



Department of Computer Science

**Dynamic Scheduling of Automated
Guided Vehicles in Container
Terminals**

Hassan Rashidi Haramabadi

A thesis submitted for the degree of PhD

Date of conferment: 27 April 2006

Supervisory Board:

Prof. Edward. P. K. Tsang

(Supervisor)

Dr. John Ford

Prof. Huosheng Hu

Abstract

The growths of containerization and transporting goods in containers have created many problems for ports. In this thesis, five scheduling decisions in the container terminals are defined and formulated as Constraint Satisfaction Optimisation Problems (CSOPs). For each of the decisions, an overview of literature is presented.

The objective of this thesis is to develop efficient and effective algorithms to solve the scheduling problem of Automated Guided Vehicles (AGV) in the port. This problem is formulated as a Minimum Cost Flow (MCF) model, which is a directed graph. Then, the model is tackled by the Network Simplex Algorithm (NSA) and its extensions in both static and dynamic aspects. These extensions are Network Simplex plus Algorithm (NSA+), Dynamic Network Simplex Algorithm (DNSA) and Dynamic Network Simplex plus Algorithm (DNSA+). To solve the problems, NSA and NSA+ start from scratch whereas DNSA and DNSA+ repair solutions when the changes occur.

In static problems (where there is no change in the situation), NSA and NSA+ can find the global optimal solutions for 3,000 jobs and ten millions arcs in the graph model within two minutes on a 2.4 GHz Pentium PC. Due to the efficiency of DNSA and DNSA+ (compared with NSA and NSA+), these algorithms are applied to dynamic problems in which the graph changes.

Although NSA and its extensions are efficient, they can only work on problems with certain limits in size. When the size of the problem goes beyond the limits, incomplete search methods are used. To complement the above algorithms, a greedy method (Greedy Vehicle Search-GVS) is designed and implemented. This incomplete search method can be applied to both static and dynamic problems.

Acknowledgements

I would like to thank my supervisor, Prof. Edward Tsang, who proposed me this topic for my research. He enormously helped in the refinement of the ideas presented in this thesis by his constant feedback and supervision. I have no doubt that I would have never achieved without Tsang's guidance and encouragement. I am grateful to Dr. John Ford for his suggestion on development Dynamic Network Simplex Algorithm and his comments on this research. I would like to thank Prof. Hu, the head of Robotics research group at University of Essex, who gave me a few suggestions on my software.

I would also like to thank my family, especially my wife (Fatema), who lived in my country without me, supported my children and encouraged me to do this research at University of Essex.

I would like to thank the Department of Computer Science for the harmonious environment and the Computing Service at University of Essex for the excellent computer facilities to perform the computational experiments reported here. I am grateful to Department of Electronic System Engineering at University of Essex that provided a few chances for PhD Students to demonstrate their research results and to get the academic staff's views.

I also wish to thank my external examiner, Dr. Sanja Petrovic from the School of Computer Science and IT at the University of Nottingham, for examining my senate viva and also giving me some valuable comments. I would like to thank my internal examiner, Dr Klaus McDonald-Maier, for his feedbacks and helping me through the final version of this thesis.

Table of Contents

List of Tables.....	viii
List of Figures	ix
List of Abbreviations	xi
Papers Published and Submitted	vii
Chapter 1: Introduction	1
Chapter 2: Problem Description and Decisions to be made	5
2.1 Compartments.....	5
2.2 Operations.....	8
2.3 Decisions to be made	9
2.3.1 Allocation of berths to arriving vessels and QCs to docked vessels	10
2.3.2 Storage space assignment	11
2.3.3 Rubber Tyred Gantry Crane (RTGC) deployment	11
2.3.4 Scheduling and routing of vehicles	11
2.3.5 Appointment times to eXternal Trucks (XTs).....	11
Chapter 3: Literature Review and Formulation of the Decisions	12
3.1 Allocation of berths to arriving vessels and quay cranes to docked vessels.....	12
3.1.1 Assumptions	14
3.1.2 Decision variables and domains	16
3.1.3 Constraints.....	16
3.1.4 Objective function	17
3.2 Storage space assignment	18
3.2.1 Assumptions	19
3.2.2 Decision variables and domains	22
3.2.3 Constraints.....	23
3.2.4 Objective function	25
3.3 Rubber Tyred Gantry Crane (RTGC) deployment	26
3.3.1 Assumptions	26
3.3.2 Decision variables and domains	28
3.3.3 Constraints.....	29
3.3.4 Objective function	29
3.4 Scheduling and routing of vehicles.....	30
3.4.1 Assumptions	32
3.4.2 Decision variables and domains	33
3.4.3 Constraints.....	33
3.4.4 Objective function	35
3.5 Appointment times to eXternal Trucks (XTs).....	36
3.5.1 Assumptions	37
3.5.2 Decision variables and domains	38
3.5.3 Constraints.....	38
3.5.4 Objective function	38
3.6 Container terminals over the world, a survey	39
3.7 Solution methods and evaluation of the decisions	40
3.8 Summary and conclusion.....	49

Chapter 4: Scheduling of AGVs and Its Problem Formulation	50
4.1 Reasons to choose this problem.....	50
4.2 Assumptions.....	51
4.3 Variables and notations.....	55
4.4 The Minimum Cost Flow model.....	58
4.4.1 Graph terminology.....	58
4.4.2 The standard form of the minimum cost flow model.....	59
4.5 The special case of the MCF model for Automated Guided Vehicles Scheduling.....	60
4.5.1 Nodes and their properties in the special graph.....	61
4.5.2 Arcs and their properties in the special graph.....	62
4.5.3 The MCF-AGV model for the Automated Guided Vehicles Scheduling.....	64
4.6 Summary and conclusion.....	66
Chapter 5: Network Simplex Algorithm and Static Scheduling of AGVs	67
5.1 Reasons to choose NSA.....	67
5.2 The Network Simplex Algorithm.....	67
5.2.1 Spanning tree solutions and optimality conditions.....	68
5.2.2 The algorithm NSA.....	71
5.2.3 The difference between NSA and original simplex.....	73
5.2.4 A short literature over pricing rules.....	74
5.2.5 Strongly feasible spanning tree.....	75
5.3 Simulation software.....	77
5.3.1 The features of our software.....	77
5.3.2 The implementation of NSA in our software.....	81
5.3.3 How the program works.....	83
5.3.4 The circulation problem.....	85
5.4 Experimental results.....	86
5.5 An estimate of the algorithm's complexity in practice.....	89
5.6 Limitation of the NSA in practice.....	92
5.7 Summary and conclusion.....	92
Chapter 6: Network Simplex plus Algorithm and Dynamic Scheduling of AGVs.....	93
6.1 Motivation.....	93
6.2 The Network Simplex plus Algorithm (NSA+).....	93
6.2.1 Anti-Cycling in NSA+.....	93
6.2.2 Memory technique and Heuristic approach in NSA+.....	94
6.2.3 The differences between NSA and NSA+.....	95
6.3 A comparison between NSA and NSA+.....	96
6.4 Statistical test for the comparison.....	98
6.5 Complexity of Network Simplex plus Algorithm (NSA+).....	99
6.6 Software architecture for dynamic aspect.....	100
6.7 Experimental results from the dynamic aspect.....	103
6.8 Summary and conclusion.....	105
Chapter 7: Dynamic Network Simplex Algorithms and Dynamic Scheduling of AGVs	106
7.1 Motivation.....	106
7.2 Classification of graph algorithms.....	107
7.3 The Dynamic Network Simplex Algorithm.....	107
7.3.1 Data structures.....	107
7.3.2 Memory management.....	111
7.3.3 The algorithms DNSA and DNSA+.....	112
7.4 Software architecture for dynamic aspect.....	120
7.5 A comparison between DNSA+ and NSA+.....	121
7.6 Statistical test for the comparison.....	123

7.7 Complexity of the algorithm.....	123
7.8 Summary and conclusion.....	124
Chapter 8: Greedy Vehicle Search and Dynamic Scheduling of AGVs	125
8.1 Motivation.....	125
8.2 Problem formalization	125
8.2.1 Nodes and their properties in the incomplete graph	126
8.2.2 Arcs and their properties in the incomplete graph.....	127
8.2.3 The special case of the MCF-AGV model for Automated Guided Vehicles Scheduling	128
8.3 Algorithm formalization	129
8.4 Software architecture for dynamic aspect.....	130
8.5 A comparison between GVS and NSA+ and quality of the solutions	131
8.6 Statistical test for the comparison	133
8.7 Complexity of Greedy Vehicle Search	134
8.7.1 Complexity of GVS for static problem.....	134
8.7.2 Complexity of GVS for dynamic problem	136
8.8 A discussion over GVS and meta-heuristic	136
8.9 Summary and conclusion.....	137
Chapter 9: Conclusions and Future Research	138
9.1 Summary of work done.....	138
9.2 Observations and conclusions.....	142
9.3 Research contributions.....	143
9.4 Future research.....	144
9.4.1 Scheduling and routing of the vehicles.....	144
9.4.2 Economic and optimization model	145
9.4.3 Other possible extension.....	146
Appendix: Information on Web	148
References	154
Index	164

List of Tables

Table 3-1: Container Terminals around the world and their decisions	39
Table 3-2: Considerations in choosing between major scheduling techniques [94].....	42
Table 3-3: Summary of Vehicle Routing Problems and Solutions [32].....	43
Table 3-4: Summary of work reviews for AGVs in General Path Topologies [79].....	45
Table 3-5: Summary of Algorithms for AGVs in Specific Path Topologies [79].....	45
Table 3-6: Summary of Static and Dynamic Routing Algorithms for AGVs in General Path Topology [79].....	46
Table 3-7: Some important indices to evaluate the decisions in the container terminals	49
Table 4-1: Example of traveling time (second) between two different points in the port	53
Table 4-2: Appointment time of containers jobs.....	54
Table 5-1: Values of parameters for the simulation	87
Table 5-2: Experimental results of Network Simplex Algorithm in static fashion	87
Table 5-3: Regression result for CPU-Time required to solve the problem by NSA (Based on the number of jobs)	90
Table 5-4: Regression result for CPU-Time required to solve the problem by NSA (Based on the number of arcs)	90
Table 6-1: Experimental results for a comparison between NSA and NSA+	96
Table 6-2: The result of T-Test for the two algorithms, NSA and NSA+	99
Table 7-1: Memory allocation for the arcs of the MCF-AGV model and its algorithm.....	112
Table 7-2: The result of T-Test for the two algorithms, DNSA+ and NSA+	123
Table 7-3: A comparison between NSA and its extensions	124
Table 8-1: The result of T-Test for the two algorithms, GVS and NSA+	133
Table 8-2: Regression result for CPU-Time required to finding a local optimum by GVS for static problem	135
Table 9-1: A summary of the algorithms studied in this thesis for the MCF-AGV model.....	141

List of Figures

Figure 1-1: The number of containers turnover in the ten largest container terminals over the world [87].	2
Figure 2-1: The container storage area in a port [68].	5
Figure 2-2: An RTGC sits across the width of a block [68].	6
Figure 2-3: Transfer of a RTGC between two blocks [107].	6
Figure 2-4: A typical quay crane [68].	7
Figure 2-5: A Straddle Carrier (left) and an Automated Guided Vehicle (right) while they are carrying a container.	8
Figure 2-6: Scheduling Decisions in the container terminals.	10
Figure 3-1: Park and Kim's two phases scheduling of berths and cranes [73].	14
Figure 3-2: An output of the berth and crane scheduling problem.	15
Figure 3-3: Port's layout with the primary and secondary storages [90].	20
Figure 3-4: Cross over problem for two RTGCs in the storage area.	28
Figure 3-5: Flow of outbound containers (SA = Storage Area, QS = Quayside).	36
Figure 3-6: Flow of inbound containers (SA = Storage Area, QS = Quayside).	36
Figure 4-1: Layout of the container terminal.	52
Figure 4-2: Phenomena arising in scheduling and routing of AGVs [79].	53
Figure 4-3: Travelling time computations between the next location of vehicle and the next job.	57
Figure 4-4: Travelling time computations between job i and job j .	57
Figure 4-5: An example of the MCF-AGV model for 2 AGVs and 4 jobs.	65
Figure 5-1: A feasible spanning tree solution (dotted).	68
Figure 5-2: The Network Simplex Algorithm.	71
Figure 5-3: An example of strongly feasible spanning tree [2].	76
Figure 5-4: The main screenshot of the software.	78
Figure 5-5: Relationships between the tables of the Database.	80
Figure 5-6: Flowchart of Network Simplex Algorithm (<i>Block Pricing Scheme</i>) to select an entering arc.	82
Figure 5-7: An example of the MCF-AGV model for 2 AGVs and 2 jobs in our software.	84
Figure 5-8: The input of the algorithm (NSA) in DIMACS format.	84
Figure 5-9: The output of the algorithm (NSA) in DIMACS format.	85
Figure 5-10: An example of the circulation problem ($P = \text{Penalty}$).	86
Figure 5-11: CPU-Time required to solve the problem by Network Simplex Algorithm, based on the number of jobs.	88
Figure 5-12: CPU-Time required to solve the problem by Network Simplex Algorithm, based on the number of arcs.	88
Figure 6-1: Flowchart of Network Simplex plus Algorithm to select an entering arc.	95
Figure 6-2: A comparison of CPU-Time required to solve the same problems by NSA and NSA+.	97
Figure 6-3: The T-Test acceptance and reject regions (NSA and NSA+H).	99
Figure 6-4: Block diagram of the software and algorithm (NSA+) for dynamic aspect.	100
Figure 6-5: Operations of the software in dynamic aspect.	102

Figure 6-6: An experimental result from the dynamic scheduling problem of AGVs (NSA+ solved the problem).	103
Figure 6-7: The attributes of the carried jobs in the dynamic scheduling problem of AGVs.	104
Figure 7-1: A sample of the spanning tree and its attributes (FIX='FIXED', UFD='UNFIXED').	109
Figure 7-2: The Dynamic Network Simplex Algorithm.	113
Figure 7-3: The pseudo code of reconstructing the spanning tree in Dynamic Network Simplex Algorithm.	113
Figure 7-4: The pseudo code of removing a node from the spanning tree in Dynamic Network Simplex Algorithm.	114
Figure 7-5: The new spanning tree after removing nodes 8 (See Figure 7-1).	115
Figure 7-6: The new spanning tree after removing node 3 (See Figure 7-1).	116
Figure 7-7: The new spanning tree after removing node 4 (See Figure 7-6).	117
Figure 7-8: The pseudo code of inserting a node into the spanning tree in Dynamic Network Simplex Algorithm.	118
Figure 7-9: The new spanning tree after inserting node 9 and 10 (See Figure 7-1).	119
Figure 7-10: Block diagram of the software and algorithm (DNSA+) in the dynamic aspect	120
Figure 7-11: A comparison of the number of iterations in DNSA+ and NSA+	122
Figure 8-1: An example of the incomplete case of the MCF-AGV model with two AGVs and four jobs.	125
Figure 8-2: The block diagram of Greedy Vehicle Search.	129
Figure 8-3: The pseudo code of Greedy Vehicle Search in dynamic aspect	129
Figure 8-4: The block diagram of the software and algorithm (GVS) in dynamic aspect.	130
Figure 8-5: A comparison of NSA+ and GVS for Travelling and Waiting Times of the Vehicles.	131
Figure 8-6: The number of carried jobs by NSA+ and GVS during 6 hour simulation.	132
Figure 8-7: A comparison of NSA+ and GVS for the Average Lateness from the appointment time	133
Figure 8-8: CPU-Time required to solve the static problems by GVS.	134
Figure 8-9: CPU-Time required to solve the dynamic problems by GVS.	136

