanned events will be investigated in Chapter 6.

If it is known in advance that some YMs are not available, CPM can be adapted by introducing additional constraints to restrict the assignment of these YMs to the time when they are available. Let \([\theta_m^1, \theta_m^2]\) be the time interval where YM \(m\) is available; and when \(m\) is available, it will be near the container to be processed (so, no setup time is required). Constraints 5.17-18 ensures \(m\) operates within this time interval.

\[
\begin{align*}
t_j &\geq \theta_m^1 \quad \forall j \in J \\
u_j &\leq \theta_m^2 \quad \forall j \in J
\end{align*}
\]

Now consider QC \(c\) is known in advance that it is only available in the time interval \([\theta_c^1, \theta_c^2]\). If handling/transfer time of the yard operations is ignored, a loading/unloading plan can be prepared with \(c\) operating only within the time it is available. Otherwise, the start loading/unloading time can not be pre-determined and the loading/unloading plan is difficult to prepare. Note that the balance of the ship must always be observed. However, it is possible to estimate the lower and the upper limit of the number of containers that \(c\) can load/unload. For the scheduling of container process with limited availability of \(c\), optimal (best solution) can be obtained by investigating a different number of containers loaded/unloaded within the estimated range.

### 5.4 Capacity at the marshalling area

The marshalling area is interfacing the sea and the land-side of the port. In previous chapters, CPM assumes that only one container is allowed at the marshalling area for each QC. During the import process, a QC unloads a container at the marshalling area; and a YM with lifting capability picks up the container from the ground and carries it to the storage yard for stacking. This operation is reversed during the export process. With a limited capacity at the marshalling area, neither the QC nor the YM can drop off any more containers until space is available again. Obviously,
the capacity at the marshalling minimises the blocking due to down-stream operations.

Assume that re-handling is not required in the marshalling area and the operations between QCs and YMs do not interfere with the other, the following extension of CPM allows for either no or limited capacity in the marshalling area.

5.4.1 Marshalling area with limited capacity

Assume that the marshalling area has a non-zero capacity $\delta$ for each QC. Let $\varphi_{c,k}(j)$ be $k$ containers succeeding container $j$ in the loading/unloading sequence of QC $c$. If no such container exists, then $\varphi_{c,k}(j) = 0$. Since the loading/unloading sequence of each $c$ is pre-determined, $\varphi_{c,k}(j)$ is also pre-determined. To ensure that there are no more than $\delta$ containers at the marshalling area for each QC, Constraints 5.12 - 5.14 are replaced by Constraints 5.19 - 5.21:

$$R_j + \tau_j \geq t_{j'} \quad \forall j, j' \in J_{c'} \text{ where } j' = \varphi_{c,\delta}(j)$$ (5.19)

$$u_j \geq t_{j'} \quad \forall j \in N_c, \forall j' \in N_{c'} \text{ where } j' = \varphi_{c,\delta}(j)$$ (5.20)

$$u_j \geq R_j \quad \forall j, j' \in J_{c'} \text{ where } j' = \varphi_{c,\delta}(j)$$ (5.21)

5.4.2 No capacity at the marshalling area

If tractors are used in the yard, they have no lifting capability and the capacity at the marshalling area is considered to be zero. In this case, QCs must unload containers directly to the tractors; or they must pick up containers directly from the tractors for loading. The start processing time $t_j$ of the inbound container $j$ must be the same as the time when the container is unloaded; and if $j$ is an outbound container, the end processing time $u_j$ must be the same as the time when loading starts. Therefore, Constraints 5.12, 5.13 and 5.14 are replaced by Constraints 5.22 and 5.23:
\[ R_j + \tau_j = t_j \quad \forall j, j' \in J^i_c, \text{ where } j \text{ is unloaded before } j' \quad (5.22) \]
\[ u_j = R_j \quad \text{where } j, j' \in J^e_c \text{ and } j \text{ is loaded before } j' \quad (5.23) \]
Chapter 6

Solution methodologies for the container process model

It is possible to reduce CPM to a parallel machine problem, which is known to be NP hard. On the other hand, real-life problems often involve thousands of containers. For these reasons, exact solution techniques (for example, brand-and-bound) do not offer any hope in solving for the optimal solution of the container process problem within reasonable time. This chapter explores various solution methodologies that may be applicable for near optimal solutions of the problem.

6.1 Heuristic algorithms

In scheduling terminology, a schedule has regular performance measure if the objective function is non-decreasing with the completion time for each job. Examples of regular performance measures are makespan ($C_{\text{max}}$), maximum lateness, total weighted tardiness. If a new schedule can decrease the completion time of a job, the objective should not increase.

A feasible schedule can be active, semi-active, or no-delay. For a semi-active schedule, no operation can be completed earlier without changing the sequence of operations of machines (see Blazewicz et al. (1996), French (1982)). An active schedule is one that no operation can be completed earlier by changing the sequence of operations of machines and without delaying any other operations. A no-delay schedule is one where no machine is idle when there is an operation available for processing. In general, a no-delay schedule must be active; and an active schedule must also be semi-active. For scheduling problems with regular performance measure, the set of optimal schedules contains at least an active schedule. In other words, an optimal schedule can be found by searching through the set of all active...
schedules. Giffler and Thompson (1960) suggest an algorithm for generating all active schedules. The idea is to assign available tasks to machines. If several tasks may be processed on the same machine, the set of these tasks is called conflict set.

### 6.1.1 Priority rule

Consider a scheduling problem with regular performance measure and the process time is not sequence-dependent. Let \( J = \{j_1, j_2, ..., j_n\} \) be the set of tasks; \( P \) be the partial schedule, \( Q \) is the set of unscheduled tasks. For each task \( e_i \), let the process time be \( p_i \), release time be \( r_i \), and completion time be \( C_i \).

#### Algorithm 6.1 Giffler and Thompson Algorithm

1. **Step 1.** Initialise \( P = \emptyset \); \( Q = J \)
2. **Step 2.** If \( Q = \emptyset \), stops
3. **Step 3.** Select \( j^* \in Q(t) \) such that \( j^* \) has the shortest complete time \( C^* \); \( m^* \) be the machine on which \( j^* \) occurs; \( C^* \) be the complete time of \( j^* \).
4. **Step 4.** Let \( V \) be the conflict set - set of unscheduled tasks that must be processed by \( m^* \) and with release time less than \( C^* \). Select \( j_k \) from \( V \) randomly and add \( j_k \) to the partial schedule \( P \)
5. **Step 5.** Assign \( Q(t) = Q(t) \setminus \{j_k\} \)
6. **Step 6.** Return to Step 2.

Giffler and Thompson (1960) also suggest the use of priority index to resolve conflicts. Examples of such a priority rule are shortest task time, longest task remaining processing time (see Blazewicz et al. (1996), Baker (1974)).

### 6.1.2 List scheduling

Another popular heuristic technique is list scheduling (see Blazewicz et al. (1996)) for constructing a schedule for parallel machines. According to a given sequence of tasks, each task is assigned to the first available machine. If there is a tie, the machine with the smallest index is selected. The list constructed according to
longest processing time (LPT) may give good results. Let \( U = \{e_1, e_2, \ldots, e_n\} \) be a sequence of tasks; \( r_m \) be the next available time of machine \( m \).

**Algorithm 6.2. List scheduling**

*Step 1.* Initialise \( Q = U \); Initialise \( r_m = 0 \) for all \( m \in M \)

*Step 2.* If \( Q = \emptyset \), stop

*Step 3.* Select the first task \( e_i \) from \( Q \)

*Step 4.* Select \( m^* \in M \) such that \( r_{m^*} = \min_{m \in M} r_m \)

*Step 5.* Schedule \( e_i \) to be processed by \( m^* \)

*Step 6.* Update \( r_{m^*} \)

*Step 7.* Remove \( e_i \) from \( Q \), \( Q = Q \setminus \{e_i\} \)

*Step 8.* Return to step 2.

When applying to CPM, the next job \( e_i \) selected in Step 2 must have no precedence in \( Q \). Otherwise, no feasible solution can be generated using the list sequence (see 6.2 for more details).

However, Ovacik and Uzsoy (1993) show that list schedule may not be able to generate an optimal schedule for parallel machine problem with sequence-dependent setup times. This limitation of list scheduling applies to CPM, because CPM can be treated as a scheduling problem with no processing times but only sequence-dependent setup times. Table 6.1 shows the setup times of three tasks 1, 2, and 3 by two yard machines \( m_1 \) and \( m_2 \) of no processing time. Applying list scheduling algorithms, the best schedule that can be generated has a makespan of 11 using the sequence \( 1,2,3 \) (see Figure 6.1(a)). Schutten (1996) shows that an optimal schedule can be generated using an alternative list scheduling algorithm by assigning the next job to the machine on which the job will complete first. This results in an optimal schedule with a makespan of 5 as shown in see Figure 6.1(b).
### Table 6.1 Setup times of yard machines

<table>
<thead>
<tr>
<th>From</th>
<th>To 1</th>
<th>To 2</th>
<th>To 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Since CPM concerns loading and unloading times, Schutten algorithms on outbound containers must be adapted to consider either the end-processing time (when a container is transferred to the marshalling area), or the complete time (when a container is loaded to the ship).

#### 6.2 Deadlocks and violation of precedence constraints

Pooling of YMs brings along efficiency to yard operations. However, a sudden change in environment (for example, a change in loading/unloading plan in response to change in weather condition) may result in deadlock violation of precedence constraints from the original feasible schedule. For manually operated machines, the
drivers may be able to recognise and resolve the issues with concerted effort. For automated machines at container terminals, they rely on the control software for the detection and resolution of such issues.

Lehmann et al. (2007) suggest a matrix-based and a graph-oriented deadlock detection scheme for AGVs. To resolve any deadlock detected, they propose algorithms for resequencing the transportation order. Kim et al. (2007) assume that the travelling area of AGVs is divided into grid-blocks, and develop algorithms for deadlock-free reservation scheduling.

Consider YM $m_1$ and $m_2$; QC $c_1$ and $c_2$; container $j_{1,1}$, $j_{1,2}$ are unloaded from $c_1$; and $j_{2,1}$, $j_{2,2}$ are unloaded from $c_2$. Recall the assumption that only one container is allowed at each loading/unloading position, the schedule with the following processing sequence violates the precedence constraints.

$$SC1$$

$m_1 : \{ j_{1,2}, j_{1,1} \}$

$m_2 : \{ j_{2,2}, j_{2,1} \}$

Some schedules may be infeasible due to the occurrence of circular-wait, even though no individual YM shows any violation of precedence constraints. Consider the following processing sequences for $m_1$ and $m_2$:

$$SC2$$

$m_1 : \{ j_{1,2}, j_{2,1} \}$

$m_2 : \{ j_{2,2}, j_{1,1} \}$

A deadlock occurs and all processes stop - $m_1$ can not process $j_{1,2}$ because it is not unloaded yet, and $m_2$ can not process $j_{2,2}$ because it is also not unloaded yet. In other words, the processing sequences will result in an infeasible solution.
When deadlock or violation of precedence-constraints occurs, there is always a task ready to be processed from the list of remaining tasks. It is possible to scan through the remaining sequence of each YM to identify which task is ready to be processed by that YM. However, this is time consuming if the sequences are long. Instead, it is quicker to scan through from the list of preceding tasks of QC and to identify a task which is ready but have not yet processed. In the example SC1 above, the task preceding \( j_{1,2} \) is \( j_{1,1} \) and the task preceding \( j_{2,2} \) is \( j_{2,1} \). By swapping the corresponding tasks, the final process sequences become:

\[
SC1^* \\
\quad m_1 : \{ j_{1,1}, j_{1,2} \} \\
\quad m_2 : \{ j_{2,1}, j_{2,2} \}
\]

Similarly for SC2, the task \( j_{1,2} \) is swapped with \( j_{1,1} \). The final process sequences become:

\[
SC2^* \\
\quad m_1 : \{ j_{1,1}, j_{2,1} \} \\
\quad m_2 : \{ j_{2,2}, j_{1,2} \}
\]

It is not sure whether the new schedule performs better or not, but at least all YMs can proceed to complete all tasks. When list scheduling algorithm is applied to generate schedules, deadlock situation is not apparent because the assignment of tasks is a "linear process" and the algorithm comes to a halt before a circular wait situation occurs.

The following list sequences will result in infeasible schedules:

\[
Q_1 : \{ j_{1,2}, j_{1,1}, j_{2,2}, j_{2,1} \} \\
Q_2 : \{ j_{1,2}, j_{2,2}, j_{1,1}, j_{2,1} \}
\]
When applying *List Scheduling Algorithms*, both $Q_1$ and $Q_2$ will result with a halt at *Step 3* without completing all tasks. No YM can be assigned to the first task of either list because of the precedence-constraints. To resolve this issue, a simple strategy is to allow $j_{1,1}$ to jump the queue by swapping. The new list sequence for $Q_1$ becomes:

$$Q_1^* : \{ j_{1,1}, j_{1,2}, j_{2,2}, j_{2,1} \}$$

and the final feasible list sequence is:

$$Q_1^* : \{ j_{1,1}, j_{1,2}, j_{2,1}, j_{2,2} \}$$

Similarly, the final feasible list sequence for $Q_2$ is:

$$Q_2^* : \{ j_{1,1}, j_{1,2}, j_{2,1}, j_{2,2} \}$$

In general, it is always possible to find a task with no precedence from the remaining non-empty list sequence. To avoid deadlocks and violation of precedence-constraints, *Step 3* of Algorithm 6.2 is replaced by the following:

*Step 3a. Select the first task $e_k$ from $Q$ with no precedence*

Note that deadlock and violation of precedence-constraints may not occur if two or more containers are allowed at each loading/unloading position. In the two examples above, $m_1$ and $m_2$ only need to wait for the second containers to be unloaded in order to follow the designated process sequences.

### 6.3 Meta-heuristics for solving large-scale problems

For solving large scale complex problems, meta-heuristics solution techniques are gaining more popularity. When finding an exact solution is almost impossible,
meta-heuristic techniques are designed to obtain the "best" solution within reasonable time. The most common meta-heuristics are Tabu Search (TS), Simulated Annealing (SA) and Genetic Algorithms (GA).

6.3.1 Local Search

Hill-climbing, Tabu Search, Simulated Annealing are classified as Local Search. Local Search in general has learning capabilities designed to deal with NP-hard problems. During the optimization process, the knowledge of the problem is exploited for the speed and quality of the process. Local search strategies normally start from an initial feasible solution, and iteratively search for improved solution by slight perturbation of the solution. For a solution \( X \), the set of small perturbations of \( X \) is often called the **neighbourhood** of \( x, N(x) \). The best neighbour of \( X \) is the solution in \( N(X) \) with the best objective function.

As an example, the following describes the Hill-climbing algorithms for minimising an objective function \( f \): 

Algorithm 6.3 Hill-climbing

1. Start with an initial feasible solution \( X_0 \) and set \( X_{\text{best}} \leftarrow X_0 \)
2. Generate the neighbourhood of \( N(X_{\text{best}}) \)
3. Obtain the best neighbour \( X \) in \( N(X_{\text{best}}) \)
4. If \( f(X) < f(X_{\text{best}}) \) then set \( X_{\text{best}} \leftarrow X \)
5. Repeat Step 2 until the stopping condition is met.