List of Abbreviations

Abbreviation	Term / Meaning
AGV	Automated Guided Vehicle
BDE	Borland Database Engine
CSOPs	Constraint Satisfaction Optimisation Problems.
DNSA	Dynamic Network Simplex Algorithm
DNSA+	Dynamic Network Simplex plus Algorithm
DSSAGV	Dynamic Scheduling Software for Automated Guided Vehicles
ERD	Entity Relationship Diagram
GVS	Greedy Vehicle Search
HOTFRAME	Heuristic OpTImisation FRAMEwork
IT	Internal Trucks
MCF	Minimum Cost Flow
MCF-AGV	Minimum Cost Flow model for Scheduling problem of AGVs
NSA	Network Simplex Algorithm
NSA+	Network Simplex Plus Algorithm.
OSA	Original Simplex Algorithm
PSCDS	PSCDS: Primary Storage Containers Discharge
PSCPI	PSCPI: Primary Storage Containers Pickup
PSCSS	Primary Storage Containers to Secondary Storage
QC	Quay Cranes
RTGC	Rubber Tyred Gantry Cranes
SAM	Simulated Annealing Method
SC	Straddle Carrier
SDSAGV	Static and Dynamic Scheduling of Automated Guided Vehicles
SSCGD	Secondary Storage Containers Grounding
SSCPI	Secondary Storage Containers for Pickup
SSCPS	Secondary Storage Containers to Primary Storage
TG	Terminal Gate
TSS	Taxi Service System
VRP	Vehicle Routing Problem
VRPTW	Vehicle Routing Problem with Time Window
XT	eXternal Truck

Papers Published and Submitted

- 1- Rashidi H., and Tsang E.,” Applying the Extended Network Simplex Algorithm to Dynamic Automated Guided Vehicles Scheduling”, the 2nd Multidisciplinary International conference on Scheduling, Theory and Applications (MISTA), Volume 2, pp 677-692, New York, USA, 18-21 July 2005.
- 2- Rashidi H., and Tsang E.,” Applying the Extended Network Simplex Algorithm and a Greedy Search Method to Automated Guided Vehicle Scheduling”, submitted to the Journal of Annals of Operations Research.
- 3- Rashidi H., and Tsang E.,” Container Terminals: Scheduling Decisions, their Formulations and solutions”, submitted to Journal of Scheduling.
- 4- Rashidi H., and Tsang E.,” Dynamic Network Simplex Algorithm and its Application to Dynamic Scheduling of Automated Guided Vehicles”, to be submitted to Journal of Scheduling.
- 5- Rashidi H., and Tsang E.,” Extensions to Network Simplex Algorithm and its Application to the Static Automated Guided Vehicles Scheduling”, to be submitted to Journal of Scheduling.

Chapter 1: Introduction

There are more than 2,000 ports over the world. These ports play an important role in global manufacturing and international business, in where ships come to load and/or unload their cargos. The cargo ships can be classified into two types. The first type transports huge quantity of commodities like crude oil, coal, grains, etc. The second type usually carries goods that are packed into steel containers of standard sizes. This research concentrates on the second type, which attracted more attentions in both investment and automation during the last decade. The main functions of these terminals are delivering containers to consignees and receiving containers from shippers, loading containers onto and unloading containers from vessels and storing containers temporarily to account either for efficiency of the equipment or for the difference in arrival times of the sea and land carriers [107].

Since the 1960s, due to both the increasing containerisation (which means that the number of goods transported in containers has steadily been grown) and increasing world trade, new container terminals are being built and existing ones are extended. Today over 60% of the world's deep-sea general cargo is transported in containers, whereas some routes, especially between economically strong and stable countries, are containerized up to 100% [87]. Figure 1-1 shows the number of containers turnover for the ten largest container ports over the world from 1993 to 2002. As we can see in the figure in the Hong Kong terminal, containers turnover has been risen from 9 millions Twenty feet Equivalent Units (TEUs) in 1993 to 19 millions TEUs in 2002. In the same period, the number of TEUs in Singapore, as the second port around the world, has been increased from 9 millions to 17 millions TEUs. The greatest increase of the number of containers over the last decade is in Shanghai, China. In this port, the number of TEUs has increased from less than 1 million in 1993 to more than 8 millions in 2002. The figure also shows that the number of container handled in Hamburg, as a major port in Europe, grew up gradually during the last decade, increased from 2 millions TEUs in 1993 to more than 5 millions in 2002.

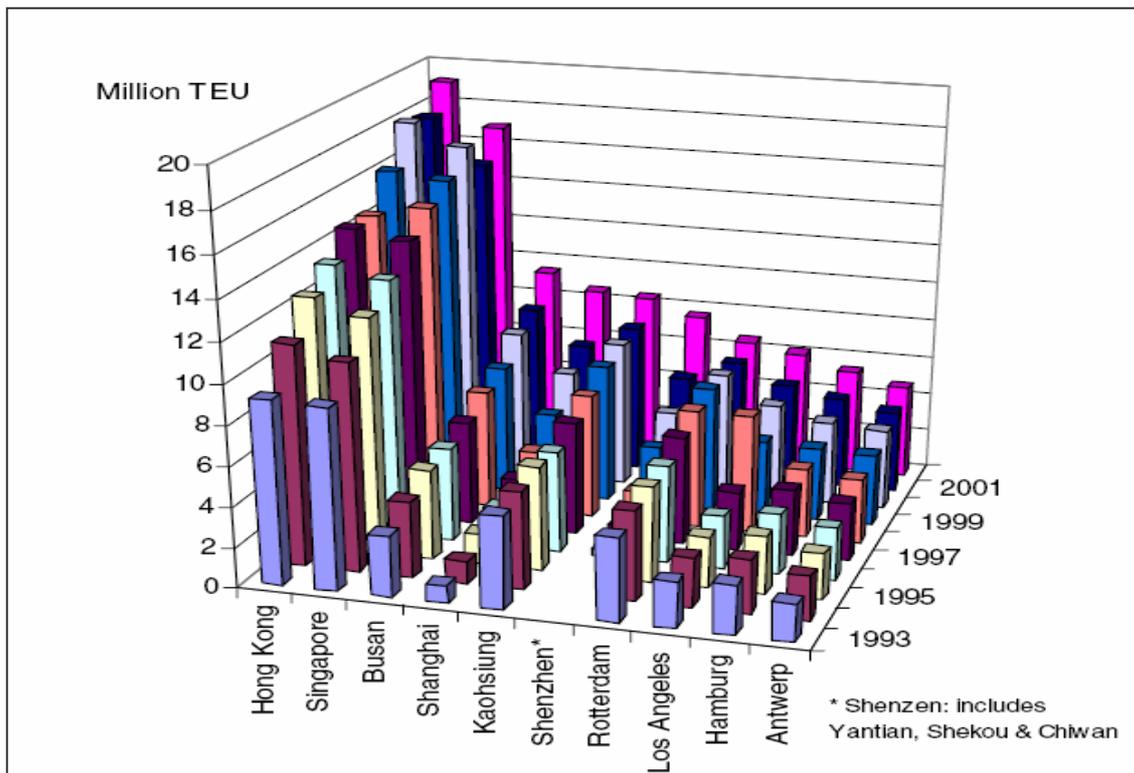


Figure 1-1: The number of containers turnover in the ten largest container terminals over the world [87].

The growths of containerization and transporting goods have created many problems for the container terminals. They face the challenge to cope with the growing number of containers. The rapid unloading/loading and turning around of ships has become an important problem in the container terminals. To meet these challenges, the container terminals have to innovate and often automate equipment and optimise their logistic processes. The main motivation for this research is to make a response to the challenges.

The remaining of this thesis is organized as follows. Chapters 2 and 3 provide a general framework and literature around the decisions in the container terminals. An outstanding matter from the literature review is that vehicle's problem is one of the challenging problems in the ports. Hence, the remaining chapters are dedicated to the automated guided vehicles scheduling. This research developed several algorithms for the problem in both static and dynamic aspects. In the static aspect, there is no change in the situation whereas in the dynamic one some changes could be happened. A short description of every chapter is presented below.

Chapter 2 describes problems and decisions to be made in container terminals. Containers are usually handled in two major compartments. These compartments and the equipment involved in

them are described in this chapter. Then, the operations in container terminal are disclosed and the main decisions are classified. The decisions are subdivided into five scheduling decisions; namely (1) allocation of berths to arriving vessels and quay cranes to docked vessels, (2) storage space assignment, (3) rubber tyred gantry crane deployment, (4) scheduling and routing of vehicles and (5) making appointment times to external trucks.

Chapter 3 makes a literature review dealing with research done in container terminals and formulates the decisions (defined in Chapter 2). Our approach is to formulate the decisions as Constraint Satisfaction Optimization Problems (CSOPs). We formulate each of the decisions independently, according to the particular assumptions. After the formulation, the latest researches over some of the major container terminals in the world are summarized. A summary of solutions for the problems are provided at the end of this chapter.

Chapter 4 focuses on one of the most important problems in the ports and then formulates it. One of the equipment in an automated container terminal is Automated Guided Vehicles (AGVs). These robotic vehicles travel along a predefined path inside the terminal and transport containers. This chapter defines a scheduling problem for these kind of vehicles in container terminals. The problem is to deploy several AGVs in a port to carry many containers from the quay-side to yard-side or vice versa. This problem is formulated under the Minimum Cost Flow model, which is a directed graph. There are two aspects for the problem, static and dynamic. In static problems there is no change in the situation whereas in dynamic ones, the problem changes over time.

Chapter 5 applied the standard Network Simplex Algorithm (NSA) to the scheduling problem of Automated Guided Vehicles (defined in Chapter 4) in static aspect. In this aspect the number of jobs, the distance between the source and destination of the jobs, and the number of vehicles don't change. In this chapter, we collected experimental results from the efficient implementation of NSA. The NSA can find the global optimal solution for 3,000 jobs and ten millions arcs in the graph model within two minutes.

Chapter 6 presents a novel version of NSA, which is called Network Simplex plus Algorithm (NSA+). In order to show NSA+ is faster than NSA, several random problems are tackled by the both algorithms and CPU-time required to solve the problems are tested statistically. After that,

NSA+ is applied to solve the dynamic Automated Guided Vehicle scheduling problem and the results of simulation are studied.

In Chapter 7, we extend Network Simplex Algorithm in dynamic aspect. In this aspect, the Dynamic Network Simplex Algorithm (DNSA) and the Dynamic Network Simplex plus Algorithm (DNSA+) are presented. The objectives of Dynamic Network Simplex Algorithm are to solve the new problem faster, to use some parts of the previous solution for the next problem and to respond to changes in the problem. In this chapter, NSA+ and DNSA+ are applied to the dynamic scheduling problem of Automated Guided Vehicles in container terminals and their results are compared.

Chapter 8 presents a greedy algorithm (Greedy Vehicle Search-GVS) to complement the above solutions for the problem defined in Chapter 4. GVS is an incomplete solution for both static and dynamic problems. In Chapters 5-7, the scheduling problem of Automated Guided Vehicles, the problem in Chapter 4, is solved by NSA and its extensions. Although these complete solutions are efficient, they can only work on problems with certain limits in size. When size of the problem goes beyond the limits or the time available to solve the problem is too short, GVS is used. To evaluate the relative strengths and weaknesses of GVS and NSA+, a few comparisons are performed in this chapter.

Chapter 9 makes a summary and conclusions of this research. In this chapter, we provide a comparative summary of the algorithms for the scheduling problem of automated guided vehicles (defined in Chapter 4). Since the container terminals have an important role in globalisation and international trade, several suggestions for further research are provided at the end of this chapter.

Chapter 2: Problem Description and Decisions to be made

This chapter describes problems in the container terminals. Containers are usually handled in two important compartments. We shall first describe what the compartments are, including the equipment involved in them. Then, the operations in container terminal are disclosed. After that, main decisions in the container terminal are defined. These decisions are subdivided into five scheduling problems.

2.1 Compartments

The first compartment is *Yard-Side*, which sometimes is referred to as **Storage Area** or **Stacking Lane** [90]. In any container terminal, storage yard serves as temporary buffers for inbound and outbound containers. Inbound containers are brought in the port by vessels for import into land, whereas outbound containers are brought in by trucks and for loading onto vessels in order to export. A large scale yard may comprise a number of areas called *zones* [107]. In each zone, containers are stacked side by side and on top of one another to form rectangular shape, which is called *block* [107]. A typical yard-side with $3\frac{1}{2}$ blocks at the front row is shown in Figure 2-1.



Figure 2-1: The container storage area in a port [68]

There is expensive equipment in the storage area for container handling, which is referred to as **Rubber Tyred Gantry Cranes (RTGCs)** [107,68, 90]. In Figure 2-2, a RTGC can be seen across the block from the front-left to the front-right, while it is unloading a container from a truck. The efficiency of yard operations often depends on productivity of these *RTGCs* and their deployment. To balance the workload among blocks, RTGCs are sometimes moved between blocks so that they can be fully utilized.



Figure 2-2: An RTGC sits across the width of a block [68]

Figure 2-3 shows a typical set-up of blocks where a *RTGC* can move from one block to the others. For example, a RTGC can move from block B_1 to B_2 along a straight line without any rotation of its wheels because the two blocks are adjacent and align longitudinally.

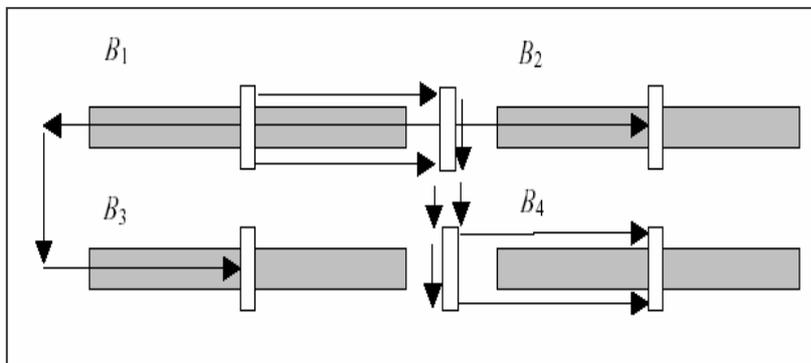


Figure 2-3: Transfer of a RTGC between two blocks [107]

To move between blocks B_1 and B_3 , or between blocks B_1 and B_4 , an *RTGC* has to make a 90-degree rotation (of its wheels) twice to move from one block to another. Since *RTGCs* are big in size and slow in motion, their movements demand a large amount of road space in the terminal for a non-trivial time period. Furthermore, any *RTGC* movement from one block to another takes time, and will result a loss in productivity of the *RTGC*.

The second compartment in the container terminal is *Quay-Side* [108, 68, 90]. Usually, *Quay-Side* consists of a limited number of berths, each of which is equipped by several *Quay Cranes* (*QC*) [108, 68, 90]. The cranes are used to unload containers from vessels of the wharf and load containers to vessels. The cranes are usually flexible to be moved from a berth to another. Figure 2-4 shows a typical *QC*, while it is unloading containers from a vessel to put it down on the truck in order to transport to the storage area.

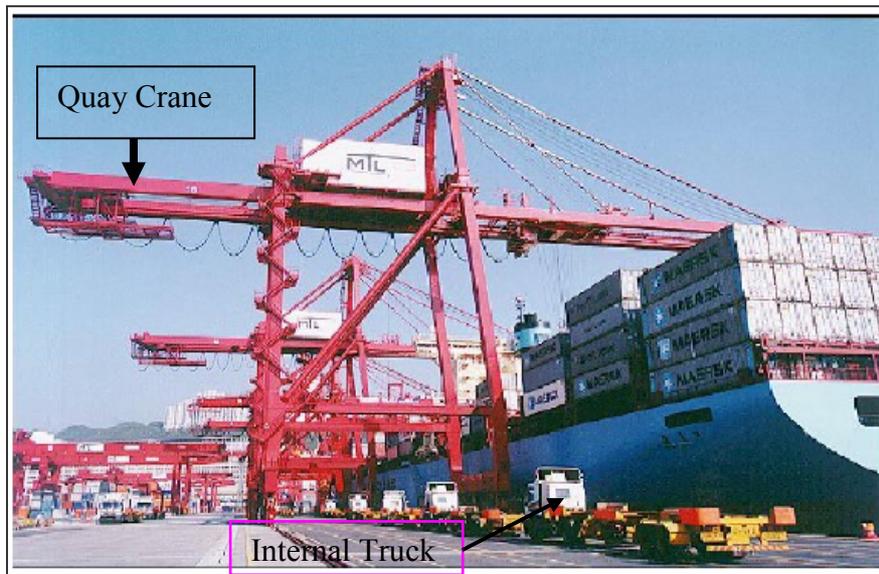


Figure 2-4: A typical quay crane [68]

Berths are essential resources in the container terminal. Therefore, with a high traffic of vessels, it would be ideal to have optimal allocation of berths to vessel to prevent undue delays of vessel in the terminal. At any time, only one ship can be docked at a berth.

2.2 Operations

The main operations in the port start by ship's arrival. After a ship is berthed, it invokes a number of delivery requests for discharging. There are some vehicles in a terminal, which are usually *Automated Guided Vehicles (AGV)* [108, 90] (see Figure 2-5, right), or *Internal Trucks (IT)* [107] (see Figure 2-4, right bottom corner). Idle vehicles are dispatched according to the unloading request list to deliver containers from the berth to designated places in the storage yard. The QCs first unload containers from the containership and put them onto the vehicles. After that the vehicles carry the containers to designated storage area blocks and *RTGCs* unload the containers from the vehicles. Then the containers are put onto the yard stacks. In some terminals there is a number of *Straddle Carrier (SC)* [90] (see Figure 2-5, left), capable of loading, transporting and unloading of containers.



Figure 2-5: A Straddle Carrier (left) and an Automated Guided Vehicle (right) while they are carrying a container.

Straddle Carrier can load/unload and transport containers.

After the unloading phase of the ship, the loading phase will begin. On the land side, *eXternal Truck (XT)* [107] brings in outbound containers before loading process of the relevant vessel, and they pick up inbound containers from the storage area or from the discharged vessel by QCs. The ship issues a number of loading requests. Vehicles are dispatched corresponding to the loading request list to deliver containers to the QCs. The operation is the reverse of the unloading process.

There are two major types of waiting lists in the port. The first one related to vehicles while the second one dedicated to the cranes. A vehicle has to wait if it has arrived at the crane's location

but the crane is busy with other vehicles. A QC has to wait for a vehicle if it is ready to put a container onto a vehicle or to pick up a container from a vehicle but the vehicle has not arrived on the quay-side. Usually the cranes waiting time is more critical than the vehicle waiting time for efficiency of the terminal operations. Any delay in a quay crane operation will cause the same amount of time delay in all subsequent operations assigned to the same quay crane [108]. This delay may even affect the ship's stay at the berth. Usually every ship has a time window and any delay lead to growing costs for the terminal. So one of the most important decisions in this system is allocation of quay cranes so that satisfy ship timing window or minimize waiting times of the ships in the port.

2.3 Decisions to be made

In this section, we classify the important problems to be made in the container terminals. There are many inter-related decisions during the planning period in a port every day or week, for example. Additionally, these scheduling-resource allocation decisions involve time, space and routes in the terminal, which increase the complexity of the system. Henry et al. (2005) are considering the interaction between QCs, AGVs and Automated Yard Cranes (AYCs) in an integrated model [38]. They made a mixed-integer programming model and now are developing a multi-layer genetic algorithm. Obviously, it is not possible to provide answers to all operations in the previous section by solving a single problem within the scope of this thesis. The problem is therefore divided into some sub-problems.

The first classification of problems in the container terminal has been suggested by Iris [45]. She proposed four sub-problems (2005); i.e. arrival of the ship, unloading and loading of the ship, stacking of containers and transportation of containers from ship to stacking area or vice versa. In her classification, each of the decisions can be studied at strategic, tactical and operation levels. At the strategic level plan over future horizons, it is decided which layout, material handling equipment and ways of operations are used. These decisions lead to the definition of set of constraints for both the tactical and operational levels. Another classification and literature review over operations in the container terminal have been provided by Steenken et al. (2004). They divided the decisions into ship planning processes, stowage and stacking logistics, and transportation problems [87]. In their classification, the first one consists of berth allocation and stowage planning and crane splitting whereas the decisions related to yard cranes and storage

area allocation are in the second category. The third category of the decisions refers to transportation problems from the quay side to the storage area or vice versa, the equipment to carry the container from their source to their destination and traffic inside the terminal. Additionally, Murty et al. (2005) classified the daily operations of a container terminal into nine decisions [68]; namely, allocation of berths to arriving vessels, allocation of QCs to docked vessels, appointment times to XTs, routing of trucks, dispatch policy at the terminal gate and the dock, storage space assignment, RTGC deployment, IT allocation to QC and IT hiring plans.

With respect to scheduling view, we classify important problems in the container terminals into five decisions, as shown in Figure 2-6. These decisions are usually executed in different time periods. A short description for each of the decisions is given as follows:

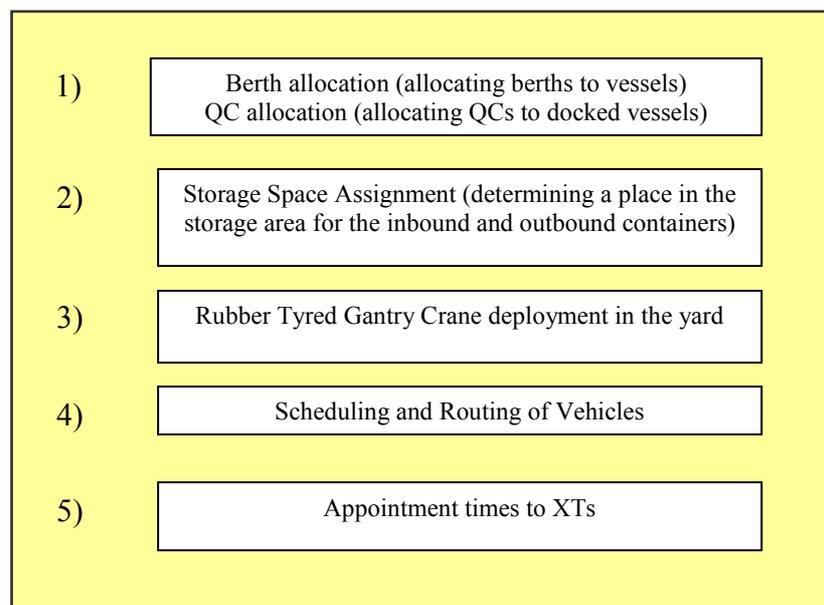


Figure 2-6: Scheduling Decisions in the container terminals.

2.3.1 Allocation of berths to arriving vessels and QCs to docked vessels

The first decision is to maximize utilization of the berths and QCs. Generally, a port has limited number of berths, efficient allocation of berths to arriving vessels and QCs is essential to guarantee ship's timing window, to minimize the ship's waiting time and to maximize port's

turnaround. This decision affects the turnaround time of vessels, and throughput rate of the terminal.

2.3.2 Storage space assignment

Two kinds of storage areas (*Primary* and *Secondary*) are proposed for medium and short-term storage of containers [90]. Assigning these storage spaces to arriving inbound and outbound containers are another scheduling-resource allocation problem. In this decision it is desirable to minimize reshuffling or reorganizing volume and minimize the costs of containers.

2.3.3 Rubber Tyred Gantry Crane (RTGC) deployment

To manoeuvre the containers in the blocks, RTGCs are used (Figure 2-3). One major decision in port automation is to determine how many RTGCs work in each block, and when a RTGC needs to move from one block to another. This decision affects the port time of vessels, the waiting times of QCs and ITs or AGVs.

2.3.4 Scheduling and routing of vehicles

In each port, there are several vehicles to carry containers between the yard-side and quay-side or vice versa. The scheduling and routing these vehicles is another important decision. The objectives of this decision are to minimize transportation costs of the containers and the waiting times of the QCs and RTGCs.

2.3.5 Appointment times to eXternal Trucks (XTs)

The fifth decision in our classification is to make appointment times for the external trucks (XTs). In reality, all consignees book the time to pick up their inbound containers, by calling beforehand and taking appointments. The customers also book a time to bring in their outbound containers. This decision helps to minimize the waiting times of XTs, and congestion in the gate of terminal.

Chapter 3: Literature Review and Formulation of the Decisions

In the previous chapter, we defined five scheduling decisions in container terminals. To recapitulate, these decisions are:

- Allocation of berths to arriving vessels and quay cranes to docked vessels.
- Storage space assignment.
- Rubber Tyred Gantry Crane deployment.
- Scheduling and routing of vehicles.
- Appointment times to external trucks.

The objectives of this chapter are to survey on research done in these decisions and then formulate them as Constraint Satisfaction Optimization Problems (CSOPs). The five decisions are formulated separately so that they can be studied independently. After the formulation, the latest researches over some of the main container terminals in the world are summarized. A summary of solutions for the problems can be found at the end of this chapter.

3.1 Allocation of berths to arriving vessels and quay cranes to docked vessels.

In container terminals, the berth is the most important resource that affects the capacity of the terminal directly. The main reason is that the construction cost of the berths is relatively very high compared with the investment on facilities in the port [73]. Thus, an effective way to increase the capacity of a terminal is to improve the efficiency of its berth.

The problem here is to allocate berths to arriving vessels and to determine which cranes in the berths process the docked vessels. The operator of the terminal usually creates and maintains a berth schedule which shows the berthing position and time of each arriving vessel. For creating the berth schedule, the calling schedule of vessels, favorable berthing location (near primary storage, for example) and the number of available cranes must be considered simultaneously.

The static and dynamic berth allocation problems have been studied by Hansen and Oguz (2003). In the static problem they assumed that ships arrive to the port before the berths become available. In the dynamic problem, there was no constraint on arrival time of the ships. Their integer programming model has been tackled by CPLEX software [33].

The most important objective in berth scheduling is to reduce the amount of time required to unload and load a ship. Thurston and Hu (2002) presented a distributed agent architecture to achieve the objective and increase the container throughput of the port [90]. Under this architecture, an intelligent planning algorithm was continuously optimized by the dynamic and co-operative rescheduling of yard resources such as RTGCs and container vehicles. Another research group, Rebollo et al. (2000) presented a multi-agent system architecture to solve the automatic allocation problem in the container terminals in order to minimize the ship's docking time [84]. Their paper focused on the management of cranes by a 'transtainer agent'. The independence of subsystems obtained for a multi-agent approach was emphasized.

The berth-scheduling and crane-scheduling problems have been considered to be independent of each other. Moon (2001) studied only the first problem by a Mixed Integer Linear Program (MILP) model [65]. In the model each vessel requires a specific amount of the space on the berth during a predetermined length of time for unloading and loading containers. Blażewics et al. (2005) modelled the berth scheduling as a moldable task scheduling problem by considering the relation between the number of quay cranes and the berthing time [7]. Moldable tasks form one type of parallel tasks that can be processed simultaneously on a number of parallel processors for which the processing times are a function of the number of processor assigned. The aim of the model was to minimize the idle time on processors so as to increase the utilization of the berths. On the second problem, the crane scheduling, Böse et al. (2000) focused on maximising the productivity of the cranes and reducing the time in port for the vessels by using evolutionary algorithm [8].

However, the duration of berthing of each vessel depends on the number of cranes assigned to the corresponding vessel. When the number of cranes assigned to a vessel increases, berthing duration of the vessel can be reduced. Because of this important reason, the berth-scheduling and crane-scheduling problems should be considered simultaneously in the port. Park and Kim (2003) made a MILP model to consider the both problems [73]. They suggested two phases for

solving the mathematical model, “berth scheduling phase” and “crane assignment phase”. These two phases are summarised by Figure 3-1. The first phase determined the berthing position and time of each vessel as well as the number of cranes assigned to every vessel at each time period. The sub-gradient optimization technique was applied to obtain a near-optimal solution for the first phase. In the second phase, a detailed schedule for each quay crane was constructed based on the solution found from the first phase. In the second phase, dynamic programming technique was applied to solve the problem.