The stopping condition is normally defined as a predefined number of iterations; a predefined running time; or no more improvement in the solution after a predefined number of iterations. For a large-scale system, searching for the best neighbour in each move may be too time-consuming. In this case, only small samples of the neighbourhood \( N(X) \) are selected randomly.
Hill-climbing algorithms are much easier to implement than the other meta-heuristics. However, there is a danger for the system being trapped in a local minimum (see Figure 6.2). Tabu Search and Simulated Annealing are similar to local search in many ways, but these frameworks are designed to escape from the local minima.

![Figure 6.2 Solution space with multiple local minima](image)

### 6.4 Tabu Search

TS is a framework originally proposed by Glover designed to guide deterministically the local search process out of local optima. The strength of TS is its use of adaptive memory for creating a more flexible search (see Blazewicz et al. (1996), Glover and Laguna (1997) and Papadimitriou and Steiglitz (1998) for details). During the search, TS builds a list of previously visited neighbours. This list of pre-defined maximum size allows TS to avoid cycling through the same neighbourhood around a local minimum. Unlike the Hill-climbing strategies, it is also possible for TS to travel through non-improving moves to escape from a local minimum.

The following algorithms present a simplified version of Tabu Search for minimising an object function $f$:

**Algorithm 6.4 Tabu Search**

1. **Define a positive but decreasing function $g$ and an empty Tabu List $L$.**
2. Set counter \( n = 0 \), an initial feasible solution \( X_0 \), the current solution as \( X' = X_0 \) and the best solution \( X_{\text{best}} = X' \).

3. Generate the neighbourhood of \( N(X') \).

4. Obtain the best neighbour \( X \) in \( N(X') \).

5. \( \Delta = f(X) - f(X') \).

6. If \( \Delta < 0 \), set \( X_{\text{best}} = X \).

7. If \( \Delta < g(n) \) or \( X' \) is not in \( L \), set \( X' = X \).

8. Increase \( n \) by 1 and update \( L \).

9. Repeat Step 3 until the stopping condition is met.

The length of the Tabu list can affect the performance of TS. If the length of the list is too short, the search may not be able to "remember" that certain solutions have already visited. If the length is too long, the search may not be able to revisit certain solutions. Initial trial runs are often used to determine the best size of Tabu List.

For solving CPM, the following algorithm is designed for searching the best neighbourhood of an existing feasible schedule (\( \mathcal{J} \)) and is denoted as TS-PM:

**Algorithm 6.5**

1. Randomly select a container \( j \) (assigned to \( m_1 \)).

2. For any container \( j' \) (\( j \neq j' \)) assigned to \( m_1 \), create a new schedule \( \mathcal{J}' \) by swapping \( j \) and \( j' \) (see Figure 6.3(a)).

3. For any container \( j' \) (\( j \neq j' \)) assigned to \( m_1 \), create a new schedule \( \mathcal{J}' \) by inserting \( j \) before \( j' \) (see Figure 6.3(b)).

4. For any \( m_2 \) (\( m_1 \neq m_2 \)) and any container \( j' \) assigned to \( m_2 \), create a new schedule \( \mathcal{J}' \) by swapping \( j \) and \( j' \) (see Figure 6.3(c)).

5. For any \( m_2 \) (\( m_1 \neq m_2 \)) and any container \( j' \) assigned to \( m_2 \), create a new schedule \( \mathcal{J}' \) by inserting \( j \) before \( j' \) (see Figure 6.3(d)).
The neighbourhood search can be greatly decreased by first checking precedence-constraints. However, the resulting new neighbour may be subject to deadlock and violation of precedence constraints in the case of multiple-YMs. For such an infeasible schedule, the objective may be assigned with a sufficient large penalty. An alternative is to resolve any deadlocks or violation of precedence-constraints similar to the method for list scheduling. When list sequence is used for generating schedules, the TS implementation (denoted by TS_LIS) is simpler in swapping and insertion as there is one sequence.

### 6.4.1 Simulated Annealing

Simulated Annealing (SA) is another local search framework based on an analogy from thermodynamics. SA starts with a high temperature, and the temperature gradually decreases. A higher temperature represents greater probability of accepting a move for non-improving solution. Eventually, a non-improving solution will be unlikely to be accepted. The following is an example of SA for minimising an objective function $f$:

**Algorithm 6.6 Simulated Annealing**

*Step 1. Select a number $\alpha$ slightly smaller than 1; and the initial temperature $T_0$*
Step 2. Set the temperature \( T = T_0 \); the initial feasible solution as \( X_0 \); the current solution as \( X' = X_0 \) and the best solution \( X_{\text{best}} = X' \)

Step 3. Generate the neighbourhood of \( N(X') \)

Step 4. Obtain the best neighbour \( X \) in \( N(X') \)

Step 5. Set \( \Delta = f(X) - f(X') \)

Step 6. If \( \Delta < 0 \), set \( X_{\text{best}} = X \)

Step 7. If \( \text{random}[0, 1] \leq \exp \left( -\frac{\Delta}{T} \right) \), set \( X' = X \)

Step 8. Decrease the temperature by \( T = \alpha T \)

Step 9. Repeat Step 3 until the stopping condition is met.

The implementation of SA is very similar to the implementation of TS. Although the two solution techniques apply different strategies in escaping from local minimum, they both apply neighbourhood search for the next move. Trial runs are often required to find the best cooling scheme. SA-PM is for schedules generated by sequences on multiple YMs; and SA-LIS is for schedules generated by list scheduling algorithms.

6.4.2 Genetic Algorithms

In natural evolution, a particular combination of genetic structure that contributes more to the gene pool in the next generation is said to have a higher fitness value. In the course of natural selection, those parents with higher fitness values will survive to form the majority to produce the next generation. A greater percentage of the next generation will have higher fitness values. In addition, mutations cause the chromosomes of the children to be different from their parents. Genetic algorithms imitate the process observed in natural evolution. It is a powerful technique for finding near-optimal solution when the conventional approach fails to find the exact solution. In applying GA, a pool of chromosomes as initial population is randomly generated. New chromosomes are then repeatedly created by cross-over and mutation until stop criteria are met. For the initial population and also new chromosomes subsequently created, the corresponding objective value of each
A particular combination of genetic structure that contributes more to the gene pool in the next generation is said to have a higher fitness value. In the course of natural selection, those parents with higher fitness values will survive to form the majority and to produce the next generation. A greater percentage of the next generation will have higher fitness values. In addition, mutations cause the chromosomes of the children to be different from their parents. Genetic algorithms imitate the process observed in natural evolution. It is a powerful technique for finding near-optimal solution when the conventional approach fails to find the exact solution. In applying GA, a pool of chromosomes as the initial population is randomly generated. New chromosomes are then repeatedly created by cross-over and mutation until stop criteria are met. For the initial population and also new
chromosomes subsequently created, the corresponding objective value of each chromosome is translated into a fitness value and the pool is updated. Reeves (1995) applies GA to solve flow-shop sequencing problem. Varela et al. (2003) apply GA to solve job-shop scheduling problem. Chew and Poh (1999) applies GA to solve RCPSP. In Dorndorf and Pesch (1995) GA is used to solve job shop scheduling problem. It is reported that the results obtained are better than those using shifting bottleneck heuristics or those using SA. In Heilmann (2003), GA is applied to solve the multi-mode resourced constrained project scheduling problem. Kozan and Preston (1999), Bortfeldt and Gehring (2001). Nishimura et al. (2001b) solve problems in a port system using GA.

**Selection of parents for the next generation**

The roulette wheel method is one for the selection of parents. Let \( f_i \) be the fitness of a chromosome \( i \), which should be decreasing with the objective in CPM. For each chromosome in the pool of chromosome, a probability \( P_i \) is computed as follows:

\[
P_i = \frac{f_i}{\sum_j f_j}
\]

Parents are then randomly selected based on this probability.

**Cross-over**

The design of the cross-over operator allows new children to inherit partly from one parent and partly from the other parent. One common approach is the one-point cross-over method. Two parents \( P_1 \) and \( P_2 \) are selected from a pool of chromosomes using the roulette wheel method, and a common cross-over position is randomly generated. As shown in the new children \( C_i \ (C_j) \) is filled with items from \( P_1 \ (P_2) \) up
to just before cross-over position. The remaining items are filled with items in the same order as those in $P_2$ ($P_1$).

![Diagram of cross-over process]

**Figure 6.5** New generations from cross-over

If the two-point cross-over method is adopted, two common cross-over points are randomly generated. $C_1$ ($C_2$) is filled with items from $P_1$ ($P_2$) up to the first cross-over position, and from the second cross-over position. The remaining items are filled in between the two cross-over positions in the same order as the $P_2$ ($P_1$).

Similarly, the number of cross-over points can be extended to 3, 4, ..., etc.

**Mutation**

Mutation operator enables a change in the chromosomes. With *scrambling sub-list*, two positions are selected randomly from a parent. A child is generated by copying all items from the parent, but with all items in-between the two selected positions randomly permuted (see Figure 6.6).

![Diagram of scrambling sub-list process]

**Figure 6.6** New generation from scrambling sub-list
Chromosome representation

For the application of GA to CPM, a linear representation is needed for the process sequence of multiple YMs. With such linear representation for different schedules as chromosomes, new chromosomes can be generated from mutation and/or crossover. Figure 6.7 shows a common form of chromosome representation of the process sequence of three YMs, where dummy tasks "*" are added in-between sequences to represent the "boundaries" between machines (see Cheng et al. (1995), Nishimura et al. (2001a)). GA using this representation is denoted as GA-PM in this study.

![Chromosome representation with machine boundaries](image)

Figure 6.7 Chromosome representation with machine boundaries

In searching for a better schedule, one would expect to try a combination with a YM taking on an additional task from another YM. This means a particular dummy task "*" is to shift left or right. Such shifting of dummy task may heavily violate the precedence-constraints unless more sophisticated mutation is in place (see example as shown in Figure 6.8).

![New chromosome from shifting dummy job](image)

Figure 6.8 New chromosome from shifting dummy job
When schedules are generated by list scheduling, chromosomes can be presented by a list sequences (denoted by GA-LIS). The implementation of cross-over and mutation are straight-forward.

In both GA-PM and GA-LIS, the control of new schedule generation is not that easy as in TS or SA because each operation involves more tasks. For this reason, each operation would have much higher probability in generating new chromosomes of infeasible schedules due to deadlock and violation of precedence-constraints. Therefore GA-LIS must convert chromosomes of infeasible schedules into those of feasible schedules.

### 6.5 Reactive scheduling approach

Planned events relating to machines not being available have been dealt with in the previous chapter. For unplanned events like machine break down, reactive scheduling is proposed in this study to minimise the impact due to the disruptions. Consider the following feasible schedule:

**SC3**

\[ m_1 : \{ j_{i,1}, j_{i,2} \} \]

\[ m_2 : \{ j_{i,1}, j_{i,2} \} \]

If \( m_1 \) breaks-down just after the ship is berthed, appending the list of all tasks originally assigned to \( m_1 \) to the end of task list of \( m_2 \) is not a workable solution.

**SC4**

\[ m_2 : \{ j_{i,1}, j_{i,2}, j_{i,1}, j_{i,2} \} \]

**SC4** can not proceed before its first task because of precedence constraints. In addition, it takes time to generate a new good solution with those remaining tasks. As meta-heuristic algorithms are used, there is a trade-off between the execution
time and the quality of the solution. For these reasons, the following rescheduling strategy is proposed:

Algorithm 6.7

Step 1: Apply list scheduling for a feasible schedule \( SC' \) for the remaining tasks and communicate the revised schedule to YMs.

Step 2: Estimate the time \( t_{resched} \) required to generate an improved schedule.

Step 3: Find the set of all remaining tasks \( J_{resched} \) after the time \( t_{resched} \) according to \( SC' \).

Step 4: Apply meta-heuristic algorithms for \( J_{resched} \) and allow an execution time not exceeding \( t_{resched} \). Communicate the new schedule \( SC^* \) to YMs.
Chapter 7

Numerical investigations

This chapter investigates the performance of the list scheduling heuristics and the meta-heuristic techniques TS, SA, GA as described in the previous chapter. Various parameters that may affect the efficiency of the port are also analysed.

7.1 Implementation of CPM and validation

CPM has been implemented using LINGO 8 (see Appendix A), which employs branch-and-bound methods in their global solver. While LINGO may only be able to solve the problems of smaller size, the results from LINGO are exact solutions. On the hand, the meta-heuristic techniques developed in the previous chapter are designed to obtain the “best solutions” with problems of much larger size. These meta-heuristics techniques are implemented using C++ (see Appendix B) because C++ is capable of generating efficient computer program. Additional C++ codes are developed (see Appendix C) to ensure that the solutions obtained satisfy all constraints of CPM.

Using a small test sample of eight containers, both LINGO and the C++ implementation give the same optimal values of 7.7 minutes.

7.2 Construction of trial data set

The trial data set used for the solution and subsequent analysis is constructed using the current practice in storage and loading/unloading at the ports as far as possible.
7.2.1 Loading/unloading times

The loading/unloading time of a container depends on its location on the ship. To unload Container \( j \) a trolley travels upwards from the marshalling area for a distance \( d_{y_1} \), outwards for a distance of \( d_x \) and then downwards for a distance \( d_{y_2} \). On its return, the trolley travels on the same route with Container \( j \). The speed of trolley without load \( (v_1) \) is taken as 180 m per minutes, and the speed with load \( (v_2) \) is taken to be 90 m per minute. The unloading time is:

\[
\tau_j = \frac{(d_x + d_{y_1} + d_{y_2})}{(v_1 + v_2)}
\]

The same formula applies when loading a container on to the ship.

![Diagram](image)

**Figure 7.1** Movement of trolley when unloading a container

A ship hull is not in the form of a regular shape like a box, and each ship bay does not in general store the same number of containers (see Figure 7.2). Also different ships have different layouts for container storage. For the purpose of analysis,
however, the containers are assumed to be stored starting from the first ship bay; 36 containers are stored are uniformly stacked on four levels in each bay.

Figure 7.2  Screenshot from PowerStow (courtesy Navis)

7.2.2 Transfer and handling times in the storage yard

In the storage yard, assume containers are arranged in rows perpendicular to the quay-line. Each row has six bays and containers are stacked uniformly to three levels high. The first row is at the same position of the first bay on the ship along the quay-line. Containers are stored at least 40 m from the marshalling area. To move a container from to and from the storage yard, the speed of each YM ($v_{YM}$) is taken as 200 m per minute (12 km per hour). Again for the purpose of analysis, each YM will only travel either along or perpendicular to the quay-line.