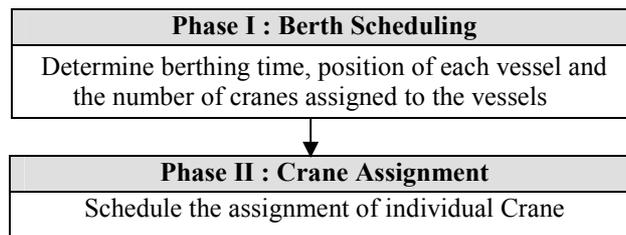


Figure 3-1: Park and Kim’s two phases scheduling of berths and cranes [73]

3.1.1 Assumptions

Here, we combine the two phases of Park and Kim’s model and convert it to CSOPs. The followings assumptions are considered in formulating this decision:

Assumption 3-1-1: A fixed time-window is considered for the quay cranes to discharging/loading a container. With this assumption, the duration of berthing of or processing a vessel is inversely proportional to the number of cranes assigned to.

Assumption 3-1-2: Each vessel determines the maximum and minimum number of cranes that can be assigned to it [73]. The number of cranes can change from a period to period.

Assumption 3-1-3: Each vessel has a pre-determined berthing time period. A cost penalty applies if the vessel berths early or departs late.

Assumption 3-1-4: Each vessel has a preferred location of berthing [73]. This preferred location can be the location nearest to the storage area where inbound/outbound containers for the corresponding vessel are stacked. Another preference of a berthing location may also come from the depth of water or the strength and direction of currents.

The output of solution methods for this decision is illustrated in Figure 3-2. In the figure there are five vessels and each rectangle represents the berthing schedule of a vessel. The berthing locations are shown on the horizontal sides and the positions of the vertical sides correspond to operation times of vessels. The number on the left side of every ship shows how many cranes process the vessel at a specific time, while the crane number has been shown in the middle of grid.

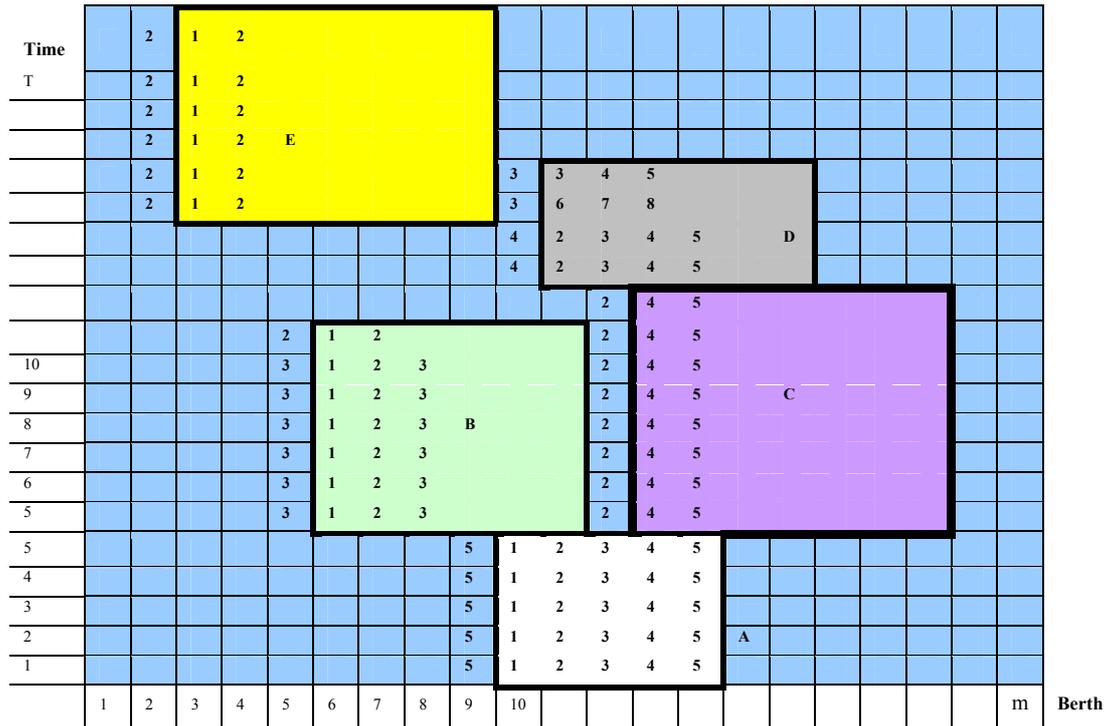


Figure 3-2: An output of the berth and crane scheduling problem

The following variables are given at the beginning of the planning horizon:

T: The total number of time periods in the planning horizon. The time period is equal to the time window of cranes (see Assumption 3-3-1).

ETA_k: The expected time of arrival of vessel *k*.

a_k: The processing time of vessel *k* (if only one crane is assigned to vessel *k*).

b_k: The length of vessel *k*.

d_k: The due time for the departure of vessel *k*.

s_k: The least-cost berthing location of the reference point of vessel *k*.

c_{1k}: The penalty cost of vessel *k* if the vessel could not dock at its preferred berth.

c_{2k}: The penalty cost of vessel *k* per unit time of earlier arrival before *ETA_k*.

c_{3k}: The penalty cost of vessel *k* per unit time of late arrival after *ETA_k*.

c_{4k} : The penalty cost of vessel k per unit time delay behind the due time.

L_k : The minimum number of cranes that can be assigned to vessel k .

U_k : The maximum number of cranes that can be assigned to vessel k .

l : The number of vessels in the planning horizon.

C : The total number of cranes in the terminal ($C > \text{Max}(U_k), k=1,2,\dots,l$).

m : The number of berths in the port.

3.1.2 Decision variables and domains

At_k : The arrival time of vessel k to the berth.

$$\text{Domain}(At_k) = \{1, 2, 3, 4, \dots, T\}$$

Dt_k : The departing time of vessel k .

$$\text{Domain}(Dt_k) = \{1, 2, 3, 4, \dots, T\}$$

X_{itk} : 1 if the berth i at time t is allocated to vessel k , otherwise 0.

$$\text{Domain}(X_{itk}) = \{0, 1\}$$

Q_{itkc} : Status of crane c ; it is 1 if the crane c in the i -th berth is processing vessel k at time t , otherwise 0. $\text{Domain}(Q_{itkc}) = \{0, 1\}$

3.1.3 Constraints

Constraint 3-1-1: The grid squares are covered by only one vessel. In fact, each berth at time t can be assigned to only one vessel.

$$\sum_{k=1}^l X_{itk} \leq 1 \text{ for } i = 1, 2, 3, \dots, m; t = 1, 2, 3, \dots, T$$

Constraint 3-1-2: Each berth is allocated for the vessel only between its arrival and departure.

$$\begin{aligned} At_k \leq t \leq Dt_k &\Rightarrow X_{itk} = 1 \\ (At_k > t) \text{ OR } (t > Dt_k) &\Rightarrow X_{itk} = 0 \\ \text{for } t = 1, 2, \dots, T; \text{ for } i = 1, 2, \dots, m; \text{ for } k = 1, 2, \dots, l \end{aligned}$$

Constraint 3-1-3: Only one crane operates on the vessel in a certain time and berth.

$$\sum_{k=1}^l \sum_{c=1}^C Q_{itkc} = 1 \text{ for } i = 1, 2, 3, \dots, m; t = 1, 2, 3, \dots, T$$

Constraint 3-1-4: The number of quay cranes assigned to each vessel is limited and the vessels have to be fully processed by the QCs.

$$L_k \leq \sum_{i=1}^m \sum_{t=At_k}^{Dt_k} \sum_{c=1}^C Q_{itkc} \leq U_k, \sum_{i=1}^m \sum_{t=At_k}^{Dt_k} \sum_{c=1}^C Q_{itkc} = a_k, \text{ for } k = 1, 2, \dots, l$$

Constraint 3-1-5: The crane c processes vessel k at berth i in time t , if the berth and crane are allocated to the vessel.

$$(Q_{itkc} \text{ AND } X_{itk}) = 1, \text{ for } c = 1, 2, \dots, C; \text{ for } i = 1, 2, \dots, m; \text{ for } k = 1, 2, \dots, l; \text{ for } t = 1, 2, \dots, T$$

Constraint 3-1-6: Two time periods are required to set-up any crane from one berth to another.

$$(Q_{itkc} \text{ AND } Q_{it'kc}) = 0, \text{ for } c = 1, 2, \dots, C; \text{ for } k = 1, 2, \dots, l \\ \text{for } t, t' = 1, 2, \dots, T, (|t - t'| = 1, t \neq t'); \text{ for } i = 1, 2, \dots, m$$

Constraint 3-1-7: If the length of a vessel is greater than the distance between two berths, other vessels are not allowed to dock at the adjacent berth.

$$(|i - i'| < b_k \text{ OR } |i - i'| < b_{k'}) = \text{True} \Rightarrow (X_{itk} \text{ AND } X_{it'k'}) = 0, \\ \text{for } k, k' = 1, 2, \dots, l; k \neq k'; \text{ for } i, i' = 1, 2, \dots, m; i \neq i'; \text{ for } t = 1, 2, \dots, T$$

In the constraint, $|i - i'|$ denotes the distance between berths i and i' .

3.1.4 Objective function

The objective function of this decision is to minimize the total penalty cost. In order to present the objective function, we introduce the following auxiliary variable:

Z_k : The sum of the absolute distance between the preferred location of vessel k and the berths allocated to the vessel. This variable is determined by the following function:

$$Z_k = f(X_{itk}, s_k) = \sum_{t=1}^T \sum_{i=1}^m \{|i - s_k| : X_{itk} = 1\}$$

Now the objective function is written as follows:

$$\text{MinCostVessels} = \sum_{k=1}^l \{c_{1k} \cdot Z_k + c_{2k} (ETA_k - At_k)^+ + c_{3k} (At_k - ETA_k)^+ + c_{4k} (Dt_k - d_k)^+\}$$

The first factor is the penalty cost incurred by the distance between the berthing locations of a vessel and the preferred location. The second and third factors are the penalty costs by the actual berthing earlier or later than the expected time of arrival. The last factor is the penalty cost caused by the delay of the departure after the promised due time. The three last terms have impacts on the objective function provided that they are only positive.

3.2 Storage space assignment

There is evidence that the yard plays an important role in global productivity of the terminal [55]. In fact, the efficiency and quality of management in the container yard operations affect all terminal decisions, related to the allocation of available handling equipment and the scheduling of all activities. The problem here is to determine a place in the storage area for the inbound and outbound containers.

Ambrosino et al. (2002) studied the impact of yard organization on the stowage of containers in terms of unproductive export containers movement in the port [5]. They tackled the problem using a heuristic approach based on a 0-1 linear programming model. Another research group, Murty et al. (2005), studied storage space assignment and vehicle routing problem, together in the same problem [68]. For the former problem, they suggested two steps, block assignment and storage position assignment. In the first step, they determined how many containers, inbound or outbound containers, are stored in every block at each time period. In the second step, the optimal available position in the block was determined for storing the containers. While the reshuffling of containers that may arise was minimized [68], the containers flow and scheduling problem have not been considered in that paper. In the same way, Steenken et al. (2001) combined container stowage and transport planning problem [88]. Then a mixed integer model was presented for just-in-time container scheduling with one quay crane. An exact and heuristic methods to solve the model, has been presented in the paper. Moreover, the storage space allocation in container terminals has been studied by Zhang et al. (2001). They considered the problem in a rolling horizon approach [106]. For each planning horizon, the problem was decomposed into two levels. At first level, the total number of inbound and outbound containers to be placed in every part of the storage was determined. The second level determined the number of containers in each block of the yard by solving a transportation problem. The objective of the problem was to minimize travelling times of the vehicles in the port. Gambardella et al. (1998) presented a decision support system for the management of an inter-modal container terminal [27]. In their model, there were the spatial allocations of containers in the terminal yard. They described some modules for the optimisation of the allocation process and for the simulation of the terminal. The former was based on integer linear programming; the latter was a discrete event simulation tool.

Frankel (1987) suggested three main types of storage systems: short term, long term, and specialized. Henesey et al. (2003) described these kinds of storages [37]. The short-term storage system is for containers that may be transhipped onto another containership. Long-term storage is for containers awaiting customs release or inspection. Specialized storage is reserved for the refrigerated (they need to be supplied with electricity) and hazardous materials. Holguin and Jara took into account the intrinsic and logistic values of containers and divided them into different priority classes. For each class the optimal amount of space and price were determined under welfare and profit maximizing rules (which has been surveyed in [18]).

3.2.1 Assumptions

We assume that the storage area is divided into the short-term and medium-term storages. These two storages are usually referred to as the primary and secondary [90]. Figure 3-3 shows a layout of the port with these storages. The purposes of the primary storage are to store transit containers [37] (from one ship to another), to minimize waiting times of QCs and ships [90], and to be used in emergency situations such as deadlock of the vehicles. The secondary storage is where the inbound containers are picked up by their consignees and the outbound ones are brought in by customers. The QCs and RTGCs handle containers in the primary and secondary area, respectively. The size of the secondary storage is usually greater than the primary.

Our approach is to consider the interaction of containers between the primary and secondary storages. Based on the layout of storages, containers are classified into the six following types according to their status at different stages:

- (a) Primary Storage Containers to Secondary Storage (PSCSS): Containers in the primary storage waiting to be moved to the secondary storage.
- (b) Secondary Storage Containers to Primary Storage (SSCPS): Containers in the secondary storage waiting to be moved to the primary storage.
- (c) Secondary Storage Container Pickup (SSCPI): Inbound containers in the secondary storage waiting for pickup by consignees.
- (d) Secondary Storage Container Grounding (SSCGD): Outbound containers before being allocated to the secondary storage.

- (e) Primary Storage Container Pickup (PSCPI): Outbound containers in the primary storage waiting to be loaded on the arriving vessels.
- (f) Primary Storage Container Discharging (PSCDS): Inbound containers, being discharged from the arriving vessels and to be allocated to the primary storage.

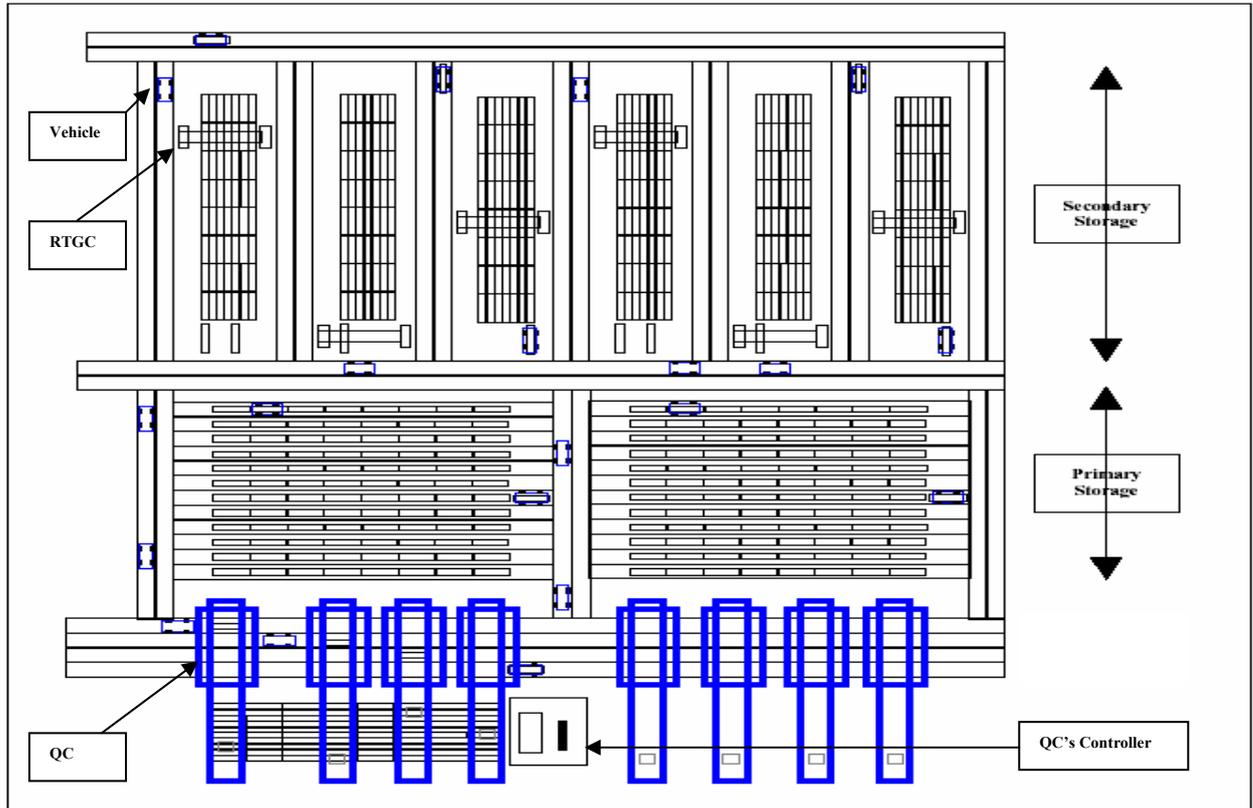


Figure 3-3: Port's layout with the primary and secondary storages [90]

The following assumptions are considered to formulate this decision:

Assumption 3-2-1: As stated in Chapter 2, the storage areas are divided into different blocks. In this decision, it is necessary to determine which blocks and how many spaces in them to be allocated to the six types of containers.

Assumption 3-2-2: Several QCs might be busy with other operations. So we assume that there is a tight constraint on the minimum and maximum of QCs in the primary storage during each time period.

Assumption 3-2-3: Our objectives are to balance the workload of RTGCs in the secondary storage [106] and to minimize the handling costs of containers in those two kinds of storages.

Assumption 3-2-4: The maximum dwell times of the inbound and outbound containers approximately equal the maximal free storage period, which is beyond the planning horizon [106]. There are containers with unknown removal times at the planning period or containers with known departure times beyond the planning horizon. Their associated workload does not occur in the planning horizon and consequently such containers cannot be considered in this storage allocation model. Instead, these containers are distributed to blocks in proportion to their available storage capacities at the beginning of the planning horizon so as to balance the block densities in the secondary storage.

Assumption 3-2-5: Within each block, the exact location of a container can be assigned to shorten the handling time by minimizing reshuffling [106]. This decision about storage location is a problem at a lower level, and is not considered in this formulation.

Assumption 3-2-6: The secondary storage is where the customers bring in their outbound containers and the consignees pick up their inbound containers. The outbound containers then transported to the primary storage. Also it is assumed the inbound containers are first stored in the primary storage and then transported to the secondary storage. We assume that the primary and secondary storages have enough space to store all the containers over the planning horizon.

In order to make the model, the following parameters are known at the beginning of a planning horizon:

TP_{ij} : The travelling time between block i of the primary storage to block j of the secondary.

TS_{ij} : The travelling time between block i of the secondary storage to block j of the primary.

T : The total number of time periods in the planning horizon. The time period has to be greater than the maximum travelling time between the primary storage and the secondary storage or vice versa.

B : The total number of blocks in the secondary storage.

C_i : The storage capacity of block i of the secondary storage.

P : The total number of blocks in the primary storage.

F_i : The capacity of block i of the primary storage.

- H_{i0}** : The initial inventory of block i of the primary storage, i.e., the number of containers in primary storage at the beginning of the planning horizon.
- S_{i0}** : The initial inventory of block i in the secondary storage, i.e., the number of containers in block i at the beginning of the planning horizon.
- PE_{0it}** : The expected number of initial SSCPI containers stored in block i of the secondary storage to be picked up during period t .
- LO_{it}** : The expected number of initial PSCPI containers stored in block i of the primary storage to be moved to the arriving vessels during period t .
- GE_{tk}** : The expected total number of SSCPS containers that to be allocated in the secondary storage during period t and to be moved to primary storage in period $t + k$.
- DE_{tk}** : The expected total number of PSCSS containers, allocated in the primary storage during period t , and to be picked up from the secondary storage in period $t + k$.
- G_t** : The expected total number of SSCGD containers that arrive at the terminal during period t and to be stored in the secondary storage.
- D_t** : The expected total number of PSCDS containers that arrive to the terminal during period t by vessels and to be stored in the primary storage.
- α_t** : The expected number of SSCGD containers storing in secondary storage during period t , and to be moved to the primary storage in periods beyond the current planning horizon.
- β_t** : The expected number of PSCDS containers arriving at the terminal during period t , and to be moved to the secondary storage, with an unknown pickup time or pickup time beyond the planning horizon.
- Q_t, R_t** : The maximum and minimum number of available QCs, respectively, to handle PSCSS, SSCPS, PSCPI and PSCDS containers in the primary storage during period t .

3.2.2 Decision variables and domains

The following decision variables are defined:

X_{ijt} : The total number of PSCSS containers in block i of the primary storage to be moved to block j in the secondary storage during time period t .

$$\text{Domain } X_{ijt} = \{0, 1, 2, \dots, \text{Max}(F_i, C_j) | i=1, 2, \dots, P, j=1, 2, \dots, B\}$$

Y_{ijt} : The total number of SSCPS containers in block i of the secondary storage to be moved to block j in the primary storage during time period t .

$$\text{Domain } Y_{ijt} = \{0, 1, 2, \dots, \text{Max}(C_i, F_j) | i=1, 2, \dots, B, j=1, 2, \dots, P\}$$

GS_{it}: The total number of SSCGD containers that arrive at the terminal during period t and to be stored in block i of the secondary storage.

$$\text{Domain GS}_{it} = \{0, 1, 2, \dots, \text{Max}(C_i) | i=1, 2, \dots, B\}$$

DP_{it}: The total number of PSCDS containers that arrive to the terminal during period t by vessels and to be stored in block i of the primary storage.

$$\text{Domain DP}_{it} = \{0, 1, 2, \dots, \text{Max}(F_i) | i=1, 2, \dots, P\}$$

3.2.3 Constraints

In order to present the constraints of this decision, we introduce the following auxiliary variables:

PE_{it}: The total number of SSCPI containers stored in block i of the secondary storage, that is picked up by consignees during period t. This variable is determined by the following expression:

$$PE_{it} = \sum_{t'=1}^{t-1} \sum_{j=1}^P X_{jit'} + PE0_{it}, \text{ for } i = 1, 2, \dots, B; \text{ for } t = 1, 2, \dots, T$$

L_{it}: The total number of PSCPI containers stored in block i of the primary storage that to be moved to the arriving vessels during period t. This variable is determined by the following expression:

$$L_{it} = \sum_{t'=1}^{t-1} \sum_{j=1}^B Y_{jit'} + L0_{it}, \text{ for } i = 1, 2, \dots, P; \text{ for } t = 1, 2, \dots, T$$

H_{it}: The inventory of block i of the primary storage at the beginning of period t. This variable is determined by the following expression:

$$H_{it} = H_{i(t-1)} + DP_{it} + \sum_{j=1}^B Y_{jit} - \sum_{j=1}^B X_{jit} - L_{it}, \text{ for } t = 1, 2, \dots, T; \text{ for } i = 1, 2, \dots, P$$

The expression represents updating of inventory in the primary storage from a period to the next period. The first term is the initial inventory of block i. The second term is the number of PSCDS containers, being allocated in block i. The third and fourth terms state the inventory of block i is increased and decreased by the number of SSCPS and PSCSS containers, respectively. The last term is the number of PSSPI containers, being moved from block i to the arriving vessels.

S_{it}: The inventory of block i of the secondary storage at the beginning of period t. This variable is determined by the following expression:

$$S_{it} = S_{i(t-1)} + GS_{it} - \sum_{j=1}^P Y_{jit} + \sum_{j=1}^P X_{jit} - PE_{it}, \text{ for } t = 1, 2, \dots, T; \text{ for } i = 1, 2, \dots, B$$

The expression represents updating of inventory in the secondary storage from a period to the next period. The first term is the initial inventory of block i. The second term is the number of SSCGD containers, being allocated in block i. The third and fourth terms state the inventory of block i is decreased and increased by the number of SSCPS and PSCSS containers, respectively. The last term is the number of SSCPI containers, being picked-up from block i.

QC_t: The number of QCs required to handle the four different type of containers (PSCPI, SSCPI, PSCSS and PSCDS) in the primary storage during period t. This variable is determined by the following expression:

$$QC_t = \sum_{i=1}^P L_{it} + \sum_{i=1}^B \sum_{j=1}^P Y_{ijt} + \sum_{i=1}^P \sum_{j=1}^B X_{ijt} + \sum_{i=1}^P DP_{it}$$

Now we present the constraints for this decision:

Constraint 3-2-1: Constraints on inventory of each block in the primary and secondary storage and their densities.

$$S_{it} \leq \lambda C_i, \text{ for } t = 1, 2, \dots, T; \text{ for } i = 1, 2, \dots, B$$

$$H_{it} \leq \gamma F_i, \text{ for } t = 1, 2, \dots, T; \text{ for } i = 1, 2, \dots, P$$

The first constraint ensures that the inventory in each block of the secondary storage in each time period will not exceed the threshold level (which is being controlled by λ ; $\lambda < 1$). The later ensures that the inventory of each block of the primary storage in each planning period will not exceed the allowable block density (which is being controlled by γ ; $\gamma < 1$).

Constraint 3-2-2: Constraints on flow of the containers.

$$\sum_{i=1}^P \sum_{j=1}^B X_{ijt} = \sum_{k=t+1}^T DE_{ik} + \beta_t; \text{ for } t = 1, 2, \dots, T$$

$$\sum_{i=1}^B \sum_{j=1}^P Y_{ijt} = \sum_{k=t+1}^T GE_{ik} + \alpha_t; \text{ for } t = 1, 2, \dots, T$$

$$D_t = \sum_{i=1}^P DP_{it}; \text{ for } t = 1, 2, \dots, T$$

$$G_t = \sum_{i=1}^B GS_{it}; \text{ for } t = 1, 2, \dots, T$$

The first constraint ensures that the expected total number of PSCSS containers to be moved to the secondary storage, DE_{tk} , and the number of containers with known departure, β_t , is the sum of PSCSS containers moved from each block of the primary storage to all blocks in the secondary storage during period t . The second constraint has a similar meaning but for the SSCPS containers. The third constraint ensures that the expected total number of PSCDS containers allocated to all blocks in the primary storage is the sum of total number of containers arriving to the terminal by the vessels during period t . The fourth constraint has a similar meaning but for the SSCGD containers.

Constraint 3-2-3: Constraints on the number of available QCs in the primary storage.

$$R_t \leq QC_t \leq Q_t, \text{ for } t = 1, 2, \dots, T$$

3.2.4 Objective function

The objective function is to minimize distribution of the total number of containers among blocks in the secondary storage and sum of the transportation costs between the both storages. In order to present the objective function in the simpler form, we define the following auxiliary variables:

RTGC_{it}: The number of RTGCs required to handle the four different types of containers (SSCGD, SSCPS, PSCSS and SSCPI) in block i of the secondary storage during period t . This variable is determined by the following expression:

$$RTGC_{it} = GS_{it} + \sum_{j=1}^P Y_{ijt} + \sum_{j=1}^P X_{jit} + PE_{it}$$

M_t , N_t : The maximum and minimum number of RTGC_{it} during period t , respectively. These variables are determined by the following constraints:

$$M_t = \text{Max}_{i=1,2,\dots,B} (RTGC_{it}), \text{ for } t = 1, 2, \dots, T$$

$$N_t = \text{Min}_{i=1,2,\dots,B} (RTGC_{it}), \text{ for } t = 1, 2, \dots, T$$

Now the objective function is written as follows:

$$\text{MinCostStorages} = \sum_{t=1}^T \left\{ W_1 (M_t - N_t) + W_2 \left(\sum_{i=1}^P \sum_{j=1}^B X_{ijt} \cdot TP_{ij} + \sum_{i=1}^B \sum_{j=1}^P Y_{ijt} \cdot TS_{ij} \right) \right\}$$

Note that W_1 is the weight of distribution of containers among blocks in the secondary storage and W_2 is the weight of transportation cost inside the terminal.

3.3 Rubber Tyred Gantry Crane (RTGC) deployment

The RTGC is a critical resource, whose performance in the storage yard affects the waiting times of XTs, ITs or AGVs, and QCs [68]. The waiting time of vessels is also indirectly effected by the productivity of RTGCs. As the workload in the different storage blocks changes over time, deployment of RTGCs among storage blocks in order to provide more RTGCs to blocks with heavier workloads is an extremely important problem in the terminal. The problem here is to determine how many RTGCs work in each block, and when a RTGC needs to be moved from one block to another.

Lim et al. (2002) studied a set of spatial constraints in crane scheduling problem [55]. The most interesting one was the non-crossing constraint, i.e. the crane arms could not be crossed over each other simultaneously. It was a structural constraint on cranes and crane tracks. The problem was modelled as bipartite graph matching. Then, the model was tackled by squeaky wheel optimization with local search technique. Murty et al. (2005) studied this decision with some restriction assumptions [68]. They made an integer programming model by defining a sink block in where the expected workload exceeds the capacity of its RTGCs. Their model was tackled by Vogel solution. Also dynamic RTGC deployment in container storage yard was studied by Zhang et al. (2002). They minimized the total delayed workload in the yard by a mixed integer programming model and tackled it through Lagrangean relaxation [107]. Moreover, Lin (2000) studied the movement problem of yard cranes in the container terminal so as to minimize workloads at the end of each time period. He made a MILP model, which was tackled by Lagrangian decomposition [56].