Let $e_x$ ($e_y$) be the distance a YM travels along (perpendicular) the quay-line. The time to move the container from the marshalling area to the storage location in the yard is $\frac{(e_x + e_y)}{v_{YM}}$. Apart from the transfer time, there is also the pick-up and drop-off time of typically 0.33 min each.
7.3 Numerical investigations

Table 7.1 shows the results from the trial data set generated. The objectives are found from C++ implementation using a Pentium 4 PC of 2.8MHz and 512M RAM. After initial trial runs and fine-tuning of parameters, the meta-heuristics are set to stop after 20,000 iterations or after 2000 iterations without improvement in the objective values. TS-PM shows good performance. However, LIS is also worth considering as this heuristic algorithm gives the solution very quickly.
Table 7.1 Results from sample data set

<table>
<thead>
<tr>
<th>Prob.</th>
<th>Containers</th>
<th>QCs</th>
<th>YMs</th>
<th>Objective values (min)</th>
<th>(% improvement over no-pooling policy )</th>
<th>CPU (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>In</td>
<td>Out</td>
<td></td>
<td>No-pooling LIS</td>
<td>TS-PM SA-PM GA-PM GA-LIS</td>
<td>TS-PM SA-PM GA-PM GA-LIS</td>
</tr>
<tr>
<td>1</td>
<td>180</td>
<td>360</td>
<td>2</td>
<td>2</td>
<td>505</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>2</td>
<td>270</td>
<td>270</td>
<td>2</td>
<td>2</td>
<td>500</td>
<td>0 0 1 2</td>
</tr>
<tr>
<td>3</td>
<td>360</td>
<td>180</td>
<td>2</td>
<td>2</td>
<td>511</td>
<td>2 1 1 2</td>
</tr>
<tr>
<td>4</td>
<td>270</td>
<td>270</td>
<td>2</td>
<td>2</td>
<td>564</td>
<td>33 1 54 11</td>
</tr>
<tr>
<td>5</td>
<td>540</td>
<td>540</td>
<td>2</td>
<td>2</td>
<td>1123</td>
<td>2 0 2 2</td>
</tr>
<tr>
<td>6</td>
<td>540</td>
<td>540</td>
<td>2</td>
<td>4</td>
<td>925</td>
<td>2 1 1 2</td>
</tr>
<tr>
<td>7</td>
<td>540</td>
<td>540</td>
<td>3</td>
<td>3</td>
<td>744</td>
<td>8 3 7 6</td>
</tr>
<tr>
<td>8</td>
<td>540</td>
<td>540</td>
<td>3</td>
<td>6</td>
<td>619</td>
<td>12 0 27 7</td>
</tr>
<tr>
<td>9</td>
<td>1080</td>
<td>1080</td>
<td>3</td>
<td>3</td>
<td>1569</td>
<td>20 3 27 9</td>
</tr>
<tr>
<td>10</td>
<td>1080</td>
<td>1080</td>
<td>3</td>
<td>3</td>
<td>1201</td>
<td>11 1 92 11</td>
</tr>
</tbody>
</table>

Note: Minimum objective values printed in bold

- The yard is quarter-full
In the last quarter of 2004, Port of Brisbane, Australia, handled about 134,000 containers from 227 ships. With this information, the benchmark problem is chosen to be one with 270 inbound and 270 outbound containers. Figure 7.1 shows the effect of the number of QC\(s\) and YM\(s\) on ship service time when the yard is half full. It can be seen that an increase in the number of QC\(s\) must also come with a corresponding increase in the number of YM\(s\), and vice-versa. Otherwise, the bottleneck would only shift from one stage of operation to the other stage without any benefit in minimising the ship service time. In operation planning, CPM can be a useful tool for allocating the minimum resources (QC\(s\) and YM\(s\)) in fulfilling the service requirement of a ship.

![Graph showing the effect of the number of QC\(s\) and YM\(s\) on ship service time.](image)

**Figure 7.3** Effect of the number of QC\(s\) and YM\(s\) when the yard is half full

### 7.4 Yard layout

When straddle-carryers, fork-lifts, or reach-stackers are used in the yard, there must be a gap between rows of containers to allow the machines to move freely along each row. These rows can be arranged along-side or perpendicular to the quay line.
(see Figure 7.4). The selection of a particular yard layout must take into consideration how easy it is for a YM to access each container, and whether a loaded YM can turn easily. On the other hand, the layout of the yard also affects the efficiency of the container process. If each row has only a small number of bays, more rows are needed for a ship. When the rows are arranged perpendicular to the quay line, YMs enter into each row from the near-end of the row. When rows are arranged along side the quay line, YMs may take a longer time to reach either end of a row. However, the YM may also select entry point to the row that is closer to the destination.

![Diagram of container yard layout](image)

**Figure 7.4** Rows of containers (a) perpendicular to and (b) along-side the quay line

Applying CPM, it is possible to analyse the performance of the port with different yard layouts. Assuming no traffic congestion, Figure 7.5 shows the relationship between the ship service time and the number of bays per row for the benchmark problem. The results indicate that it takes a shorter time to process the ship when the rows are perpendicular to the quay line, and each row has about 10 to 14 bays. Note that the length of the ship will also have an influence on the best number of bays per row and can be further analysed by applying CPM. Future research is recommended on the efficiency of the port in relation to the layout of the yard and the length of the ship.
In dealing with unplanned disruptions, list-scheduling heuristic algorithm is a quick way to generate a new feasible schedule. On the other hand, Algorithm 6.7 proposed in the previous chapter aims to improve the quality of the new schedule. With the benchmark problem and assuming that the yard is original half full, Table 7.2 shows the ship service time from different algorithms when YMs are available at all times.

\begin{table}[h]
\centering
\begin{tabular}{|l|c|c|c|}
\hline
YM\textsuperscript{s} available & Ship service time (min) \\
\textit{at all time} & No pooling & List heuristics & TS \\
\hline
2 & 691 & 540 & 505 \\
3 & 491 & 481 & 455 \\
\hline
\end{tabular}
\caption{Ship service time of benchmark problem with yard originally half full}
\end{table}

\section{Reactive scheduling approach}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure7_5.png}
\caption{Ship service time against the number of bays per row in the yard for the benchmark problem}
\end{figure}

In dealing with unplanned disruptions, list-scheduling heuristic algorithm is a quick way to generate a new feasible schedule. On the other hand, Algorithm 6.7 proposed in the previous chapter aims to improve the quality of the new schedule. With the benchmark problem and assuming that the yard is original half full, Table 7.2 shows the ship service time from different algorithms when YMs are available at all times.
Suppose 3 YMs are originally assigned to service the ship and best schedules are generated using TS. However, one of the YMs becomes unavailable due to breakdown. To reschedule according to Algorithm 6.7, 30 minutes are required for $t_{resched}$.

Obviously, the disruption will be more significant if the breakdown occurs near the start of the container process. Table 7.3 shows the ship service time under the following situations:

- $Z_1$: Best ship service time (min) if the event is known in advance;
- $Z_2$: Ship service time (min) from list scheduling heuristics when machine break down occurs;
- $Z_3$: Best ship service time (min) from rescheduling when machine break down occurs.

<table>
<thead>
<tr>
<th>$t$</th>
<th>$Z_1$</th>
<th>$Z_2$</th>
<th>$Z_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>490</td>
<td>528</td>
<td>490</td>
</tr>
<tr>
<td>200</td>
<td>487</td>
<td>503</td>
<td>487</td>
</tr>
</tbody>
</table>

From the benchmark problem, the rescheduling of the container process gives the same results as if the breakdown of the machine is known in advance.

### 7.6 Capacity at the marshalling area

A QC can reach five to six containers on the land-side. Having this area acting as an intermediate buffer will minimise the blocking from down-stream operation. Figure
7.6 shows the effect of the capacity for each QC in the marshalling area on the ship service time on the benchmark problem. When only two YMs are in use, a small capacity does improve the efficiency of the container process. However, this improvement becomes less significant when the capacity reaches three containers. In fact, having too many containers in the marshalling area may lead to stacking of containers, which in turn increases the likelihood of rehandling. Extra-care is also needed to avoid the interference and even collision between QCs and YMs. On the other hand, the advantage of having capacity in the marshalling area becomes negligible with four YMs in the yard.

As the cost structure is different for different types of yard equipment, the extension of CPM to no, or limited, capacity in the marshalling area provides a decision tool for yard operators in investment appraisal.
Chapter 8

Container Process Model in a multi-berth and multi-ship environment

Due to the complexity of the container process at seaport terminals, the previous chapters were focused on addressing the single-ship case with pre-defined storage location. In this chapter, the CPM is extended to a multi-ship and multi-berth environment. A model is also developed for the improvement of the solution by iterating through the refinement of storage locations and re-scheduling.

8.1 Multi-berth and multi-ship environment

In a multi-berth and multi-ship environment, two or more ships can be berthed side-by-side, or one after the other. If two or more ships berth side-by-side, it is possible to have the pooling of yard machines for these ships to increase the efficiency of the container process. Consider two ships $s_1$ and $s_2$ of typical length 250m each, and separated by 30m. QC working on near-ends of $s_1$ and $s_2$ may be closer than QC working on far-ends of the same ship (see Figure 8.1). The scheduling of two ships berthing side-by-side is, to a certain degree, similar to the case when the two ships join together to form a bigger ship. Having more berths available would obviously allow more ships to be processed at the same time. However, there will need to be corresponding investment in QC, YM and storage space so that ships can be loaded and unloaded quickly.
If two ships berth one after the other, loading/unloading of the second ship cannot start until the first ship has left. On the other hand, these two ships have less demand on QCs, YMs and yard storage. After the first ship has left, the storage area initially occupied by its out bound containers will become available for the inbound containers of the second ship. As a result, the second ship may be assigned with storage space closer to the berth.

The following is for the extension of CPM to multi-berth and multi-ship environment:

**Parameters and indices**

- $b$: Berth $b$, $b \in B$
- $s$: Ship $s$, $s \in S$
- $s_b$: Ship $s$ at berth $b$, $s_b \in S_b$
- $c$: Quay cranes $c$, $c \in C$
- $m$: Yard machine $m$, $m \in M$
- $K$: Sufficient large number
- $j$: Container $j$, $j \in J$
- $j_{sc}$: Container $j$ of ship $s$ loaded/unloaded by $c$, $j_{sc} \in J_{sc}$
Variables

\( J_{sc}^i \) : Import (export) containers of ship \( s \) unloaded (loaded) by ship \( s \),
\( J_{sc}^i \in J_{sc}^i \ ( J_{sc}^e \in J_{sc}^e ) \)

\( j_0 ( j_i ) : \) Dummy starting (ending) job of nil processing time for all YMs

\( J^* : J \cup \{ j_0, j_i \} \)

\( \tau_{ja} : \) Loading/unloading time of \( j_{sc} \)

\( p_j : \) Processing time of \( j \) where \( j \in J \)

\( \pi_{jj'} : \) The setup time required by a YM to process \( j' \in J \) just after processing \( j \in J \)

\( STA_s \) (STD_s) : Scheduled arrival (departure) time of ship \( s \). If a ship arrives early or late, then the scheduled time should be revised.

\( w_s : \) Penalty on service time of ship \( s \)

The Model

The objective of this model is to minimise the total weighted ship service time:

\[
Minimise \sum_{s \in S} w_s \times (ATD_s - ATA_s) \quad (8.1)
\]

Constraints 8.2 and 8.3 ensure that ship \( s \) berths after its scheduled arrival time, and after actual departure time of the previous ship:
\[ \text{ATA}_s \geq \text{STA}_s \quad \forall s \in S_b \quad (8.2) \]
\[ \text{ATA}_s \geq \text{ATD}_s \quad \forall s, s' \in S_b, \text{ where } s \text{ precedes } s' \quad (8.3) \]

Constraint 8.4 ensures that loading and unloading of containers can not start until the ship's berths:

\[ R_{j_s} \geq \text{ATA}_s \quad (8.4) \]

Constraint 8.5 ensures that ship \( s \) leaves the berth after completion of loading and unloading of containers:

\[ \text{ATD}_s \geq R_{j_s} + \tau_{j_s} \quad (8.5) \]

Constraints 8.6 - 8.9 ensure that each container must be processed by a single YM and each YM can process only one container at a time:

\[ \sum_{j \in J} X_{mj,j} = 1 \quad (8.6) \]
\[ \sum_{j \in J} X_{mj,j} = 1 \quad (8.7) \]
\[ \sum_{m} \sum_{j \in J} X_{mj,j} = 1 \quad \forall j \in J \quad (8.8) \]
\[ \sum_{j \in J} X_{mj,j} - \sum_{j \in J} X_{mj,j}' = 0 \quad \forall j \in J \quad (8.9) \]

Constraint 8.10 ensures that QC loads and unloads containers one after the other:

\[ R_{j} \geq R_{j} + \tau_{j} \quad \forall j, j' \in J_{sc}, \text{ where } j \text{ precedes } j' \quad (8.10) \]

Constraint 8.11 ensures that container \( j \) must be unloaded from ships before processing by YMs:

\[ t_{j} \geq R_{j} + \tau_{j} \quad \forall j \in J_{sc}' \quad (8.11) \]

Constraint 8.12 ensures that export containers can only be loaded to the ship after they are moved to the marshalling area:

\[ R_{j} \geq u_{j} \quad \forall j \in J_{sc}' \quad (8.12) \]
Constraints 8.13-8.15 ensure that no more than one container at the marshalling area for each QC:

\[ R_j + \tau_j \geq t_j \quad \forall j, j' \in J_{sc}^i, \text{ where } j \text{ is unloaded before } j' \quad (8.13) \]

\[ u_j \geq t_j \quad \forall j \in J_{sc}^i; j' \in J_{sc}^c, \text{ where } j \text{ is unloaded before } j' \text{ is loaded} \quad (8.14) \]

\[ u_j \geq R_j \quad \text{where } j, j' \in J_{sc}^c \text{ and } j \text{ is loaded before } j' \quad (8.15) \]

Constraints 8.16-8.17 satisfy the starting and finishing time of \( m \)

\[ u_j \geq t_j + p_j \quad (8.16) \]

\[ t_{j'} \geq u_j + \pi_{m_{j'}} - (1 - X_{m_{j'}})K \quad \forall j, j' \in J \quad (8.17) \]

The heuristic and meta-heuristic techniques proposed in Chapter 6 are applicable to the multi-berth and multi-ship case.

### 8.2 Resolving violation of precedence-constraints and deadlocks

In a multi-berth and multi-ship environment, loading/unloading of containers cannot start until the previous ship of the same berth has left. Consider Ship \( s_1 \) followed by \( s_2 \) at the same berth; QC \( c_i \) is to load the outbound container \( j_i \) to \( s_1 \), and \( j_2 \) to \( s_2 \).

The process sequence \textit{Seq 8.1} of YM \( m_i \) is infeasible because of the violation of precedence-constraints on ship sequence.

\textit{Seq 8.1}

\[ m_i : \{ j_2, j_1 \} \]
Now suppose there are two berths for two ships, $s_{11}$ and $s_{21}$, and they are followed by $s_{12}$ and $s_{22}$. Containers $j_{11}$, $j_{12}$ are to be loaded to $s_{11}, s_{21}$ by QC $c_1$; and $j_{21}$, $j_{22}$ are to load to $s_{12}$ and $s_{22}$ by QC $c_2$. The process sequences in **Seq 8.2** for YM $m_1$ and $m_2$ are infeasible because of deadlocks: $m_1$ can not process $j_{12}$ until $j_{11}$ has been loaded to Ship $s_{11}$, and $m_2$ can not process $j_{22}$ until $j_{12}$ has been loaded to Ship $s_{21}$.