3.3.1 Assumptions

Here, we present a combination of the assumptions in Zhang's model [107] and Lim's formulation [55] for this decision. These assumptions are:

Assumption 3-3-1: The capacity of RTGCs are measured in time-unit (minutes, for example) [107]. Similarly, the workload of each block is converted to time-unit. It is also assumed the nominal numbers of container moves are given in each time period. These containers are handled by the RTGCs in the yard. Since containers are stacked on each other and may be stored in a predefined pattern, each nominal container retrieval or storage may take more than one real RTGC move. So the total number of container moves is converted into the workload-times by multiplying the average number of real moves per nominal move with the average time needed per move.

Assumption 3-3-2: Because of the limitation of blocks size and the potential danger of RTGCs collision, there is a limited number of RTGCs in each block at any time. There are situations where up to two RTGCs can be worked in each block [107]. But we do not allow more than K RTGCs to be moved from one block to another in a time period.

Assumption 3-3-3: Every RTGC movement starts and finishes within the same time period [107]. This assumption entails that the time period has to be greater than the maximum travelling times of RTGCs among blocks.

Assumption 3-3-4: It is assumed that unfinished work in a block at the end of a time period will be carried over to the next period [107]. As a result, the workload of a block in a time period is the sum of the workload in the current period and the workload carried over from the previous time period. The workload carried over from the previous period will be finished during the early part in the current period.

Assumption 3-3-5: The maximum and minimum available numbers of RTGCs or Yard Cranes in the yard, respectively, are known and fixed during each time period.

Assumption 3-3-6: The RTGCs can not cross over each other in the same period [55]. Figure 3-4 shows a part of the storage yard. Moving an RTGC from block 1 to block 4 and another one from block 3 to block 2 at the same period produces a dangerous situation in the yard.

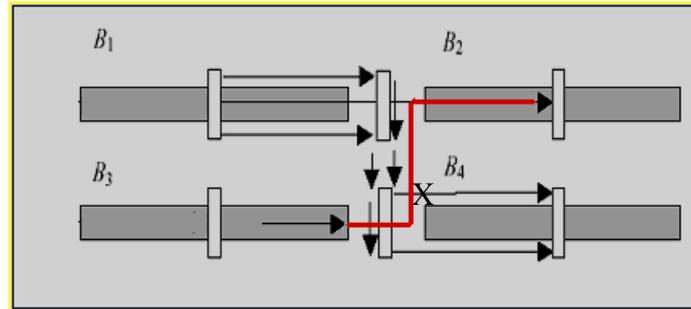


Figure 3-4: Cross over problem for two RTGCs in the storage area

The following parameters are known at the beginning of the planning horizon:

TT_{ij}: The travelling time of a RTGC from block *i* to block *j*.

T: The total number of time periods in the planning horizon. The time period has to be greater than the maximum travelling time of RTGCs between the blocks.

X_{ii0}: The numbers of RTGCs in block *i* at the beginning of the planning horizon.

C: The capacity of a RTGC within a time period.

K: The total permitted number of RTGCs in each block.

N: The total number of blocks in the yard.

B_{it}: The workloads of block *i* within time period *t*. Average time to handle a container are used to determine the workload of each block in time-unit.

M_t, N_t: The maximum and minimum available number of RTGCs or Yard Cranes in the yard, respectively, during period *t*.

L_{ij, kl}: 1 if the movement of RTGC from block *i* to block *j* and from block *k* to block *l* creates cross over problem. Otherwise it is zero. These parameters are determined according to the layout of the storage area.

W_{i0}: The workload of block *i* at the beginning of the planning horizon.

3.3.2 Decision variables and domains

The decision variables are defined as follows:

X_{ijt}: The number of RTGCs moving from block *i* to block *j* during time period *t*.

Domain (X_{ijt}) = $\{0, 1, 2, \dots, K\}$, for $i, j=1, 2, \dots, N$; $t=1, 2, 3, \dots, T$, $i \neq j$

Note that when $i = j$, X_{ijt} indicates the RTGCs stay in the same block during period *t*.

Z_{ijt} : The workload fulfilled in block i by RTGCs that move from block i to block j during time period t . Domain (Z_{ijt}) = $\{0,1,2,\dots,B_{it}\}$

3.3.3 Constraints

Constraint 3-3-1: Maintaining the RTGC flow or movement conservation in each block when RTGCs are deployed from one period to the next period [107].

$$\sum_{j=1}^N X_{ijt} = \sum_{j=1}^N X_{ji(t-1)}, \text{ for } i = 1, 2, \dots, N; t = 1, 2, \dots, T.$$

Constraint 3-3-2: Only K RTGCs can serve a block in a time period.

$$\sum_{j=1}^N X_{ijt} + \sum_{j=1, j \neq i}^N X_{jit} \leq K, \text{ for } i = 1, 2, \dots, N; t = 1, 2, \dots, T.$$

Constraint 3-3-3: The total maximum and minimum available numbers of RTGCs or Yard Cranes in the yard are limited.

$$N_t \leq \sum_{i=1}^N \sum_{j=1}^N X_{ijt} \leq M_t; \text{ for } t = 1, 2, \dots, T$$

Constraint 3-3-4: Two RTGCs can not cross over each other in the same time period.

$$L_{ij,kl} = 1 \Rightarrow [(X_{ijt} > 0) \text{ XOR } (X_{klt} > 0) \text{ XOR } (X_{ijt} + X_{klt} = 0)] \\ \text{for } i, j, k, l = 1, 2, 3, \dots, N; i \neq j, k \neq l, k \neq i, l \neq j, \text{ for } t = 1, 2, \dots, T$$

3.3.4 Objective function

The objective function of this decision is to minimize the remaining workload at each block [107] and travelling time of the RTGCs among blocks during the planning horizon. In order to formulate the objective function, we introduce the following auxiliary variables:

Y_{ijt} : The workload fulfilled in block j by the RTGCs that move from block i to block j during time period t . This variable is determined by the following expression:

$$Y_{ijt} = (C - TT_{ij})X_{ijt} - Z_{ijt}, \text{ for } i, j = 1, 2, \dots, N; t = 1, 2, 3, \dots, T$$

The first term represent the total net capacity of RTGCs that move from block i to block j (a part of their capacities is missed due to the travelling time from block i to block j). The second term is the workload fulfilled in block i by the RTGC.

W_{it} : The workload left in block i at the end of time period t . This variable is determined by the following expression:

$$W_{it} = W_{i(t-1)} + B_{it} - \sum_{j=1}^N Z_{ijt} - \sum_{j=1}^N Y_{jit} \quad \text{for } i = 1, 2, \dots, N; t = 1, 2, 3, \dots, T$$

The first term is the workload in block i from the previous period. The second term is the workload of block i within time period t . The third term states the workload fulfilled in block i by RTGCs that move from this block to others. The last term represents the workload fulfilled in block i by RTGCs that move from the other blocks to this block.

Now the objective function is written as follows:

$$\text{MinCostRTGCs} = \left[w_1 \left(\sum_{i=1}^N \sum_{t=1}^T W_{it} \right) + w_2 \sum_{t=1}^T \sum_{i=1}^N \sum_{j=1}^N TT_{ij} \cdot X_{ijt} \right]$$

The first term is the sum of workload left in all blocks and the second term is travelling times of RTGCs between the blocks. Note that w_1 and w_2 are the weights of those two terms in the objective function.

3.4 Scheduling and routing of vehicles

The Vehicle Routing Problem (VRP) is a well known integer programming problem which falls into the category of NP Hard problems, meaning that the computational effort required solving this problem increase exponentially with the problem size. The VRP is being studied in a broad class of routing problems [99]. Some of these variants are Capacitated Vehicle Routing Problem (CVRP), Vehicle Routing Problem with Time Windows (VRPTW), Capacitated Vehicle Routing Problem with Time Windows (CVRPTW), Multiple Depot Vehicle Routing Problem (MDVRP), Periodic Vehicle Routing Problem (PVRP), Split Delivery Vehicle Routing Problem (SDVRP), Stochastic Vehicle Routing Problem (SVRP), Vehicle Routing Problem with Backhauls (VRPB), Vehicle Routing Problem with Satellite Facilities (VRPSF), Time Dependent Vehicle Routing Problem (TDVRP).

In each port, there are several vehicles to carry containers in the port. The scheduling and routing these vehicles is an extremely important decision. In this section, we review the latest research around dispatching and scheduling of AGVs. After that, scheduling and routing problem of vehicles in the container terminal is formulated as a VRPTW.

During the recent years, several researches have been devoted on dispatching of vehicles in the port [8, 108, 14, 103, 31]. Zhang et al. (2002) made two integer programming model for dispatching vehicles in a container terminal [108]. Two heuristic algorithms have been constructed based on the models and Lagrangian relaxation has provided a better solution for the second model. They applied the models to a real size virtual terminal. Grunow et al. (2004) studied dispatching multi-load AGVs in highly automated seaport container terminals [31]. They made a Mixed Integer Linear Program (MILP) model and presented some priority rules to handle container jobs in the container terminals. Then, the performance of the priority rule based approach and the MILP model have been analysed for different scenarios with respect to total lateness of the AGVs. The main focus of their numerical investigation was on evaluating the priority rule based approach for single and dual-load vehicles as well as comparing its performance against the MILP modelling approach. Additionally, dispatching automated guided vehicles in a container terminal has been studied by Cheng et al. (2003). They presented a network flow formulation to minimize the waiting time the AGVs at the berth side [14]. Böse et al. (2000) focused on the process of container transport by gantry cranes and straddle carriers between the container vessel and the container yard [8]. Their primary objective was the reduction of the time in port for the vessels by maximizing the productivity of the gantry cranes. They tackled the problem using evolutionary algorithm. Wook and Hwan (2000) applied two different dispatching strategies for AGVs in container terminals [103], “dedicated dispatching” and “pooled dispatching”. In the dedicated dispatching, every AGV is assigned to a single QC. In pooled dispatching, an AGV performs delivery tasks for more than one QC. Their primary goal of dispatching AGVs was to complete all the loading and discharging operations as early as possible and their secondary goal was to minimize the total travel distance of AGVs. Their integer programming models were tackled by LINDO software.

Qiu and Hsu (2000 and 2001) addressed scheduling and routing problems for AGVs. They developed conflict-free routing algorithms for two different path topologies and two scheduling strategies. The methods were applied together in a case study [76, 77, 78]. Another

phenomenon in the container terminal for AGVs, while they are moving inside the port and carrying the jobs, is deadlock. This aspect has been studied by Moorthy et al. (2003). They proposed an algorithm for cyclic deadlock prediction and avoidance for zone-controlled AGV system [66]. The algorithm is based on wait and proceeds strategy.

3.4.1 Assumptions

It is assumed that there are several vehicles in the port, which can transport the inbound and outbound containers from a pickup location to a delivery location, inside the terminal. The inbound containers in the berth are transported to the storage area, whereas the outbound containers in the storage area are moved to the berth. The following assumptions and notations are used to formulate this decision:

Assumption 3-4-1: The problem is to serve a number of transportation requests. Each request involves moving a number of container jobs. A directed graph or network is considered for this transportation system. Given n request in the problem, let node i and node $n+i$ represent the pickup and delivery location of the i^{th} job, respectively. In this network, different nodes obviously may represent the same physical location in the yard or berth. By adding node 0 and node $2n+1$, as the depot, to the network, it has the node set $N=\{0,1,2,\dots,n,n+1,n+2,\dots,2n, 2n+1\}$. The pick up and delivery points are respectively included into two sets $P^+=\{1,2,\dots,n\}$ and $P^-=\{n+1,n+2,\dots,2n\}$. Therefore, $P = P^+ \cup P^-$ is the set of nodes other than the depot node.

Assumption 3-4-2: We are given a fleet of $V=\{1,2,\dots, |V| \}$ vehicles. The vehicles are heterogeneous and every vehicle transports a few containers from a given node, i , to a destination node, j ($j \neq i$). At the start of the process, vehicles are assumed to be empty.

Assumption 3-4-3: It is assumed the vehicles move with an average speed so that there are no Collisions, Congestion, Live-locks, Deadlocks [79] and breakdown problem.

Assumption 3-4-4: To load/unload the containers from the vessel or in the yard, a QC or RTGC is used. Every pick up/delivery node has a certain time window.

Assumption 3-4-5: We assumed the container jobs are distributed in the terminal so that each node is visited once only by a vehicle. In other word, a QC and RTGC are not busy in each node by different container jobs at the same time.

Assumption 3-4-6: In practice, it is not possible to serve every job. Hence, the objective function is to minimize the transportation costs, to serve each job within its time window as much as possible and to minimize the total number of jobs left at the end of process.

The following parameters are known at the beginning of the process:

TS_{v_0} : The times at which the vehicle v leaves the depot.

S : The processing time of a container job to be picked up or dropped off.

q_v : The capacity of vehicle v .

TT_{L_i, L_j} : The travel time from the physical location of node i , L_i , to physical location of node j , L_j (for each pair of i, j in N).

C_{L_i, L_j} : The cost of travelling from the physical location of node i , L_i , to physical location of node j , L_j (for each pair of i, j in N).

d_j : the number of container jobs to be moved from node j to node $n+j$.

$[a_i, b_i]$: The time window to pick up container jobs at node i .

$[a_{n+i}, b_{n+i}]$: The time window to deliver container jobs at $n+i$.

$[a_0, b_0]$: The time window of the vehicles to departure the depot.

$[a_{2n+1}, b_{2n+1}]$: The time window of the vehicles to back to the depot.

3.4.2 Decision variables and domains

X_{ijv} : 1 if vehicle v moves from node i to node j . otherwise it is 0.

Domain $(X_{ijv}) = \{0,1\}$, $i, j \in P$, $v \in V$.

F_j : the number of jobs that fulfilled at node j .

Domain $(F_j) = \{0,1,\dots,d_j\}$, $i, j \in P^+$, $v \in V$.

3.4.3 Constraints

To present the constraints and objective function, we need the following auxiliary variables:

Z_j : the number of jobs left at node i at the end of process. At the start of the process $Z_i=0$.

Y_{vi} : the load of vehicle v when it leaves node i . At the start of the process $Y_{v0}=0$.

Q_j : the number of jobs to be lifted or dropped off at node j .