**Seq 8.2**

$m_1 : \{ j_{12}, j_{21} \}$

$m_2 : \{ j_{22}, j_{11} \}$

Since the process sequence of each QC follows the ship sequence, the procedures for resolving deadlocks and violation of precedence constraints described in Chapter 6 can also apply to the multi-ship and multi-berth environment. Thus in **Seq 8.1**, $j_2$, $j_1$ are swapped to form the feasible sequence **Seq 8.1***; and in **Seq 8.2** $j_{12}$ and $j_{11}$ are swapped to form the feasible sequences shown in **Seq 8.2***.

**Seq 8.1**

$m_1 : \{ j_1, j_2 \}$

**Seq 8.2**

$m_1 : \{ j_{11}, j_{21} \}$

$m_2 : \{ j_{22}, j_{12} \}$
8.3 Storage Allocation Model (SAM)

During the planning stage, it is possible to re-assign the storage location for containers. Since containers are picked up from the top and stacked up from the bottom, it is better to consider a stack of containers in swapping storage locations in the yard. In this thesis, two stacks of 20-foot containers (or a stack of 40-foot containers) of the same type (all inbound or all outbound, and same ship) are regarded as a block. As CPM is already NP hard, the complexity of the problem will be further increased when yard storage allocation is also included. By grouping containers into blocks, the search space for optimal storage location is reduced. In a single-ship environment, alternative storage allocation schemes can be explored by neighbourhood search, using swapping and insertion as illustrated in Figure 8.2.

![Swapping and Insertion Diagram](image)

(a) Swapping of storage blocks

(b) Inserting storage block after another one

Figure 8.2 Generating new storage allocation scheme by (a) swapping and (b) insertion

However, storage locations in a multi-ship and multi-berth environment are re-usable. The re-assignment of storage locations of one ship may cause conflict with the storage scheme of other ships. For instance, Figure 8.3 shows the occupancy of Block1 and Block2 over time. It is not possible to swap the storage allocation of Block1 and Block 2 for containers of Ship 1 without affecting the storage scheme for containers of Ship 2.
It is difficult to calculate precisely the time when a storage block is free without being given the schedule of the container process. On the other hand, the construction of best scheduling plan will require the knowledge of the storage allocation scheme. Table 8.1 shows how storage blocks are shared in the yard based on the following rules when four ships $s_1$, $s_2$, $s_3$, and $s_4$ are to berth at $b_1$, $b_2$ ($s_1$ and $s_3$ berth at $b_1$; $s_1$ and $s_2$, $s_3$, and $s_4$ berth at $b_2$):

1. Containers of the same ship do not share the same storage block;
2. Containers of the first ship of each berth can not be stored in storage blocks occupied by ship of previous planning horizon;
3. Inbound containers stay in the yard for the whole planning horizon after they are stored in the storage block;
4. Storage blocks for outbound containers will be freed to the succeeding ships.
Table 8.1 Sharable storage locations

<table>
<thead>
<tr>
<th>Ship</th>
<th>Containers from previous ships</th>
<th>s₁</th>
<th>s₂</th>
<th>s₃</th>
<th>s₄</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(Inbound/outbound)</td>
<td>In</td>
<td>Out</td>
<td>In</td>
<td>Out</td>
</tr>
<tr>
<td>s₁</td>
<td>In</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Out</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s₂</td>
<td>In</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Out</td>
<td></td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s₃</td>
<td>In</td>
<td></td>
<td></td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>s₄</td>
<td>In</td>
<td></td>
<td></td>
<td>√</td>
<td>√</td>
</tr>
</tbody>
</table>

Legend:

√ shardable

Let $Z = \{z\}$ be the set of storage blocks in the yard; $\Pi = \{H\}$, where $H$ is a set of containers (of the same ship) to be placed in the same storage block; $s_H$ is the ship associated with $H$. Assume containers of the same ship do not share yard storage space and it is possible to build a set of ordered pairs, $E = \{(H, H') \in \Pi \times \Pi : H$ and $H'$ do not share the same storage block\} . To handle the case for containers left over from the previous planning horizon, let $Z_0 (\subset Z)$ be the set of storage blocks occupied by these containers; and the set $E_0 = \{H \in \Pi : H$ is not stored in any $z \in Z_0\}$. If the distance travelled by the yard machines is the only consideration, the problem under investigation becomes an assignment problem.

Let $H_{sz} (\in \Pi \cup \{\emptyset\})$ be the set of containers of ship $s$ occupying the storage block $z$. A list sequence of storage blocks $Z_H (\subset Z)$ in ascending order of total yard transfer distance can be created for each $H \in \Pi$. 

95
Algorithm 8.1: List sequence assignment

Step 1. Initialise a list sequence $L_{\Pi}$ of $H$.

Step 2. Initialise $H_\infty = \emptyset$ for all $s \in S$ and $z \in Z$.

Step 3. Initialise $Q = L_{\Pi}$.

Step 4. If $Q = \emptyset$, go to Step 11.

Step 5. Select the first set of containers $H \in Q$. Assume all containers for $H$ are of ship $s$.

Step 6. Go to Step 12 to find the nearest storage block available, $z^*$.

Step 7. Assign the storage block $z^*$ to all containers $j \in H$.

Step 8. Record the storage block $z^*$ as assigned to Ship $s$ associated in $H_\infty$.

Step 9. Remove $H$ from $Q$: $Q = Q \setminus \{H\}$.


Step 11. Update the transfer distance for all containers. Go to Step 20.

(Step 12-19 are for finding the nearest available storage block).

Step 12. Initialise $Z' = Z_H$.

Step 13. If $Z' = \emptyset$, stop - system errors.

Step 14. Select the first storage block $z'$ from $Z'$.

Step 15. Remove $z'$ from $Z'$: $Z' = Z' \setminus \{z'\}$.

Step 16. If $z \in Z_0$ and $H \in E_0$ (H should not be using storage block occupied ships from previous planning horizon), return to Step 14.

Step 17. For all $s \in S$, assign $H' = H_\infty$. If $(H, H') \in E$, return to Step 14.

Step 18. Assign $z^* = z'$.

Step 19. The nearest block is found. Return to Step 7.

Step 20. Stop.

With a predefined processing sequence of containers by YMs, meta-heuristics described in Section 6.3 can be adapted to find the best storage allocation scheme:
Algorithm 8.2 Tabu Search for best storage allocation

Step 1. Define a positive but decreasing function $g$ and an empty Tabu List $L$

Step 2. Set counter $n=0$, an initial list sequence $X_0$; the current solution as $X' = X_0$ and the best solution $X_{best} = X'$

Step 3. Generate the set of neighbourhood $N(X')$

Step 4. For each $X^*$ in $N(X)$, obtain the storage allocation scheme according Algorithm 8.1 and evaluate the objective value $f(X^*)$

Step 5. Obtain the best neighbour $X$ in $N(X')$

Step 6. $\Delta = f(X) - f(X')$

Step 7. If $\Delta \leq 0$, set $X_{best} = X$

Step 8. If $\Delta < g(n)$ or $X'$ is not in $L$, set $X' = X$

Step 9. Increase $n$ by 1 and update $L$

Step 10. Repeat Step 3 until the stopping condition is met

8.4 Storage allocation scheme and container process sequence

An import container must be unloaded before it can be moved to the yard; and an outbound container must be moved to the marshalling area before it can be loaded to the ship. With a limited capacity at each loading/unloading position in the marshalling area, YMs can only transfer containers according to the loading/unloading sequence. However, a change in storage allocation scheme has the effect of changing the process sequence of the land-side tasks.

A land-side task is better considered as one which transfers a container to a particular storage location instead of one which transfers a particular physical container. Figure 8.4(a) shows two inbound Containers $A$ and $B$ unloaded to the same position at the marshalling area; Container $A$ is moved to the storage location $a$, and Container $B$ is moved to storage location $b$. Figure 8.4(b) and (c) illustrate that
the swapping of the storage location in the yard has the same effect as swapping the process order of two containers.

Figure 8.4 Relationship between process order and storage locations

For the export process, there is a one-to-one correspondence between storage location on the ship and the location on the yard. Provided all export containers are of the same destination and the same weight category, swapping two land-side tasks of the same loading position at the marshalling area has the same effect as swapping storing locations on the ship (see Figure 8.5(a)-(c)).
Relationship between storage locations on the ship and those in the yard

Figure 8.5 Relationship between storage locations on the ship and those in the yard

From above, although the container process sequence appeared to be violating precedence-constraints, it may become feasible by re-allocation of storage location – giving sequence flexibility as discussed in Section 5.1.

8.5 CPM-SAM

Given a predefined storage allocation scheme, the ship service time (or ship delay, etc) can be improved by optimising the schedule of YMs. On the other hand, the ship service time can be improved by optimising the storage allocation. For the best ship service time, a natural course of action is to combine the optimisation of scheduling YMs as well as the storage allocation together.

This approach involves interweaving the meta-heuristic techniques for CPM and SAM, and a framework is proposed in Algorithm 8.3:
Algorithm 8.3

Step 1. According to an initial list sequence of storage blocks, \( L_\Pi \). Assign storage blocks according to Algorithms 8.1.

Step 2. With storage allocation scheme from Step 1, obtain the best schedule using meta-heuristics (MH1).

Step 3. Apply meta-heuristics (MH2) on the best storage allocation scheme. For each new alternative schedule (a neighbour in Tabu Search and Simulated Annealing; or a new child in genetic algorithm), apply meta-heuristics (MH3) for the best schedule.

MH1, MH2 and MH3 can be meta-heuristics of the same type, or a combination of meta-heuristics of different types. However, all meta-heuristics require a large number of iterations in searching for the best solution regardless whether they are TS, SA, or GA. If \( MH2 \) requires \( N_2 \) iterations and \( MH3 \) requires \( N_3 \) iterations, the total number of iterations would be \( N_2 \times N_3 \). There would have to be a trade-off between the quality of the solution and the total number of iterations being run.

When using TS (or SA) for \( MH2 \) and \( MH3 \), each move in \( MH2 \) is only a small perturbation from the previous one. The new solution as an input, would allow continual neighbourhood search using \( MH3 \).

Although good results have been reported in using GA for solving large scale optimisation problems, GA may not be a suitable candidate for \( MH2 \) and \( MH3 \). GA requires an initial population of chromosome, causing a substantial amount of computational overhead when used as \( MH3 \). When GA is selected for \( MH2 \) as well, new generations are normally much different from the parent than the neighbours in TS. It would be unsure whether best chromosomes obtained in previous run of \( MH3 \) are still good chromosomes for the new generations from \( MH2 \).
8.6 Numerical investigations

Under a multi-ship situation, more containers in the yard are to be taken into consideration and some of these containers are to be placed further away from their ship. On the other hand, some storage area assigned to a particular ship will be available again after the ship is left. Table 8-2 shows the results from C++ implementation of CPM and CPM-SAM in multi-ship and multi-ship environment obtained using a Pentium IV personal computer of 2.8 MHz. The sample data set is constructed with each ship carrying 270 inbound and 270 outbound containers; each ship is assigned with two QCs; rows in yard storage are perpendicular to the water-line; all ship service times have equal weighting. The lowest-bounds are estimated by assuming zero transfer times of YMs. In Problem 1-6 and Problem 9, the storage yard is initially empty and the space is assumed to be reusable: storage space for outbound containers is assigned at the start of the planning horizon; the space will be free after their ship has left and is available for the inbound containers of the next arriving ships. The rules for re-usable storage area can be adapted to suit the environment of a particular port. In Problem 7 and 10, the yard is initially half-full of containers from the previous planning horizon - the space will be released to the second ship or each berth. In Problem 8 and 11, the storage space is not re-usable but the storage yard is initially free.

For all of Problem 1-11, Tabu Search is selected for the meta-heuristics MH1, MH2 and MH3 with the following stopping conditions:

\[MH1\]: maximum 20,000 iterations or no improvement after 2,000 iterations;

\[MH2\]: maximum 800 iterations or no improvement after 200 iterations; and

\[MH3\]: maximum 10 iterations

The results in Table 8-2 show CPM generally improves the container processes significantly over initial schedules of no-pooling. The container processes are further improved with the combined CPM-SAM.
Note that initially the weighted ship service times for Problem 9 (no left over containers) is greater than that of Problem 10 (with left-over containers). This is not unreasonable, but merely indicates the complexity of the system. Even though some storage space is taken by containers of the previous planning horizon, the space is released to the second and third ship of each berth. Besides, the distance travelled by yard machines are sequence-dependent. CPM-SAM does give a better result for Problem 9 than for Problem 10.

When storage area is not re-usable during the planning horizon, the initial no-pooling schedule gives much greater weighted ship service time. CPM and CPM-SAM are able to improve the container process.
Table 8-2 Results from CPM in multi-berth and multi-ship environment and CPM-SAM

<table>
<thead>
<tr>
<th>Problem</th>
<th>Berth</th>
<th>ship</th>
<th>QC</th>
<th>YM</th>
<th>Init Obj</th>
<th>CPM</th>
<th>% improvement</th>
<th>CPM-SAM</th>
<th>% improvement</th>
<th>Lowest Bound</th>
<th>CPU (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1046</td>
<td>1038</td>
<td>0.7%</td>
<td>909</td>
<td>13.1%</td>
<td>905</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>1237</td>
<td>1014</td>
<td>18.0%</td>
<td>908</td>
<td>26.6%</td>
<td>905</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>2006</td>
<td>1615</td>
<td>19.5%</td>
<td>1366</td>
<td>31.9%</td>
<td>1358</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>2919</td>
<td>1915</td>
<td>34.4%</td>
<td>1691</td>
<td>42.0%</td>
<td>1358</td>
<td>68</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2732</td>
<td>2252</td>
<td>17.6%</td>
<td>1828</td>
<td>33.1%</td>
<td>1810</td>
<td>35</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3232</td>
<td>2758</td>
<td>14.7%</td>
<td>2281</td>
<td>29.4%</td>
<td>2263</td>
<td>46</td>
</tr>
<tr>
<td>7†</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3313</td>
<td>2769</td>
<td>16.4%</td>
<td>2283</td>
<td>31.1%</td>
<td>2263</td>
<td>47</td>
</tr>
<tr>
<td>8†</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3398</td>
<td>2877</td>
<td>15.4%</td>
<td>2321</td>
<td>31.7%</td>
<td>2263</td>
<td>47</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>4610</td>
<td>3397</td>
<td>26.3%</td>
<td>2806</td>
<td>39.1%</td>
<td>2515</td>
<td>65</td>
</tr>
<tr>
<td>10†</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>4458</td>
<td>3398</td>
<td>23.8%</td>
<td>2898</td>
<td>35.0%</td>
<td>2515</td>
<td>69</td>
</tr>
<tr>
<td>11†</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>6045</td>
<td>4088</td>
<td>32.4%</td>
<td>3147</td>
<td>47.9%</td>
<td>2515</td>
<td>65</td>
</tr>
</tbody>
</table>

Note: † The storage yard is originally half full with containers from previous planning horizon

* The storage space is not re-usable during the planning region

Objectives are weighted ship service time in minutes
Chapter 9

Ship delays

At a seaport terminal, ships are expected to arrive and leave within a scheduled time window. The scheduled time of arrival (STA) of a ship depends on when the ship can leave the previous port, and the time it will take to travel to this port. The scheduled time of departure (STD) should exceed the STA by the estimated ship service time. The Container Process Model (CPM) developed in Chapter 8 can be a useful tool for estimating the ship service time. Both the ships and the goods they carried have a very high dollar value, and so the costs of delay are also very high. In the ideal situation, ship delays are not expected if the port has sufficient resources and a good planning. Nevertheless, a ship may arrive late due to poor weather conditions or disruptions at the previous port. In addition, a ship may depart late due to machine breakdowns or labour disputes. If a delay does happen to a ship, the ship may also be late for its subsequent port. This chapter investigates a model for minimising the total weighted ship delay in a multi-berth and multi-ship environment. The aim of this study is to develop a decision tool to address problem of ship delays arising from disruptions. Numerical investigations on solving the ship delay problem are presented.