These variables are determined by the following conditional statements:

$$(1) \left\{ \begin{array}{l} (X_{0jv} = 1) \cdot (d_j = F_j) \Rightarrow Z_j = 0, Q_j = Q_{j+n} = d_j, Y_{vj} = Q_j \\ (X_{0jv} = 1) \cdot (d_j > F_j) \Rightarrow Z_j = d_j - F_j, Q_j = Q_{j+n} = d_j - Z_j, Y_{vj} = Q_j \end{array} \right\} v \in V, j \in P^+$$

$$(2) \left\{ \begin{array}{l} (X_{ijv} = 1) \cdot (d_j = F_j) \Rightarrow Z_j = 0, Q_j = Q_{n+j} = d_j, Y_{vj} = Y_{vi} + Q_j \\ (X_{ijv} = 1) \cdot (d_j > F_j) \Rightarrow Z_j = d_j - F_j, Q_j = Q_{n+j} = d_j - Z_j, Y_{vj} = Y_{vi} + Q_j \end{array} \right\} v \in V, j \in P^+, i \in P, i \neq j$$

$$(3) X_{ijv} = 1 \Rightarrow Y_{vj} = Y_{vi} - Q_j, v \in V, i \in P, j \in P^-, i \neq j$$

The first set of the statements represents the number of jobs left and lifted at node j as well as the load of the vehicle when it leaves the first pickup point after the depot. The number of jobs left at node j is the difference between the number of jobs requested and the number of jobs fulfilled. The number of jobs to be picked up at node j and the number of deliveries at the destination node are updated. Additionally, the load of vehicle v when it leaves node j is equal to the number of jobs picked up at the node. The second set of the statements has a similar meaning but for when the vehicle goes to any pick up or drop-off point after the first pickup. The last set of the statements means that if the vehicle goes to any delivery point, its load is decreased by the number of deliveries.

TS_{vi} : The time at which the vehicle v starts service at node i ($TS_{v0}=0$). This variable is determined by the following conditional statements:

$$X_{0jv} = 1 \Rightarrow TS_{vj} = TS_{v0} + TT_{L0,Lj}, j \in P^+, v \in V$$

$$X_{ijv} = 1 \Rightarrow TS_{vj} = TS_{vi} + S \times Q_i + TT_{Li,Lj}, i, j \in P, v \in V$$

$$X_{i(2n+1)v} = 1 \Rightarrow TS_{v(2n+1)} = TS_{vi} + S \times Q_i + TT_{Li,L(2n+1)}, i \in P^-, v \in V$$

The first statement represents leaving the depot where the vehicles follow by a pickup point. The second statement shows that the vehicles can go to any pickup or delivery point after the first pickup. The last statement represents going the depot where the vehicles have a delivery before that. To calculate the starting service time at each node, the service time of the current node and the travelling time between the previous and current nodes have to be considered.

Constraint 3-4-1: Constraints on pick-up and delivery points.

$$\begin{aligned}\sum_{v \in V} \sum_{j \in N} X_{ijv} &= 1, i \in P^+ \\ \sum_{j \in N} X_{ijv} - \sum_{j \in N} X_{jiv} &= 0, i \in P, v \in V \\ \sum_{j \in N} X_{ijv} - \sum_{j \in N} X_{j(n+i)v} &= 0, i \in P^+, v \in V\end{aligned}$$

The first constraint ensures that each pick-up point is visited once by one of the vehicles. The second constraint indicates that if a vehicle enters a node it exits it. The third constraint ensures that if a vehicle visits a pickup node then it has to visit the associated delivery node.

Constraint 3-4-2: Constraints on the first and last visit points.

$$\begin{aligned}\sum_{j \in P^+} X_{0jv} &= 1, v \in V \\ \sum_{i \in P^-} X_{i(2n+1)v} &= 1, v \in V\end{aligned}$$

The first constraint ensures that the first visit of every vehicle is a pick up node. The second constraint ensures that the last visit of the vehicles is a delivery node.

Constraint 3-4-3: Constraints on the capacity of the vehicles.

$$Y_{vi} \leq q_v, v \in V, i \in P$$

The load of vehicle v when it leaves node i must not exceed the capacity of the vehicle.

3.4.4 Objective function

According to Assumption 3-4-6, the objective function is as follows:

$$MinCostJobs = \left\{ \sum_{v \in V} \left[w_1 \sum_{i \in P} \sum_{j \in P, j \neq i} X_{ijv} \cdot C_{Li,Lj} + w_2 \sum_{i \in P} |a_i - TS_{vi}|^+ + w_3 \sum_{i \in P} |TS_{vi} - b_i|^+ \right] + w_4 \sum_{i \in P^+} Z_i \right\}$$

The first term is the sum of transportation costs of the vehicles. The second and third terms are the penalty cost by the actual arriving of vehicle v to the node i earlier than the expected time and the penalty by the delay of the arriving time after the promised due time. These two last terms have impacts on the objective function provided that they are only positive. The last term is the jobs left at the end of process. Note that w_1 , w_2 , w_3 and w_4 are the weights of those four terms in the objective function.

3.5 Appointment times to eXternal Trucks (XTs)

A port usually serves as an interface and temporary storage of containers between ocean and land. In this way the main functions are to receive outbound containers from customers for loading into vessels, and unload inbound containers from vessels for picking up by consignees. The outbound containers are brought in by XT. The inbound containers are also received by XT. The problem here is to make appointment times for these XTs.

The flow of outbound containers is represented by Figure 3-5 [68]. These containers are brought in by customer's XTs into the terminal through the Terminal Gate (TG) where the containers and their documentations are checked. The TG then instructs the XT to go to the storage block where the container will be stored until the vessel arrives. The Yard Crane (YC) or RTGC working at that block removes the container from the XT and puts it in its storage position. When the time to load comes true, the YC removes the container from the stored position, puts it on an IT or AGV. Then, the IT or AGV carries the container to a QC for loading into the vessel. The flow of inbound containers is reverse as depicted in Figure 3-6.

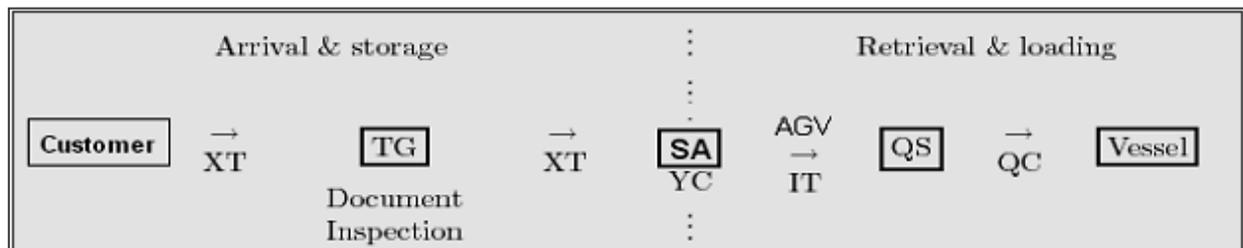


Figure 3-5: Flow of outbound containers (SA = Storage Area, QS = Quayside)

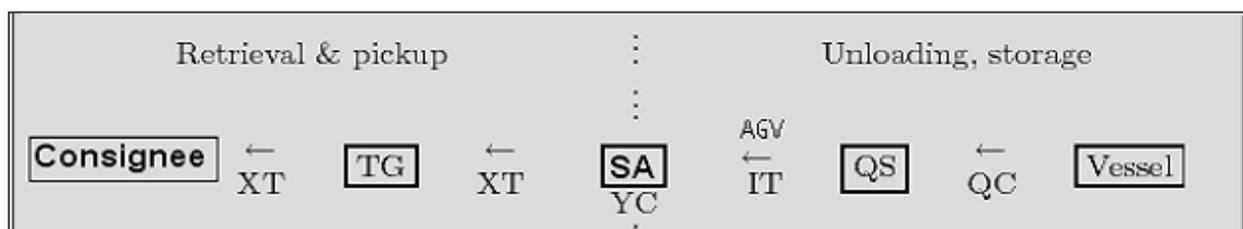


Figure 3-6: Flow of inbound containers (SA = Storage Area, QS = Quayside)

Murty et al. (2005) described a dispatching policy at the terminal gate [68]. According to their policy, the consecutive trucks are dispatched to different blocks in the storage yard, so that each block has adequate time to process the truck reaching it before the next truck sent to this block.

Also, the dispatching policy should distribute these trucks in all directions to ensure that the truck traffic on the roads is evenly distributed in all directions. In order to execute this dispatching policy, we considered a component in the objective function of Storage Space Assignment (see Assumption 3-2-3) to distribute the containers among blocks in the storage area.

3.5.1 Assumptions

In order to make appointments for the XTs, we consider the following assumptions:

Assumption 3-5-1: According to Assumption 3-2-6, the inbound containers are stored in the storage area (secondary storage) before they are picked up by their consignees. Also it is assumed the outbound containers are stored in the storage area before they are loaded to the corresponding vessels.

Assumption 3-5-2: According to the definition of SSCPI and SSCGD in Section 3.2.1, they are inbound containers in the storage area waiting for pickup by consignees and outbound containers before being allocated to storage area, respectively.

Assumption 3-5-3: The storage area has enough space to store all outbound containers in the planning horizon. Note the Storage Space Assignment (see Section 3.2.1) has considered this problem for the outbound containers.

The following parameters are known at the beginning of the planning horizon:

N: The total number of SSCPI containers over the planning horizon.

M: The total number of SSCGD containers over the planning horizon.

TSSCPI_i: The time at which the SSCPI container i is placed into the secondary storage area after discharging from the ships.

TPG_i: The processing time of a container i , including unloading/loading time and gating time.

T: The number of time periods in the planning horizon. The time period has to be greater than the maximum of TPG _{i} , $i=1,2,\dots,M+N$.

3.5.2 Decision variables and domains

The following decision variables are defined:

DT_i: Delivery time of SSCGD container *i* to the port.

$$\text{Domain } (DT_i) = \{1, 2, \dots, T\}$$

PT_j: Pick up time of SSCPI container *j* from the port.

$$\text{Domain } (PT_j) = \{1, 2, \dots, T\}$$

3.5.3 Constraints

Constraint 3-5-1: Delivery time of any SSCGD container to the gate and pick up time of any SSCPI from the terminal is different.

$$(DT_i + TPG_i \geq PT_j) \text{ OR } (PT_j + TPG_j \geq DT_i); \text{ for } j = 1, 2, \dots, N; \text{ for } i = 1, 2, \dots, M$$

Constraint 3-5-2: Any SSCPI container can be picked up after it is moved to the storage area.

$$PT_j \geq TSSCPI_j, \text{ for } j = 1, 2, \dots, N$$

Constraint 3-5-3: Delivery time of any two SSCGD containers and pick up time of any two SSCPI containers are different.

$$(DT_i + TPG_i \geq DT_{i'}) \text{ OR } (DT_{i'} + TPG_{i'} \geq DT_i); \text{ For } i, i' = 1, 2, \dots, M, i \neq i'$$

$$(PT_j + TPG_j \geq PT_{j'}) \text{ OR } (PT_{j'} + TPG_{j'} \geq PT_j); \text{ For } j, j' = 1, 2, \dots, M, j \neq j'$$

3.5.4 Objective function

The objective function of this decision is to minimize the terminal gate's cost. In fact, delivery of the outbound containers and pickup of the inbound containers should be carried out as soon as possible in the planning horizon. This function is written as follows:

$$\text{MinCostGate} = w_1 \sum_{i=1}^N DT_i + w_2 \sum_{i=1}^M PT_i$$

The first term is the sum of time periods that spend on delivery time of the outbound containers. The second term is the sum of time periods that spend on pick up time of the inbound containers. Note that w_1 and w_2 are the weights of those two terms in the objective function.

3.6 Container terminals over the world, a survey

In this section, we summarized the latest research in some of the major container terminals in the world. Table 3-1 shows this summary. In the table the first and third columns show the port name and authors respectively, in where and who has done the research. The second column shows decisions and solutions for the problem. From this table it can be clearly seen that the most of the container terminals considered their vehicles problems in the research.

Table 3-1: Container Terminals around the world and their decisions

Ports	Decisions and Solution Method	Authors (Year)[Ref. No]
1. Port of Hamburg, Germany 2. Port of Bremen, Germany	<ul style="list-style-type: none"> Storage Space Allocation (MILP) Generating Scenarios (Simulation) Vehicle Scheduling (Evolutionary/Genetic Algorithm) 	Steenken et al(2001) [88] Hartman (2002) [34] Böse et al (2000) [8]
1. Contship La Spezia , Italy 2. Maritime Terminal in Genoa, Italy	<ul style="list-style-type: none"> Storage Space Allocation (Simulation) Yard Storage Management (Integer Programming) 	Gambardella et al (1998) [27] Ambersino et al (2002) [5]
Port of Pusan, Korea	<ul style="list-style-type: none"> Berth Allocation (MILP) Berth Allocation and Quay Crane Assigning (Lagrangean Relaxation, Dynamic Programming) Dispatching of Automated Guided Vehicles (Linear Programming Relaxation) 	Moon (2001) [65] Park & Kim (2003) [73] Wook & Hwan (2000) [103]
1. Port of Rotterdam, The Netherlands 2. Port of Amsterdam, The Netherlands	<ul style="list-style-type: none"> Vehicle and Crane Scheduling, but its data has been collected by simulation (Branch and Bound/ Beam Search Heuristic Method) Deadlock prediction and avoidance (Wait and Proceed strategy). 	Meersman et al (2001) [61] Meersman et al (2001)[62] Moorthy et al (2003) [66]
Port of Singapore, Singapore	<ul style="list-style-type: none"> Routing AGVs(Sorting Techniques) Whole System (Simulation) Dispatching of Automated Guided Vehicles (Network Flow Model) 	Qiu & Hsu(2000) [81] Liu et al(2002) [57] Cheng et al(2003) [14]
Port of Los Angeles, USA	<ul style="list-style-type: none"> Vehicle Scheduling and Routing (Dynamic Programming and Genetic Algorithms). 	Ioannou et al(2002) [42]
Hong Kong Container Terminal No 9 (New)	<ul style="list-style-type: none"> RTGC Deployment in the yard (Vogel Solution) Storage Space Assignment and Vehicle Routing (Linear Programming). Storage Space Allocation (Integer Programming). Crane/RTGC Deployment in the yard (MILP and Lagrangean relaxation). 	Murthy et al (2005) [68] Zhang et al(2001) [106] Zhang et al(2002) [107]
Real Size Terminal	<ul style="list-style-type: none"> Vehicle Scheduling (Heuristic Algorithm/Lagrangean Relaxation) 	Zhang et al (2002) [108]
Virtual Terminal	<ul style="list-style-type: none"> Rescheduling of Quay Cranes and Vehicles (Distributed-Agent System) 	Thurston & Hu(2002)[90]
Real Port (Not mentioned)	<ul style="list-style-type: none"> Whole System (Multi-Agent System, Not Implemented) 	Rebollo(2000) [84]

3.7 Solution methods and evaluation of the decisions

In the previous sections, we formulated the problems. There are three important phases to provide a practical software for the decisions: requirement analysis, selecting the design architecture and solution methods. In this section, we briefly review the first two phases and suggest two frameworks for the solutions. After that, the solutions for the problems are summarized and some indices to evaluate each decision are presented.