9.1 Ship delay model

To minimise the impact of ship delays, port operators must consider alternate arrangements including re-assignment of berths, re-sequencing of ships and rescheduling of the container process. Assume that quay-crane (QC) operate at the same speed and the number of QCs is pre-assigned to the ships; one and only one ship can stay in a berth at any time, so ship sequences can be processed independently on different berths. Note that the outbound containers gradually move into the storage yard over a number of days. These outbound containers should be
stored close to the berthing position of a ship. If a different berth is re-assigned to the ship, its outbound containers may become much further away.

The following is the ship delay model (SDM) adapted from CPM developed in Chapter 8, and it also addresses the issues of berth allocation as well as ship sequencing:

**Parameters and indices**

- $n_S (n_B)$: Number of ships (berths)
- $p_w$: Processing time of $j \in J$ when its ship berths at $b$
- $\pi_{b,j,j'}$: The setup time required by a YM to process $j' \in J$ just after processing $j \in J$, where ship of $j$ berths at $b$ and ship of $j'$ berths at $b'$
- $STA_s (STD_s)$: Scheduled arrival (departure) time of ship $s$. If a ship arrives early or late, then the scheduled time should be revised
- $K$: Sufficient large number
- $w_s$: Penalty on service time of ship $s$

**Variables**

- $X_{bsk}$: Binary variable represents whether $s$ is the $k^{th}$ ship berthed at $b$

**The Model**

The objective of this model is to minimise the total weighted ship delay:

$$\text{Minimise } \sum_{s \in S} \left( w_s \times (ATD_s - STD_s) \right)$$

(9.1)

In addition to Constraint 8.2 to 8.15, Constraint 9.2 and 9.3 ensure that each ship is processed but not more than one time:

$$\sum_b \sum_k X_{bsk} = 1$$

(9.2)
\[
\sum_b \sum_s X_{bsk} \leq 1 \tag{9.3}
\]

If Ship \(s\) and \(s'\) are the \(k\)th and \((k')\)th ships berth at \(b\), \(X_{bsk} = 1\) and \(X_{bsk'} = 1\). Constraint 9.4 ensures that a ship leaves before the berthing of its successive ship:

\[
ATA_s \geq ATD_s - K(2 - X_{bsk} - X_{bsk'}) \quad \forall s, s' \in S; b \in B; 1 \leq k < k' \leq n_s \tag{9.4}
\]

Constraint 8.16 and 8.17 are to be replaced by 9.5 and 9.6 for the inclusion of the berths where the containers are loaded or unloaded:

\[
u_j \geq t_j + p_{bj} \tag{9.5}
\]

\[
t_j \geq u_j + \pi_{mbh'j'} - (1 - X_{mbh'j'})K \quad \forall j, j' \in J \tag{9.6}
\]

For a fixed berth allocation and ship sequencing, CPM and the meta-heuristics developed in Chapter 8 can be applied to find the best solution on weighted ship service time. By using the same objective function 9.1 as SDM, the model denoted as CPM-SD minimise the total weighted ship delay for fixed berth allocation and ship sequencing. The solution techniques developed in Chapter 8 is also applicable to CPM-SD. CPM-SD can be reduced to CPM and is therefore NP hard as well.

Let \(q_b = \{s_{b1}, s_{b2}, \ldots, s_{bn_b}\}\) be a sequence of \(N_b\) ships on berth \(b\); and \(Q = \{q_b : b \in B\}\) be the set of ship sequences on all berths; \(D(Q)\) be the total weighted ship delay with the pre-defined set of ship sequences \(Q\). SDM deals with the case when berth allocation and ship sequencing are deterministic; and the best \(D(Q)\) is determined by CPM-SD. Even if \(D(Q)\) can be determined in polynomial time, the ship delay problem can still be reduced to parallel machine problem (berths as parallel machines and ships as job) and is NP hard. In other words, the ship delay problem is a combination of two NP hard problems into one - making it complex and formidable.

Using "|" as a separator for two ship sequences, for example, Figure 9.1 shows set of ship sequences \(Q\) on multi-berth represented by a single sequence. The number of
permutations is \((n_s + n_B - 1)!/(n_B - 1)!\). Each permutation represents a feasible set of ship sequences, and requires CPM-SD to find the ship delay. Only for a very small of berths and a very small number of ships, ship delay problem can be solved by iterating through all the permutations within reasonable duration of time.

\[
\begin{array}{cccccc}
\varepsilon_1 & \varepsilon_2 & \varepsilon_2 & \varepsilon_4 & | & \varepsilon_3 & \varepsilon_6 \\
\hline
\hat{b}_1 & & & & \hat{b}_2
\end{array}
\]

**Figure 9.1** Schedule of multi-ship on multi-berth represented by partial sequences

Starting with an investigation into the lower bounds, solution techniques for solving SDM is developed later in this chapter.

### 9.2 Lower bound of total weighted ship delay

If the transfer and handling times of YM are disregarded, the service time is calculated from the loading/unloading times only and the result is the lower bound of service time \(p^{LB}_s\). With the assumption that QCs are pre-assigned ships and they operate at the same speed, \(p^{LB}_s\) is independent of berth allocation and ship sequencing. The lower bound of total weighted ship delay \(D^{LB}(Q)\) can be calculated using the following set of recursive equations (9.6 to 9.9)

\[
\begin{align*}
ATA_s &= STA_s & \text{if } s \text{ is the first ship in } q_b & \quad (9.6) \\
ATA_s &= \max(ATA_{s'}, STA_{s'}) & \text{for } s, s' \in q_b \text{ such that } s \text{ precedes } s' & \quad (9.7) \\
ATD_s &= ATA_s + p^{LB}_s & \quad (9.8) \\
D^{LB}(Q) &= \sum_{s \in S} \left( w_s \times (ATD_s - STD_s) \right) & \quad (9.9)
\end{align*}
\]

The complexity of finding \(D^{LB}(Q)\) is \(O(n_B + n_s)\).
Let $Q^*$ be the set of ship sequences currently found. When iterating through the
feasible sets of ship sequences, the lower bound of the set of ship sequences
$D^{LB}(Q)$ can be computed quickly. If $D^{LB}(Q) > D(Q^*)$, $Q$ should be rejected and
there is no need to take substantially longer time to find $D(Q)$ by CPM-SD.
Nevertheless, a more efficient way is necessary for the iteration of the feasible sets
of ship sequences. For such purpose, Theorem 9.1 to 9.3 help reducing the amount
of search space.

**Theorem 9.1** Let $Q$ be the set of ship sequences $\{q_1, q_2, \ldots, q_n\}$. For any
$Q' = \{q'_1, q'_2, \ldots, q'_{n+k}\}$ formed by the permutation of $Q$,

$$D^{LB}(Q') = D^{LB}(Q)$$

**Proof.** For any ship sequence $q \in Q$, $q$ is also in $Q'$. Since $p_s^{LB}$ is independent the
berth Ship $s$ is assigned, $ATA_s$ and $ATD_s$ are also independent on the berth
assignment.

Therefore,

$$D^{LB}(Q') = \sum_{s \in S} \left( w_s \times (ATD_s - STD_s)^{\top} \right) = \sum_{q \in Q} \sum_{s \in q} \left( w_s \times (ATD_s - STD_s)^{\top} \right)$$

$$= \sum_{q \in Q} \sum_{s \in q} \left( w_s \times (ATD_s - STD_s)^{\top} \right) = D^{LB}(Q)$$

**Theorem 9.2** Let $A = \{s_1, s_2, s_3, \ldots, s_n\} \subset S$ and $p_{s_1}^{LB}, p_{s_2}^{LB}, \ldots, p_{s_n}^{LB}$ are non-decreasing.

If $k$ ships $s_1', s_2', s_3', \ldots, s_k'$ ($k \neq 0$) are selected from $A$ to form a sequence, then

i) \[ ATD_{s_i} \geq \min_{1 \leq i \leq m} \left( STA_{s_i} \right) + \sum_{i=1}^{k} p_{s_i}^{LB} \]
ii) \[
D^{LB}(Q) \geq \min_{i \in S_n} \left( w_{s_i} \right) \times \left( \min_{i \in S_n} \left( STA_{s_i} \right) + \sum_{i=1}^{k} p_{s_i}^{LB} - \max_{i \in S_n} \left( STD_{s_i} \right) \right)^+
\]

Proof.

i) For any \( i \) (\( 1 < i \leq k \)),

\[
ATA_{s_i} \geq ATD_{s_{i-1}} \geq ATA_{s_{i-1}} + p_{s_{i-1}}^{LB} \geq STA_{s_{i-1}} + p_{s_{i-1}}^{LB}
\]

Therefore,

\[
ATD_{s_{k}} \geq ATA_{s_{k-1}} + p_{s_{k-1}}^{LB} \geq STA_{s_{k-1}} + p_{s_{k-1}}^{LB}
\]

\[
\geq ATA_{s_{i}} + p_{s_{i}}^{LB} + p_{s_{i+1}}^{LB} + \ldots + p_{s_{k-2}}^{LB} + p_{s_{k}}^{LB}
\]

\[
\geq ATA_{s_{i}} + \sum_{i=1}^{k} p_{s_{i}}^{LB} \geq ATA_{s_{i}} + \sum_{i=1}^{k} p_{s_{i}}^{LB}
\]

\[
\geq \min_{i \in S_n} \left( STA_{s_i} \right) + \sum_{i=1}^{k} p_{s_i}^{LB}
\]

ii) Since \( s_i' \in A \), \( w_{s_i} \geq \min_{i \in S_n} \left( w_{s_i} \right) \) and \( STD_{s_i} \leq \max_{i \in S_n} \left( STD_{s_i} \right) \).

Therefore,

\[
D^{LB}(Q) \geq w_{s_i} \times \left( ATD_{s_{i}} - STD_{s_{i}} \right)^+
\]

\[
\geq \min_{i \in S_n} \left( w_{s_i} \right) \times \left( \min_{i \in S_n} \left( STA_{s_i} \right) + \sum_{i=1}^{k} p_{s_i}^{LB} - \max_{i \in S_n} \left( STD_{s_i} \right) \right)^+
\]

Consider two berths and four ships with \( STA, STD, p_{s_i}^{LB}, w_s \) as shown in Table 9.1.
Table 9.1 Sample data from four ships

<table>
<thead>
<tr>
<th>Ship</th>
<th>$STA$</th>
<th>$STD$</th>
<th>$p_s^{LB}$</th>
<th>$W_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>$s_2$</td>
<td>2</td>
<td>14</td>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td>$s_3$</td>
<td>8</td>
<td>12</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>$s_4$</td>
<td>10</td>
<td>16</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

Suppose the initial set of ship sequences is $Q = \{ \{s_1,s_3\}, \{s_2,s_4\} \}$. From Table 9.1, $D^{LB}(Q) = 18$

Example 9.1 From Theorem 9.1, both of the following two sets of sequences have the same lower bound:

$\{ \{s_1,s_2,s_3\}, \{s_4\} \}$

$\{ \{s_4\}, \{s_1,s_2,s_3\} \}$

Example 9.2 Let $Q$ be of form $'\overline{XXXX}'$ (all ships are serviced at the second berth).

From Theorem 9.2, $D^{LB}(Q) \geq 1 \times (0 + 44 - 16) = 28$

Example 9.3 Let $Q$ be of form $'X \overline{XX}s_1'$. The ships before $s_1$ are selected from $s_2, s_3, s_4$. Let $s^*$ be the ship just precedes $s_1$.

$ATD_{s^*} \geq 2 + 6 + 12 = 20$

$ATD_{s_1} \geq ATD_{s^*} + 8 = 28$

Therefore, $D^{LB}(Q) \geq w_{s_1} \times (ATD_{s_1} - STD_{s_1})^+ = 20$

9.3 Solution techniques for SDM

When the berth allocation and ship sequences ($Q$) are pre-defined, CPM-SD can be applied to find $D(Q)$. When berth allocation and ship sequencing are deterministic, the container process problem becomes much more complex and difficult to solve.
Fortunately, ship time windows are not defined arbitrary and many of the feasible sets of ship sequences can be eliminated by testing the lower bound.

In solving SDM, a search tree is constructed using the following rules:

Rule 0. Level 0 is the dummy node.

Rule 1. Each node in Level 1 is a combination of ‘|’ and ‘X’, where ‘|’ representing the berth separator and ‘X’ representing a ship. Only combinations with non-decreasing number of berths are considered, for example, ‘|XXXX’, ‘X|XXX’ and ‘XX|XX’.

Rule 2. Each node in Level 2 and beyond is generated by fixing the last ship on the queue with the highest number of unassigned ships (see Figure 9.2).

The branching method for finding $\min_{Q}(D(Q))$ traverses the nodes in the search tree with the depth-first is shown below:
Algorithm 9.1

Step 1. Obtain the initial set of ship sequences $Q_0$. Generate the permutation set $P(Q_0)$ from $Q_0$. Apply CPM-SD to every permutation in $P(Q_0)$; and find the best permutation $Q'$ such that $D(Q') = \min_{q \in P(Q_0)} (D(q))$. Set $Q^* = Q'$

Step 2. Generate a new Level 1 node $\Theta_1$ using Rule 1. Set the level $k=1$.

Step 3. If no more nodes are generated using Rule 1, go to Step 12

Step 4. If the lower bound of $\Theta_1$ is greater than $D(Q^*)$ (see Example 9.2) go to Step 2

Step 5. Generate a new Level $k+1$ node $\Theta_{k+1}$ from $\Theta_k$ using Rule 2

Step 6. If no more nodes are generated, go to Step 11

Step 7. If the lower bound of $\Theta_{k+1}$ is greater than $D(Q^*)$ go to Step 11

Step 8. If $\Theta_{k+1}$ is the not the deepest node, go to Step 10

Step 9. Let $\Theta_{k+1}$ represents the set of ship sequences $Q = \{q_1, q_2, ..., q_n\}$.

Generate the permutation set $P(Q)$ of $Q$.

Step 10. Apply CPM-SD to every permutation in $P(Q)$ and find the permutation $Q'$ with the least total weighted ship delay. If $D(Q') \leq D(Q^*)$, set $Q^* = Q'$

Step 11. Go to Step 11

Step 10. Set Level $k=k+1$ and go to Step 5

Step 11. If $k>1$, set Level $k=k-1$ and go to Step 5; otherwise go to Step 2

Step 12. Take $Q^*$ as the optimal solution

As to the nature of the problem, a great number of schedules will be eliminated through quick check on the lower bound $D^{LB}(Q)$. Most meta-heuristics (GA, TS, SA, and the like) all require testing neighbours/children in deciding on the next move/generation, even the neighbours/children are obviously unfit solutions.
9.4 A case study

Consider six ships \((s_1, s_2, s_3, s_4, s_5, s_6)\) two berths \((b_1, b_2)\); the initial plan is to have \(s_1, s_3, s_5\) berthed at \(b_1\) one after the other, and \(s_2, s_4, s_6\) to berthed at \(b_2\). These ships are ready and to be serviced by four QC's and four YMs. From a preliminary run using CPM with weighted ship service time as the objective function (assuming all ships are of the same weighting), the scheduled ship time windows shown in Table 9.2 are achievable and with no ship delays. In fact 60 minutes have been added to STD of all ships to allow for a tolerance to small disruptions.

<table>
<thead>
<tr>
<th>Berth</th>
<th>Ship</th>
<th>No of containers</th>
<th>STA (min)</th>
<th>STD (min)</th>
<th>Ship time from LB (min)</th>
<th>CPM (min)</th>
<th>(p_{s}^{LB}) (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(s_1)</td>
<td>540</td>
<td>0</td>
<td>522</td>
<td>462</td>
<td>453</td>
<td>453</td>
</tr>
<tr>
<td></td>
<td>(s_3)</td>
<td>900</td>
<td>462</td>
<td>1300</td>
<td>778</td>
<td>754</td>
<td>754</td>
</tr>
<tr>
<td></td>
<td>(s_5)</td>
<td>300</td>
<td>1240</td>
<td>1562</td>
<td>261</td>
<td>251</td>
<td>251</td>
</tr>
<tr>
<td>2</td>
<td>(s_2)</td>
<td>300</td>
<td>0</td>
<td>318</td>
<td>258</td>
<td>251</td>
<td>251</td>
</tr>
<tr>
<td></td>
<td>(s_4)</td>
<td>540</td>
<td>258</td>
<td>778</td>
<td>460</td>
<td>453</td>
<td>453</td>
</tr>
<tr>
<td></td>
<td>(s_6)</td>
<td>900</td>
<td>718</td>
<td>1558</td>
<td>779</td>
<td>754</td>
<td>754</td>
</tr>
</tbody>
</table>

Just before the start of the planning horizon, \(s_2\) informs the port that it will be late and the new STA is now 180 minutes. The port operator will need to decide whether to main the original berth allocation and ship sequencing; or to find other alternative arrangement. The initial schedule has the lower-bound is 299 minutes and the objective value is 366 minutes. From Algorithm 9.1, there are only four deepest nodes with lower bound less than or equal to 366 minutes (see Appendix D). CPM-SD is then applied for the weighted ship delay of these nodes (see results as shown in Table 9.3). From the analysis above, the initial schedule remains the best. The computation time for finding the best schedule takes about 5 minutes only. This case study demonstrates that SDM can be solved very efficiently using the proposed solution techniques.
Table 9.3  Weighted ship delay of nodes with lower bound less than or equal to the initial objective value

<table>
<thead>
<tr>
<th>$b_1$</th>
<th>$b_2$</th>
<th>LB</th>
<th>Objective value (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$, $s_3$, $s_5$</td>
<td>$s_2$, $s_4$, $s_6$</td>
<td>299</td>
<td>366</td>
</tr>
<tr>
<td>$s_2$, $s_4$, $s_6$</td>
<td>$s_1$, $s_3$, $s_5$</td>
<td>299</td>
<td>386</td>
</tr>
<tr>
<td>$s_1$, $s_4$, $s_6$</td>
<td>$s_2$, $s_3$, $s_5$</td>
<td>343</td>
<td>441</td>
</tr>
<tr>
<td>$s_2$, $s_3$, $s_5$</td>
<td>$s_1$, $s_4$, $s_6$</td>
<td>343</td>
<td>425</td>
</tr>
</tbody>
</table>
Chapter 10

Conclusions and directions for future research

This research has focussed on the planning and operation in multimodel container terminals. Due to the complexity of seaport systems, most previous research work either considers only a single berth or one transfer machine in the yard. Other research studies may consider import or export operation but not both. There have been some attempts to solve the various aspects of this overall problem. However, to the best of the author's knowledge from the literature review and "insider knowledge" (i.e. visits to various seaport container terminals, liaison with colleagues at seminars and conferences), little contemporary work of the problem investigated has been done or publicised for large size container terminals. This confirms the present lack of research into integrated planning and control models. To fill this gap, models are developed in this thesis for the effective use of infrastructure and associated functions in container terminals.

This chapter summarises the findings of the research and recommends directions for future research.

10.1 Conclusions

The Container Process Model (CPM) was developed in Chapter 5 concerning the loading/unloading operations, yard layout and the handling and transfer operations in multimodal container terminals. However, CPM is NP hard and the real-life problems often involve large numbers of containers. For these reasons, heuristics (priority rule, list scheduling), meta-heuristics, as well as deadlock resolutions were studied. From the results of the numerical investigations, Tabu Search was found to perform better with new solutions being constructed from set of list sequences on
yard machines. Demonstrated in Chapter 7, the model can be applied to analyse the efficiency of the yard layout. On the benchmark problems, the container process was found to perform better if rows are perpendicular to the quay line and that each row has about 10 to 14 bays.

The port is a complex system. Resources including QCs and YMs may not be available at all times. For the case when QCs and YMs are known to be unavailable in advance, the issue was addressed in Chapter 5. For unplanned events like machine breakdowns, a re-active scheduling algorithm was proposed in Chapter 7. On the benchmark problems, the reactive scheduling algorithm gave results similar to those as if the disruptions were planned in advance.

The marshalling area acts an intermediate buffer between the land-side and the sea-side operations. CPM was extended to include the capacity of the marshalling area. On the benchmark problems, the improvement was found to be less apparent when the capacity reaches three containers per QC. For those ports using terminal trucks, the capacity at the marshalling area is zero. This model can be a decision tool for analysing the benefits on changing over to equipment with lifting capabilities (for example, automated straddle-carriers) and to see whether the costs can be justified.

A multi-berth and multiple-ship environment adds more complexity to the container process problem. Such an environment also involves many more containers and ships having precedence relationships among themselves, making the problem even harder to solve with today's fast computer processing power. An extension to CPM for the multi-berth and multi-ship environment was proposed in Chapter 8.

To further improve the container process, the problem relating to storage allocation was investigated in Chapter 8. The model CPM-SAM was developed in addressing the issues concerning both scheduling and storage allocation, taking into consideration that storage space is released when containers have left the yard. The problem was solved using meta-heuristic techniques, which iteratively searched for the best schedule and storage allocation scheme. The best solutions on the numerical examples were obtained within a reasonable amount of time.
In minimising the impact of ship delays, the Ship Delay Model was proposed in Chapter 9 for the optimum schedule, berth allocation and ship sequencing. An innovative meta-heuristic technique was also developed using Tabu Search along with bounding and branching.

In summary, the models developed in this thesis establish the relationship between significant factors and the options for increasing throughput by discovering the bottlenecks, which restrict entity flow through the system. The associated solution techniques have been successfully implemented as computer software. The findings of this research may be adopted by ports to address a range of important issues for efficiency improvement, subject to the support of information and telecommunications technology. In any case, the models may be applicable as decision tools for operation planning, yard layout, and cost and benefit analysis for future investment in infrastructures.

10.2 Directions for future research

The following areas are recommended for future research:

1. This thesis assumes that re-handling is not required in the storage yard if heavier containers are stacked on top of lighter ones as they arrive. Even if this is the case, re-handling may still be required for the retrieval containers out of the port. With advancement of technology, port operators would know in advance the time when a truck will arrive to collect the containers. By integrating the road transport and the stacking of containers in the yard, less re-handling will be required. The port will have less demand for handling equipment and truck turn-around time will decrease.

2. The use of rail for transporting containers is increasingly popular. In Australia, several state governments set targets for the percentage of containers to and from the port by rail. In this thesis, rail intermodal terminal has been
considered just as an extension of the storage yard. In practice, a delay of train schedule can adversely affect the port as well. If the operations of the sea-port and the intermodal terminal can be synchronised, there will be less demand for storage space. It may be possible to have containers transferred directly between trains and the ships with less re-handling requirement.

3. It may be interesting to investigate the practical issue when the service of a ship spans over two planning periods. This will also enable us to solve much larger sized problems by dividing the planning horizon into several shorter intervals, where each interval involves a smaller number of ships and containers.

4. This thesis relies on simple rules for re-using storage space. More research is needed for the development of models or strategies to enable better efficiency in storage yard in a multi-ship environment.

5. The lower bound of weighted ship delay is calculated using Theorem 9.1 and 9.2. If a better lower bound can be obtained, the ship delay problem can be solved quicker.

6. In Chapter 8, the algorithm for iterating through the refinement of storage scheme and re-scheduling involves three meta-heuristics MH1, MH2 and MH3. It may be interesting to investigate the performance of the algorithm with different combinations of meta-heuristics (for example, GA for MH1, SA for MH2, TS for MH3), as well as different stopping conditions.
Appendix A. Solving CPM using LINGO 8

MODEL:
Title: Container Process Model;

!Declaration of sets;
sets:
sMach    /1 2/ ;  !set of machines;
sCrane   /1 2/ ;  !set of cranes;
sJob  / 1 2 3 4 5 6 7 8 /:  !set of jobs;
pJobShip,
pJobCrane,   !quay crane for the job;
pJobImport,  !0-import, 1-export;
pLoadUnLoad,  !loading/unloading time;
pProcessTime,
vR,     !release time of the job;
vt,     !start processing time;
vu,     !end processing time;
vJobMach;  !machine processing the job;

sJob_p  / 1 2 3 4 5 6 7 8 pSource pSink/;

sJobJob(sJob_p, sJob_p):
pSetupTime     !setup time;
;

sMachJobJob(sMach, sJob_p, sJob_p):
vX
;
endsets
!End declaration of sets;

!-------------------------------------------------------------------------------------
Definition of bounds
-------------------------------------------------------------------------------------;

!Declaration of variables;
!Also set uper bound for import start and export complete time;
@bnd(0,vATA,M);
@bnd(0,vATD,M);
@for(sJob(iJob):
    @bnd(0,vR(iJob),M);
    @bnd(0,vt(iJob),M);
    @bnd(0,vu(iJob),M);
);
@for(sJob_p(iJob):
    @for(sMach(iMach):
        vX( iMach, iJob, pSource) = 0;  !source;
        vX( iMach, pSink, iJob) = 0;  !sink;
        vX( iMach, iJob, iJob) = 0;
    );
);
Definition of objectives and constraints

![Objective]
MIN = vATD - vATA;

[eq2]
vATD >= pSTA;

@for (sJob(iJob):
[eq3]
vR(iJob) >= vATA;
);

@for (sJob(iJob):
[eq4]
vATD >= vR(iJob) + pLoadUnload(iJob);
);

@for(sMach(iMach):
[eq5]
@sum(sJob(iJob):
 vX(iMach, pSource, iJob)) !13 source;
 ) = 1;
);

@for(sMach(iMach):
[eq6]
@sum(sJob(iJob):
 vX(iMach, iJob, pSink)) !14-sink;
 ) = 1;
);

@for(sJob(iJob):
[eq7]
@sum(sMach(iMach):
@sum(sJob_p(iJobp) | iJob #ne# iJobp:
 vX(iMach, iJob, iJobp)
 )
 )
 = 1;
);

@for(sMach(iMach):
[eq8]
@for(sJob(iJob):

@for(sMach(iMach):
@for(sJob_p(iJob1):
@for(sJob_p(iJob2):
@bin(vX(iMach, iJob1, iJob2));
);
);
)
@bin(vX(iMach, iJob1, iJob2));
);
@sum(sJob_p(iJobp):
    vX(iMach, iJob, iJobp)
  )
- @sum(sJob_p(iJobp):
    vX(iMach, iJobp, iJob)
  )
= 0;
);

@for(sCrane(iCrane):
  @for(sJob(iJob1)|pJobCrane(iJob1) #eq# iCrane:
    @for(sJob(iJob2) |
        (pJobCrane(iJob2) #eq# iCrane)
        #and# (iJob1 #lt# iJob2):
            [eq9]
            vR(iJob2) >=
            vR(iJob1)
            + pLoadUnload(iJob1);
    );
  );
);

@for(sCrane(iCrane):
  @for(sJob(iJob1)|pJobCrane(iJob1) #eq# iCrane)
  #and# (pJobImport(iJob1) #eq# 0): !import;
  [eq10]
  vt(iJob1) >= vR(iJob1)
  + pLoadUnload(iJob1);
);}

@for(sCrane(iCrane):
  @for(sJob(iJob1)|pJobCrane(iJob1) #eq# iCrane)
  #and# (pJobImport(iJob1) #eq# 1): !export;
  [eq11]
  vR(iJob1) >= vu(iJob1);
);}

@for(sCrane(iCrane):
  @for(sJob(iJob1)|pJobCrane(iJob1) #eq# iCrane)
  #and# (pJobImport(iJob1) #eq# 0): !import;
  @for(sJob(iJob2)|pJobCrane(iJob2) #eq# iCrane)
  #and# (pJobImport(iJob2) #eq# 0)
  #and# (iJob2 #gt# iJob1):
  [eq12]
  vR(iJob2) + pLoadUnload(iJob2) > vt(iJob1) + delta;
);}

@for(sCrane(iCrane):
@for(sJob(iJob1)|vJobCrane(iJob1) #eq# iCrane) #and# (vJobImport(iJob1) #eq# 0):  !import;  
@for(sJob(iJob2)|vJobCrane(iJob2) #eq# iCrane) #and# (vJobImport(iJob2) #eq# 1)  !export;  
#and# (iJob2 #gt# iJob1)  
  [eq13]  
vu(iJob2)  > vt(iJob1) + delta;  
);  
);

@for(sCrane(iCrane):  
  @for(sJob(iJob1)|vJobCrane(iJob1) #eq# iCrane) #and# (vJobImport(iJob1) #eq# 1):  !export;  
  @for(sJob(iJob2)|vJobCrane(iJob2) #eq# iCrane) #and# (vJobImport(iJob2) #eq# 1)  !export;  
#and# (iJob2 #gt# iJob1):  
  [eq14]  
vu(iJob2)  > vR(iJob1) + delta;  
);  
);

@for(sMach(iMach):  
  @for(sJob(iJob)):  
[eq15]  
vU(iJob)  
  >=  
v(iJob)  
  + pProcessTime(iJob)  
  ;  
);  
);

@for(sMach(iMach):  
  @for(sJob(iJob1)):  
  @for(sJob(iJob2)):  
[eq16]  
v(iJob2)  
  >=  
v(iJob1)  
  + pSetupTime(Job1, iJob2)  
  - (1-vX(iMach, iJob1, iJob2))*M  
  ;  
);  
);

Data:  
  M = 100000;  ! Maximum value;  
  delta = 0.0001;  
  pSTA = 0;
vATA = 0; !ship starts immediately upon arrival;
pSource=9;
pSink=10;

pJobCrane = @odbc('cpm','qryJobCrane', 'crane');
pJobImport = @odbc('cpm', 'qryContainer', 'import');
pLoadUnload = @odbc('cpm', 'qryContainer', 'load_unload');
pProcessTime = @odbc('cpm', 'qryProcessTime', 'ProcessTime');
pSetupTime = @odbc('cpm', 'qrySetupTime', 'SetupTime');

enddata

end
Appendix B. Selected Computer Code for CPM

.isHidden(true);
if (pCont->m_bImport)
{
  //import
  //the previous container must have been unloaded and buffer in the marshalling area is limited
  if (pPrevCraneCont!=NULL)
  {
    if (!(pPrevCraneCont->m_bLoadUnloadComplete)
        ||!(pPrevCraneCont->m_bProcessStarted))
    {
      return false;
    }
  }
  if (pLastBufCraneCont!=NULL)
  {
    if (!(pLastBufCraneCont->m_bLoadUnloadComplete)
        || !(pLastBufCraneCont->m_bProcessStarted))
    {
      return false;
    }
  }
  else
  {
    //export
    if (!(pCont->m_bProcessStarted))
    {
      //the export container has not yet arrived at the marshalling area
      return false;
    }
  }
}

//Is pCont an import container?
if (pCont->m_bImport)
{
  if (pPrevCraneCont==NULL)
  {
    //pCont is the first container to be loaded/unloaded by the shore-crane
    pShip = pBatch->m_pShip + pCont->m_lShip;
    //loading/unloading starts after the STA of the ship
    pCont->m_dStartLoadUnload = pShip->m_dT1;
  }
  else
  {
    //pCont is the not first container to be loaded/unloaded by the shore-crane
    //loading/unloading does not start until the previous one has been completed
    dStartLoadUnload = pPrevCraneCont->m_dStartLoadUnload;
  }
}
if (dStartLoadUnload < pShip->m_dT1)
{
    dStartLoadUnload = pShip->m_dT1;
}

//time when previous container loading/unloading is complete.
dStartLoadUnload = dStartLoadUnload + pPrevCraneCont->m_dLoadUnload;

//time unloading is complete
dTemp = dStartLoadUnload + pCont->m_dLoadUnload;

//check if there is any blocking
if (dTemp < pPrevCraneCont->m_dStartProcessTime)
{
    dTemp = pPrevCraneCont->m_dStartProcessTime;
dStartLoadUnload = dTemp - pCont->m_dLoadUnload;
}
pCont->m_dStartLoadUnload = dStartLoadUnload;

else
{
    //pCont is an export container
    dEndProcessTime = 0.0;
dStartProcessTime = 0.0;

    if (pPrevCraneCont!=NULL)
    {
        //pCont is not the first export container of the shore-crane
        //Deliver to marshalling area only after the previous container is moved away
dEndProcessTime = pPrevCraneCont->m_dStartProcessTime;
dStartLoadUnload = pPrevCraneCont->m_dStartLoadUnload + pPrevCraneCont->m_dLoadUnload;
    }

    if (pCont->m_dEndProcessTime < dEndProcessTime)
    {
        pCont->m_dEndProcessTime = dEndProcessTime;
    }

    if (dStartLoadUnload < pShip->m_dT1)
    {
        dStartLoadUnload = pShip->m_dT1;
    }

    if (dStartLoadUnload < pCont->m_dEndProcessTime)
    {
        dStartLoadUnload = pCont->m_dEndProcessTime;
    }

    pCont->m_dStartLoadUnload = dStartLoadUnload;
pCont->m_dCompleteTime = pCont->m_dStartLoadUnload + pCont->m_dLoadUnload;
}
pCont->m_bLoadUnloadComplete=true;

pShip = pBatch->m_pShip + pCont->m_lShip;
pShip->load_unload();

if (pShip->m_dT2 < pCont->m_dCompleteTime)
{
    pShip->m_dT2 = pCont->m_dCompleteTime;
}

return true;

bool CAssignJob::processMachJob(CBatch *pBatch, long lCrane, long lMach,
CContainer *pCont, CContainer *pPrevCraneCont, CContainer *pPrevMachCont)
{
    double dStartTime = 0.0;
    double dStartLoadUnload = 0.0;
    double dEndTime = 0.0;
    double dSetupTime = 0.0;
    double dReleaseTime = 0.0;
    double dTemp = 0.0;
    double dSpeed;
    CShip *pShip;

    long n;
    CContainer *pTempCont;
CContainer *pLastBufCraneCont; //container that should be removed from buffer

CContMap *pCraneContMap = pBatch->m_pCraneContMap + lCrane;
pLastBufCraneCont = pCraneContMap->getPrevCont(pCont->m_lJobNo, Pt_Ma_Buffer);

CMachine *pMachine = pBatch->m_pMachine + lMach;
dSpeed = pMachine->m_dSpeed;

//get the pointer to the ship object
pShip = pBatch->m_pShip + pCont->m_lShip;

if (pCont->m_bImport)
{
    //the import container container must have been unloaded
    if (!pCont->m_bLoadUnloadComplete)
    {
        return false;
    }
}
else
{
    //avoid blocking at marshalling area
    if (pLastBufCraneCont!=NULL)
    {
        if (pLastBufCraneCont->m_bImport)
        {
            if (!pLastBufCraneCont->m_bProcessStarted)
            {
                return false;
            }
        }
        else
        {
            if (!pLastBufCraneCont->m_bLoadUnloadComplete)
            {
                return false;
            }
        }
    }
}

///////Machine must complete one job before the next
if (pPrevMachCont!=NULL && !pPrevMachCont->m_bProcessStarted)
{
    return false;
}

if (!pCont->m_bImport) & & pPrevCraneCont!=NULL)
{
    //for outbound container we should wait until all previous QC job
    // to complete first for avoiding deadlock
    if (!pPrevCraneCont->m_bLoadUnloadComplete)
    {
        return false;
    }
}

dSetupTime = pBatch->getSetupTime(lMach, pPrevMachCont, pCont);
if (pCont->m_bImport) {
    //import container

dReleaseTime = pCont->m_dStartLoadUnload + pCont->m_dLoadUnload;

if (pPrevMachCont == NULL) {
    //this container is the first job of the yard machine
    //machine processing starts after the container is unloaded
    pCont->m_dStartProcessTime =
        pCont->m_dStartLoadUnload + pCont->m_dLoadUnload;
}
else {
    if (pPrevMachCont->m_bImport) {
        //the prev job of the machine for an import container
        //note: the one before the previous container may be an export one

        dTemp = pPrevMachCont->m_dEndProcessTime
            + dSetupTime;
        //checks whether the container has been unloaded before the machine is available
        if (dTemp < dReleaseTime) {
            dTemp = dReleaseTime;
        }
        pCont->m_dStartProcessTime = dTemp;
    } else {
        //previous container is an export one

        pCont->m_dStartProcessTime = pPrevMachCont->m_dEndProcessTime
            + dSetupTime;
        //checks whether the container has been unloaded before the machine is available
        if (pCont->m_dStartProcessTime < dReleaseTime) {
            //C9
            pCont->m_dStartProcessTime = dReleaseTime;
        }
    }
}
}

pCont->m_dEndProcessTime = pCont->m_dStartProcessTime
    + pBatch->getProcessTime(lMach, pPrevMachCont, pCont);
pCont->m_dCompleteTime = pCont->m_dEndProcessTime;
}
else {
    //export
if (pPrevMachCont==NULL) {
    //this container is the first job of the yard machine
    pCont->m_dStartProcessTime = pShip->m_dT1;
    pCont->m_dEndProcessTime = pCont->m_dStartProcessTime
    + pBatch->getProcessTime(lMach, NULL, pCont);
} else {
    //this container is not the first job of the machine
    if (pPrevMachCont->m_bImport) {
        //IE
        dStartTime = pPrevMachCont->m_dEndProcessTime;
        pCont->m_dStartProcessTime = pPrevMachCont->m_dEndProcessTime
        + dSetupTime;
        dTemp = pCont->m_dStartProcessTime
        + pBatch->getProcessTime(lMach, pPrevMachCont, pCont);
        pCont->m_dEndProcessTime = dTemp;
    } else {
        //Case EE - the previous container of the yard machine is also an export one
        pCont->m_dStartProcessTime = pPrevMachCont->m_dEndProcessTime
        + dSetupTime;
        dTemp = pCont->m_dStartProcessTime
        + pBatch->getProcessTime(lMach, pPrevMachCont, pCont);
        if (dTemp < pPrevCraneCont->m_dStartLoadUnload) {
            dTemp = pPrevCraneCont->m_dStartLoadUnload;
        }
        pCont->m_dEndProcessTime = dTemp;
    }
    if (pPrevCraneCont->m_bImport) {
        if (pCont->m_dEndProcessTime<pPrevCraneCont->m_dStartProcessTime) {
            pCont->m_dEndProcessTime=pPrevCraneCont->m_dStartProcessTime;
        }
    } else {
        if (pCont->m_dEndProcessTime<pPrevCraneCont->m_dStartLoadUnload) {
            pCont->m_dEndProcessTime=pPrevCraneCont->m_dStartLoadUnload;
        }
    }
}
pCont->m_dCompleteTime = pCont->m_dEndTime + pCont->m_dLoadUnload;
}
pCont->m_dSetupTime = dSetupTime;
pCont->m_bProcessStarted = true;
pShip = pBatch->m_pShip + pCont->m_lShip;
if (pShip->m_dT2 < pCont->m_dCompleteTime)
{
    pShip->m_dT2 = pCont->m_dCompleteTime;
}
return true;
}

///////////////////////////////////////////////////////////////////////////////
// CAssignJob::assignCont
// Parameters:
//  pBatch - pointer to batch object
// Return:   success of the method
// Pre-condition:
// Pointer pBatch is not null
// Post-condition:
// Remarks:
// This method loops through the list of job for shore-crane and yard machines, and
// calculates the start/end processing time, start loading/unloading time, etc.
/////////////////////////////////////////////////////////////////////////////////
bool CAssignJob::assignCont(CBatch *pBatch)
{
    long lMach;
    long lCrane;
    long lJobNo;
    bool bCont=true, bSuccess=true;
    double dT2=0.0;

    CContList::iterator *pMachContListIt = new CContList::iterator[pBatch->m_lMachine];
    CContList::iterator machIt;

    CContMap::iterator *pCraneContMapIt = new CContMap::iterator[pBatch->m_lCrane];
    CContMap::iterator craneContIt;

    for (lMach=0L; lMach<pBatch->m_lMachine; ++lMach)
    {
        pMachContListIt[lMach] = (pBatch->m_pMachContList[lMach]).begin();
    }

    for (lCrane=0L; lCrane<pBatch->m_lCrane; ++lCrane)
    {
        pCraneContMapIt[lCrane] = (pBatch->m_pCraneContMap[lCrane]).begin();
    }

    bCont= true;
    while (bCont)
    {
        bSuccess = true;
        for (lMach=0L; lMach<pBatch->m_lMachine; ++lMach)
        {
            pMachContListIt[lMach] = (pBatch->m_pMachContList[lMach]).begin();
        }

        for (lCrane=0L; lCrane<pBatch->m_lCrane; ++lCrane)
        {
            pCraneContMapIt[lCrane] = (pBatch->m_pCraneContMap[lCrane]).begin();
        }

        bCont= true;
        while (bCont)
        {
            // Assign container to batch
            // Calculate start/end processing time, start loading/unloading time, etc.
            // Update batch information
            // Repeat until all containers are assigned
        }
    }

    return bSuccess;
}
bCont=false;
bSuccess=false;

for (lCrane=0L; lCrane<pBatch->m_lCrane;++lCrane)
{
  craneContIt = pCraneContMapIt[lCrane];
  if (craneContIt!=pBatch->m_pCraneContMap[lCrane].end())
  {
    //still have jobs available
    bCont = true;
    pCont = (*craneContIt).second;
    lJobNo = pCont->m_lJobNo;
    lMach = pCont->m_lMachine;

    pPrevMachCont=pCont->m_pPrevCont;
    pPrevCraneCont=pBatch->m_pCraneContMap[lCrane].getPrevCont(lJobNo);

    if (processCraneJob(pBatch, lCrane, pCont, pPrevCraneCont,
                        pPInterMachCont))
    {
      ++craneContIt;
      pCraneContMapIt[lCrane] = craneContIt;
      bSuccess = true;
    }
  }
}

for (lMach=0L; lMach<pBatch->m_lMachine;++lMach)
{
  machIt = pMachContListIt[lMach];
  if (machIt!=pBatch->m_pMachContList[lMach].end())
  {
    //still have jobs available
    bCont = true;

    pCont = (*machIt);
    lJobNo = pCont->m_lJobNo;
    lCrane = pCont->m_lCrane;

    pPrevMachCont=pCont->m_pPrevCont;
    pPrevCraneCont=pBatch->m_pCraneContMap[lCrane].getPrevCont(lJobNo);

    if (processMachJob(pBatch, lCrane, lMach, pCont, pPrevCraneCont,
                        pPInterMachCont))
    {
      pCont = (*machIt);
      ++machIt;
      pMachContListIt[lMach] = machIt;
      bSuccess = true;
    }
  }
}

if (bCont && !bSuccess)
{
  //there is a deadlock. There should be at least one job ready to be processed,
  // but the previous QC job have not been complete yet
}
CContainer *pPrevCont = NULL;
CContainer *pTempCont;
CContList::iterator tempMachIt;

bool bFound=false;
long lSwapType = 0L;    //0-not found; 1 - same YM; 2- diff YM
long lTempMach = pPrevCraneCont->m_lMachine;

pCont=NULL;
pPrevCraneCont=NULL;
for (lMach=0L; lMach<pBatch->m_lMachine;++lMach)
{
    machIt = pMachContListIt[lMach];
    if (machIt!=(pBatch->m_pMachContList[lMach].end()))
    {
        pCont = (*machIt);

        pPrevCraneCont = pCont->m_pPrevCraneCont;
        if (pPrevCraneCont!=NULL && (pPrevCraneCont->m_bProcessStarted))
        {
            //find the previous job for the QC that has not been processed
            while (pPrevCraneCont!=NULL)
            {
                if (pPrevCraneCont->m_bImport)
                {
                    if (pPrevCraneCont->m_bLoadUnloadComplete
                        && !(pPrevCraneCont->m_bProcessStarted))
                    {
                        bFound = true;
                        break;
                    }
                }
                else
                {
                    if (!((pPrevCraneCont->m_bProcessStarted))
                    {
                        //the previous job of the QC has been
                        //processed
                        pTempCont
                        = pPrevCraneCont->m_pPrevCraneCont;
                        if (pTempCont==NULL
                            || pTempCont->m_bLoadUnloadComplete)
                        {
                            bFound=true;
                            break;
                        }
                    }
                }
            }
        }
        if (bFound)
        {
            break;
        }
    }
} //END FOR MACHINE
if (!bFound)
{
    std::cout<<"error\n";
}

lTempMach = pPrevCraneCont->m_lMachine;

if (lTempMach==lMach)
{
    tempMachIt = machIt;
    while ((*tempMachIt)!=pPrevCraneCont)
    {
        ++tempMachIt;
    }  //swap machine process order
    if (pCont->m_pNextCont==pPrevCraneCont)
    {
        //pCont is followed by pPrevCraneCont - now reverse order
        pTempCont = pPrevCraneCont->m_pNextCont;
        pCont->m_pNextCont = pTempCont;
        if (pTempCont!=NULL)
        {
            pTempCont->m_pPrevCont = pCont;
        }
        pTempCont = pCont->m_pPrev;
        pPrevCraneCont->m_pPrevCont = pTempCont;
        if (pTempCont!=NULL)
        {
            pTempCont->m_pNextCont = pPrevCraneCont;
        }
    }
    pCont->m_pPrevCont = pPrevCraneCont;
    pPrevCraneCont->m_pNextCont = pCont;
}
else if (pCont->m_pPrevCont==pPrevCraneCont)
{
    //not possible unless there is a buffer
    assert(false);
}
else
{
    pTempCont = pCont->m_pNextCont;
    pCont->m_pNextCont = pPrevCraneCont->m_pNextCont;
    pPrevCraneCont->m_pNextCont = pTempCont;
    if (pTempCont!=NULL)
    {
        pTempCont->m_pPrevCont = pPrevCraneCont;
    }
    pTempCont = pCont->m_pNextCont;
    if (pTempCont!=NULL)
    {
        pTempCont->m_pPrevCont = pCont;
    }
    pTempCont = pCont->m_pPrev;
    pPrevCraneCont->m_pPrevCont = pTempCont;
}
pPrevCraneCont->m_pPrevCont = pTempCont;
if (pTempCont!=NULL)
{
    pTempCont->m_pNextCont = pPrevCraneCont;
}
pTempCont = pCont->m_pPrevCont;
if (pTempCont!=NULL)
{
    pTempCont->m_pNextCont = pCont;
}
else
{
    tempMachIt = pMachContListIt[lTempMach];
    while (tempMachIt!=(pBatch->m_pMachContList[lTempMach].end()))
    {
        pTempCont = (*tempMachIt);
        if ((pTempCont)==pPrevCraneCont)
        {
            break;
        }
        ++tempMachIt;
    }
    assert(tempMachIt!=(pBatch->m_pMachContList[lTempMach].end()));

    //swap the machines
    pCont->m_lMachine = lTempMach;
    pPrevCraneCont->m_lMachine = lMach;

    //swap machine process order
    pTempCont = pCont->m_pPrevCont;
    pCont->m_pPrevCont = pPrevCraneCont->m_pPrevCont;
    pPrevCraneCont->m_pPrevCont = pTempCont;

    pTempCont = pCont->m_pNextCont;
    pCont->m_pNextCont = pPrevCraneCont->m_pNextCont;
    pPrevCraneCont->m_pNextCont = pTempCont;

    //realign to prev and next cont
    pTempCont=pCont->m_pPrevCont;
    if (pTempCont!=NULL)
    {
        pTempCont->m_pNextCont = pCont;
    }
    pTempCont=pPrevCraneCont->m_pPrevCont;
    if (pTempCont!=NULL)
    {
        pTempCont->m_pNextCont = pPrevCraneCont;
    }

    pTempCont=pCont->m_pNextCont;
    if (pTempCont!=NULL)
    {
        pTempCont->m_pPrevCont = pCont;
    }
    pTempCont=pPrevCraneCont->m_pNextCont;
if (pTempCont! = NULL) {
    pTempCont->m_pPrevCont = pPrevCraneCont;
}

// swap containers of the list
(*tempMachIt)=pCont;
(*machIt) = pPrevCraneCont;

assert(pBatch->validateMachineList());
}
}

delete [] pMachContListIt;
delete [] pCraneContMapIt;

return true;
}
Appendix C. Selected Computer Code for validating CPM

```cpp
#include "StdAfx.h"
#include <iostream>
#include "param.h"
#include "constraint.h"

CConstraint::CConstraint(CBatch *pBatch)
{
    m_pBatch = pBatch;
}

CConstraint::~CConstraint(void)
{
    CShip *pShip = m_pBatch->m_pShip;
}

bool CConstraint::C2(void)
{
    long lShip;
    CShip *pShip = m_pBatch->m_pShip;
    for (lShip=0L; lShip<m_pBatch->m_lShip; ++lShip)
    {
        if (pShip->m_dT1 + SMALL_NO<pShip->m_dSTA)
        {
            return false;
        }
        ++pShip;
    }
    return true;
}

bool CConstraint::C3(void)
{
    long lShip = 0L;
    long lCont;
    CContainer *pCont;
    CShip *pShip = m_pBatch->m_pShip;
    pCont = m_pBatch->m_pCont;
    for (lCont=0L; lCont<m_pBatch->m_lCont; ++lCont)
    {
        pShip = m_pBatch->m_pShip + pCont->m_lShip;
        if (pCont->m_dStartLoadUnload+SMALL_NO<pShip->m_dATA)
        {
            return false;
        }
        ++pCont;
    }
```
bool CConstraint::C4(void)
{
    long lShip = 0L;
    long lCont;
    CContainer *pCont;
    CShip *pShip = m_pBatch->m_pShip;
    double dTemp;

    pCont = m_pBatch->m_pCont;
    for (lCont=0L;lCont<m_pBatch->m_lCont; ++lCont)
    {
        pShip = m_pBatch->m_pShip + pCont->m_lShip;

        dTemp = pCont->m_dStartLoadUnload+ pCont->m_dLoadUnload;

        if ((pShip->m_dT2+SMALL_NO)<dTemp)
        {
            return false;
        }

        ++pCont;
    }

    return true;
}

bool CConstraint::C5_8(void)
{
    long lShip = 0L;
    long lCont;
    long lMach;
    CContainer *pCont, *pPrevCont, *pNextCont;
    CContMap::iterator it;

    pCont = m_pBatch->m_pCont;

    for (lCont=0L;lCont<m_pBatch->m_lCont;++lCont)
    {
        lMach=pCont->m_lMachine;
        //each container must be assigned with one machine
        if (lMach<0L || lMach>=m_pBatch->m_lMachine)
        {
            return false;
        }

        //Validate the list of containers processed by the same YM
        pPrevCont = pCont->m_pPrevCont;
        if (pPrevCont!=NULL && (lMach!=pPrevCont->m_lMachine))
        {
            return false;
        }
    }

    return true;
}
pNextCont = pCont->m_pPrevCont;
if (pNextCont!=NULL && (lMach!=pNextCont->m_lMachine))
{
    return false;
}
++pCont;
return true;
}

bool CConstraint::C9(void)
{
    long lCrane;
    CContainer *pCont,*pPrevCont;
    CContMap *pCraneContMap;
    CContMap::iterator it;

    pCraneContMap = m_pBatch->m_pCraneContMap;
    for (lCrane=0L; lCrane<m_pBatch->m_lCrane; ++lCrane)
    {
        pPrevCont=NULL;
        pCont=NULL;
        it = pCraneContMap->begin();
        if (it!=pCraneContMap->end())
        {
            pCont=(*it).second;
            ++it;
        }
        while (it!=pCraneContMap->end())
        {
            pPrevCont = pCont;
            pCont = (*it).second;
            if ((pCont->m_dStartLoadUnload+SMALL_NO)
                <(pPrevCont->m_dStartLoadUnload + pPrevCont->m_dLoadUnload))
            {
                return false;
            }
            ++it;
        }
        ++pCraneContMap;
    }
    return true;
}

bool CConstraint::C10(void)
{
    long lCont;
    CContainer *pCont;
    CContList::iterator it;

    pCont = m_pBatch->m_pCont;
    for (lCont=0L;lCont<m_pBatch->m_lCont; ++lCont)
    {
        if (pCont->m_bImport
            && (pCont->m_dStartProcessTime+SMALL_NO<
                (pCont->m_dStartLoadUnload+pCont->m_dLoadUnload )))
        {  
            return false;
        }
        ++it;
        ++pCraneContMap;
    }
    return true;
}
bool CConstraint::C11(void)
{
    long lCont;
    CContainer *pCont;
    CContList::iterator it;

    pCont = m_pBatch->m_pCont;
    for (lCont=0L;lCont<m_pBatch->m_lCont; ++lCont)
    {
        if (!(pCont->m_bImport)
            && ((pCont->m_dStartLoadUnload + SMALL_NO)<
                pCont->m_dEndProcessTime ))
        {
            std::cout<<"Job,Start L/U,End Time\n";
            std::cout<<pCont->m_lJobNo<<","<<
                pCont->m_dStartLoadUnload<<","<<pCont->m_dEndProcessTime<<"\n";
            return false;
        }
        ++pCont;
    }
    return true;
}

bool CConstraint::C12(void)
{
    long lShip = 0L;
    long lCrane;
    CContainer *pCont, *pPrevCont;
    CContMap *pCraneContMap;
    CContMap::iterator it;

    pCraneContMap = m_pBatch->m_pCraneContMap;
    for (lCrane=0L;lCrane<m_pBatch->m_lCrane; ++lCrane)
    {
        pCont = NULL;
        pPrevCont = NULL;
        for (it=pCraneContMap->begin();it!=pCraneContMap->end();++it)
        {
            pPrevCont = pCont;
            pCont = (*it).second;
            if (pPrevCont!=NULL)
            {
                if (pCont->m_bImport && pPrevCont->m_bImport)
                {
                    if ((pCont->m_dStartProcessTime + SMALL_NO)
                        <(pPrevCont->m_dStartLoadUnload+pPrevCont->m_dLoadUnload))
                    {
                        return false;
                    }
                }
            }
        }
    }
}
bool CConstraint::C13(void)
{
    long lShip = 0L;
    long lCrane;
    CContainer *pCont, *pPrevCont;
    CContMap *pCraneContMap;
    CContMap::iterator it;

    pCraneContMap = m_pBatch->m_pCraneContMap;
    for (lCrane=0L;lCrane<m_pBatch->m_lCrane; ++lCrane)
    {
        pCont = NULL;
        pPrevCont = NULL;
        for (it=pCraneContMap->begin();it!=pCraneContMap->end();++it)
        {
            pPrevCont = pCont;
            pCont = (*it).second;
            if (pPrevCont!=NULL)
            {
                if (pPrevCont->m_bImport && !(pCont->m_bImport))
                {
                    if ((pCont->m_dEndProcessTime+ SMALL_NO)<
                        pPrevCont->m_dStartProcessTime )
                    {
                        return false;
                    }
                }
            }
        }
        ++pCraneContMap;
    }
    return true;
}

bool CConstraint::C14(void)
{
    long lShip = 0L;
    long lCrane;
    CContainer *pCont, *pPrevCont;
    CContMap *pCraneContMap;
    CContMap::iterator it;

    pCraneContMap = m_pBatch->m_pCraneContMap;
    for (lCrane=0L;lCrane<m_pBatch->m_lCrane; ++lCrane)
    {
        pCont = NULL;
        pPrevCont = NULL;
        for (it=pCraneContMap->begin();it!=pCraneContMap->end();++it)
        {
            pPrevCont = pCont;
            pCont = (*it).second;
            if (pPrevCont!=NULL)
            {
                if (pPrevCont->m_bImport && !(pCont->m_bImport))
                {
                    if ((pCont->m_dEndProcessTime+ SMALL_NO)<
                        pPrevCont->m_dStartProcessTime )
                    {
                        return false;
                    }
                }
            }
        }
        ++pCraneContMap;
    }
    return true;
}
pPrevCont = NULL;
for (it=pCraneContMap->begin();it!=pCraneContMap->end();++it)
{
    pPrevCont = pCont;
    pCont = (*it).second;
    if (pPrevCont!=NULL)
    {
        if (!(pPrevCont->m_bImport) && !(pCont->m_bImport))
        {
            if ((pCont->m_dEndProcessTime + SMALL_NO)<
                  pPrevCont->m_dStartLoadUnload)
            {
                std::cout<<"ID: "<<pCont->m_lID<<","<<pPrevCont->m_lID<<"\n";
                std::cout<<"End process time: "<<pCont->m_dEndProcessTime<<"\n";
                std::cout<<"Start loading time: "<<pPrevCont->m_dStartLoadUnload<<"\n";
                return false;
            }
        }
    }
    ++pCraneContMap;
}
return true;

bool CConstraint::C15(void)
{
    long lCont = 0L;
    CContainer *pCont = m_pBatch->m_pCont;

    for (lCont=0L;lCont<m_pBatch->m_lCont; ++lCont)
    {
        if ((pCont->m_dEndProcessTime + SMALL_NO)<
                  pCont->m_dStartProcessTime)
        {
            return false;
        }
    }
    return true;
}

bool CConstraint::C16(void)
{
    long lCont = 0L;
    long lMachine = 0L;
    CContainer *pCont, *pPrevCont;
    double dSetupTime;

    pPrevCont=NULL;
pCont=m_pBatch->m_pCont;
for (lCont=0L;lCont<m_pBatch->m_lCont; ++lCont)
{

pPrevCont = pCont->m_pPrevCont;
IMachine = pCont->m_lMachine;
dSetupTime = m_pBatch->getSetupTime(IMachine, pPrevCont, pCont);
if ((pPrevCont!=NULL) && (pCont->m_dStartProcessT ime + SMALL_NO) <
    (pPrevCont->m_dEndProcessTime + dSetupTime))
    {
        return false;
    }

++pCont;
return true;

bool CConstraint::checkConstraints(void)
{
    bool bSuccess = true;
    if (!C2())
    {
        bSuccess=false;
        std::cout<<"Error in C2\n";
    }
    if (!C3())
    {
        bSuccess=false;
        std::cout<<"Error in C3\n";
    }
    if (!C4())
    {
        bSuccess=false;
        std::cout<<"Error in C4\n";
    }
    if (!C5_8())
    {
        bSuccess=false;
        std::cout<<"Error in C5, C6, C7 or C8\n";
    }
    if (!C9())
    {
        bSuccess=false;
        std::cout<<"Error in C9\n";
    }
    if (!C10())
    {
        bSuccess=false;
        std::cout<<"Error in C10\n";
    }
    if (!C11())
    {
        bSuccess=false;
        std::cout<<"Error in C11\n";
    }
if (!C12())
{
    bSuccess=false;
    std::cout<<"Error in C12\n";
}

if (!C13())
{
    bSuccess=false;
    std::cout<<"Error in C13\n";
}

if (!C14())
{
    bSuccess=false;
    std::cout<<"Error in C14\n";
}

if (!C15())
{
    bSuccess=false;
    std::cout<<"Error in C15\n";
}

if (!C16())
{
    bSuccess=false;
    std::cout<<"Error in C16\n";
}

return true;
Appendix D. An example on the finding lower-bound of ship delays
References