The first step is requirement analysis. For this phase, we propose to provide a program to animate or simulate some operations in the terminal. The program will be very useful to understand the problem and to generate some input data for next steps. In the problem specification, some operations or decisions should be synchronized to each other, if two or more decisions are likely to be studied together. For example scheduling and routing of vehicles (the problem in Section 3.4), can be combined with the storage space assignment (the problem in Section 3.2). In the complex system, a few parameters should be considered in the integrated model to synchronize the decisions. Tsang (1998) suggested some methods to represent time and space [95]. Gambardella et al. (1998) [27], Hartman (2002) [34] as well as Thurston and Hu (2002) [90] applied some scenarios for simulation of terminal systems with several restrictions. Additionally, Kim et al. (2000) introduced a simulation-based test-bed to test various control rules. They suggested a control system consists of ship operation manager, system controllers for automated guided vehicle, automated yard crane, and quay crane [51]. Three control strategies, synchronization, postponement, and re-sequencing, were introduced in the paper as promising alternatives for controlling traffics of vehicles.

The second phase is to design a architecture for the system. Two distinct systems architecture including *Centralised system* and *Distributed system* have been suggested by Thurston and Hu (2002) [90]; the latter was implemented by agents. For the first architecture, Tsang (1993) provided different solutions to satisfy the constraints of every Constraint Satisfaction Problem (CSP) [93]. The solutions are divided into four groups, including problem reduction, complete search methods, stochastic method and synthesize the solutions. For the distributed system, Yokoo et al. (1998) formalized Distributed Constraint Satisfaction Problem [104]. They also developed asynchronous backtracking, asynchronous weak-commitment search solutions, distribution breakout and distributed consistency algorithms for these kinds of problem [105].

Some well-known software such as GAMS (Generalised Algebraic Modeling System), LINDO (Linear INteger and Discrete Optimizer), and others can be used to solve the problems. But our suggestion is ILOG Solver or HOTFRAME since a lot of the classical, heuristics and meta-heuristics approaches have been used in their components library [26, 40]. The components of ILOG Optimization Suite rely on mathematical programming and constraint-based optimization. A core of large number of successfully deployed applications was provided in ILOG. In addition, it is the most comprehensive portfolio of optimization components for efficient resource allocation, involved in scheduling and planning of resource utilization. For the second suggestion, Fink and Voß (2002) surveyed, designed and implemented HOTFRAME [26], a Heuristic OpTImisation FRAMEwork that provides reusable software components in the meta-heuristics domain. The framework architecture, in which has been implemented by C++, defined the collaborations among software components, in particular with respect to the interface between meta-heuristic components and problem-specific components. Also in this framework different applications have been considered. The scope of HOTFRAME comprises meta-heuristic solutions such as iterated Local Search, Simulated Annealing method and its variations, different kinds of Tabu Search (e.g. static, strict, and reactive), Evolutionary Algorithms, Candidate Lists, Neighbourhood Depth variations, and Pilot Method [98, 26]. The primary design objectives of HOTFRAME have provided run-time efficiency and a high degree of flexibility with respect to adaptations and extensions. Then, their developers built generic meta-heuristic components, which are parameterized by some concepts such as the solution space, the neighbourhood structure, or Tabu-criteria. Note that in C++, generic components can be implemented as template classes or a function, which enables achieving abstraction without loss of efficiency.

Several different solutions methods can be applied to the problems. Tsang (1995) provided a comparative study of scheduling techniques [94]. In that paper, the techniques have been divided into two groups. The first group consists of Linear Programming, Branch and Bounds and Tabu Search, which are studied extensively in Operation Research. In the second group, some other techniques such as Hill Climbing, Simulated Annealing, Connectionism, Expert Systems and Genetic Algorithm have been studied in Artificial Intelligence. Tsang summarized his studies by Table 3-2, including considerations in choosing between the major scheduling techniques.

Table 3-2: Considerations in choosing between major scheduling techniques [94]

Solution Methods	General Considerations	Major technique-specific considerations
Linear Programming	<ul style="list-style-type: none"> Used for optimisation with linear functions. Intractable. 	<ul style="list-style-type: none"> Problem must be specified by a set of inequalities
Branch-and-Bound	<ul style="list-style-type: none"> Used for optimisation. Intractable 	<ul style="list-style-type: none"> Requires heuristic for pruning. Ordering of branches is important.
Constraint satisfaction	<ul style="list-style-type: none"> Most existing algorithms used for finding single or all solution satisfying constraints. Both complete and incomplete algorithms available. 	<ul style="list-style-type: none"> Large number of algorithms available. Particularly useful when problem involves non-trivial amount of constraints.
Hill climbing Simulated Annealing Tabu Search	<ul style="list-style-type: none"> Useful for both constraint satisfaction and optimisation when near-optimal solutions are acceptable. Flexible in computation time, this makes them widely useful. Hill climbing could be trapped in local optima. Simulated annealing and Tabu search attempt to escape from local optimal 	<ul style="list-style-type: none"> Requires a neighbourhood function which is crucial to its effectiveness. Neighbourhood function is crucial to its effectiveness. Cooling schedule could be important. Effectiveness mainly depends on strategy on Tabu-list manipulation. Representation is crucial. Effectiveness could be sensitive to choice of parameters values and operators.
Genetic Algorithms	<ul style="list-style-type: none"> Useful for finding near-optimal solutions. Requires non-trivial time, but hopefully will search a wider part of the solution space. 	<ul style="list-style-type: none"> Representation is crucial. Effectiveness could be sensitive to choice of parameter values and operators.
Connectionisms	<ul style="list-style-type: none"> Useful for satisfiability problems or for finding near optimal solutions. Good potential for parallel implementation which may suit real time application. 	<ul style="list-style-type: none"> Set up and network updating mechanism are crucial to it effectiveness. Specialized network may be expensive to build.
Expert systems	<ul style="list-style-type: none"> Wide range of applicability, can be tailor-made to meet the requirements (including time and optimality requirement) Power comes from domain-specific knowledge 	<ul style="list-style-type: none"> Expert knowledge elicitation is important and may be difficult. Conflict resolution may be non-trivial.

Another research group, Gunadi et al. (2002), studied different types of problems and solutions to vehicle routing problem [32]. They classified the solutions into three groups; Operations Research algorithms, Artificial Intelligence techniques and Decision Support System solutions. They summarized their studies by Table 3-3.

Table 3-3: Summary of Vehicle Routing Problems and Solutions [32]

Classifications	Solutions & Authors	Application	Characteristics
Operation Research Algorithms	Sweep Algorithm: Gillet & Miller (1971)	<ul style="list-style-type: none"> Goods delivery vehicle Public bus with capacity constraint 	<ul style="list-style-type: none"> Minimum total length of route is a major concern Additional distance may occur Demand is uncertain
	Matching Based Savings Algorithm: Desrochers & Verhoog (1990)	<ul style="list-style-type: none"> Goods delivery vehicle 	<ul style="list-style-type: none"> Solving fleet size and mix vehicle Short distance is a major concern
	Chain Exchange Principle: Fahrion & Wrede (1990)	<ul style="list-style-type: none"> Goods delivery vehicle Vehicle routing problem with time windows 	<ul style="list-style-type: none"> Number of customer is known Time constraint is major concern
	Branch and Bound Algorithm: Laporte et al. (1992)	<ul style="list-style-type: none"> Shortest Path Problem and goods delivery 	<ul style="list-style-type: none"> Short distance is major concern Focuses on the minimum number of visit
	New Crossover: Uchimura & Sakaguchi (1995)	<ul style="list-style-type: none"> Shortest round trip tour 	<ul style="list-style-type: none"> Short distance and time constraint
	Parallel Branch and Bound Algorithm: Lau & Kumar (1997)	<ul style="list-style-type: none"> Vehicle routing problem on Networks of Workstation 	<ul style="list-style-type: none"> Minimum total distance for goods delivery
	Dijkstra Method: Ikeda et al. (1994)	<ul style="list-style-type: none"> Shortest-Path Problem 	<ul style="list-style-type: none"> Short distance is major concern All-directional approach
	Modified Dijkstra Method: Eklund et al. (1996)	<ul style="list-style-type: none"> Emergency service vehicles routing 	<ul style="list-style-type: none"> Shortest path is the main concern
	Tabu Search : Taillard et al. (1996)	<ul style="list-style-type: none"> Shortest-Path Problem One depot VRP 	<ul style="list-style-type: none"> Short distance is major concern Number of customer is known
	Tabu Search : Garcia et al. (1993)	<ul style="list-style-type: none"> VRP with time windows constraint 	<ul style="list-style-type: none"> Solving VRP with time windows constraint Demand is known.
	A* Algorithm	<ul style="list-style-type: none"> Shortest-Path Problem 	<ul style="list-style-type: none"> Shortest distance is major concern
	2-opt* Exchange: Potvin & Rousseau (1995)	<ul style="list-style-type: none"> VRP with time windows Best implemented for travelling salesman problem 	<ul style="list-style-type: none"> Time constraint is a major concern
	Or-opt-1 & Or-opt exchange	<ul style="list-style-type: none"> Goods delivery vehicle 	<ul style="list-style-type: none"> Focus on node exchange Number of customer is known;
Artificial Intelligence Techniques	GENESIS: Thangiah & Gubbi (1993)	<ul style="list-style-type: none"> Goods delivery vehicle 	<ul style="list-style-type: none"> Demand is known
	Niche Search: Pedroso et al. (1998)	<ul style="list-style-type: none"> Goods delivery vehicle 	<ul style="list-style-type: none"> Route is selected based on time average
	Bimodal Dial-A-Ride: Liaw et al. (1996)	<ul style="list-style-type: none"> Paratransit vehicle routing 	<ul style="list-style-type: none"> Involves transit between paratransit vehicle and fixed bus route
Decision Support System Solutions	Micro-ALTO: Potvin et al. (1994)	<ul style="list-style-type: none"> Goods delivery vehicle 	<ul style="list-style-type: none"> Concerns on minimum operational cost, service quality and service time
	Fuzzy-neural approach: Takahashi et al. (1995);	<ul style="list-style-type: none"> In-vehicle route guidance system 	<ul style="list-style-type: none"> Route selection based on driver's preference
	Fuzzy Route Choice: Shaout et al. (1993), Pang et al. (1995)	<ul style="list-style-type: none"> Automotive Navigation System, Dynamic Route Guidance 	<ul style="list-style-type: none"> Route selection based on driver behaviour

For the first group, they considered the algorithms of Sweep, Matching Based Savings, Chain-exchange, Branch and Bound, Crossover, Tabu Search, Dijkstra, A* and 2-opt* exchange

heuristic. They expressed that GENESIS, Niche Search and Biomodal Dial-A-Ride methods are in the second group. They believed that Micro-ALTO method, fuzzy neural approach and fuzzy Route Choice are in Decision Support System solutions.

Qiu et al. (2002) provided a survey of scheduling and routing algorithms for AGVs [79]. They showed similarities and differences between scheduling and routing AGVs and related problems like the vehicle routing problem, the shortest path problem and scheduling problem. They classified algorithms in groups for general path topologies, for path optimization, for specific path topologies and dedicated scheduling algorithms. In the general path topologies, the methods adopted have been classified into three categories: (a) Static methods, where an entire path remains occupied until a vehicle completes the tour; (b) Time-window-based methods, where a path segment may be used by different vehicles during different time-windows; and (c) Dynamic methods, where the utilization of any segment of path is dynamically determined during routing rather than before routing as with cases (a) and (b). In the path optimization, the methods have been classified into three categories: (d) 0/1 integer programming model, where the path layout problem is as a binary integer programming model with considerations of the given facility layout and Pickup/Delivery stations; (e) Intersection graph method, where only a reduced subset of nodes in path network is considered and only intersection nodes are used to find optimal for solving AGV; (f) Integer LP model, where the problem is modeled as an Integer linear programming of selecting the path and location of Pickup/Delivery stations. In the specific path topologies, the three different layouts could be considered: Linear, Circle and Mesh topology. Tables 3-4 to 3-6 summarize the works reviewed in the paper.

Moreover, Voß (2000) provided high-quality solutions to important applications in business, engineering, economics and science in reasonable time-horizons [98]. A family of meta-heuristics search methods including simple Local Search, Adaptive Memory Procedures, Tabu Search, Ant System, Greedy Randomised Adaptive Search, Variable Neighbourhood Search, Evolutionary Methods, Genetic Algorithms, Scatter Search, Neural Network, Simulated Annealing and their hybrid have been presented briefly in the study. Also important references for solving combinatorial optimisation problems have been provided in the paper.

Table 3-4: Summary of work reviews for AGVs in General Path Topologies [79]

Authors	Gaskins & Tanchoco (1987)	Kaspi & Tanchoco (1990)	Goetz & Egbelu (1990)	Sinriech & Tanchoco (1991)
Problems Solved	Path optimization to minimize total distance traveled by loaded vehicles	Path optimization to minimize total distance traveled by loaded vehicles	Path optimization to minimize total distance traveled by loaded and unload vehicles	Path optimization to minimize total distance traveled by loaded vehicles
Basic Algorithms	Zero-one integer programming	Zero-one integer programming; branch-and bound	Integer linear programming	Intersection Graph Method; Branch-and-bound
Path Topologies	General	General	General	General
Path Direction	Uni-directional	Uni-directional	Uni-directional	Uni-directional
Advantages	Very easy to implement for a fleet of AGVs with the same origins and destinations	An improvement of the approach in [Gaskins & Tanchoco 1987]; reduced computation; optimality guaranteed	Problem size is reduced; distance traveled by unloaded vehicles is considered together; optimality is hence better ensured	An improved model of that proposed in [Kaspi & Tanchoco 1990]; reduced number of problem branches; optimality guaranteed
Disadvantages	Conflicts may occur when there are AGVs with different origins and destinations; heavy computation; low system throughput	Distance traveled by unloaded AGVs is not considered; low system throughput; still heavy computation	Routing control and vehicle number are not considered in the study which are important for AGV systems	Only intersection nodes of the path network are considered; optimal solutions may be missed

Table 3-5: Summary of Algorithms for AGVs in Specific Path Topologies [79]

Authors	Tanchoco & Sinriech (1992)	Lin & Dgen (1994)	Sinriech & Tanchoco (1994)	Hsu & Huang (1994)
Problems Solved	Optimizing the path layout configuration in a closed single circle	Routing AGVs among several non-overlapping closed circles; finding shortest travel time path	Routing AGVs among several non-overlapping, path segments; finding shortest travel time path	Route planning for basic routing functions on several specific basic path topologies
Basic Algorithms	Integer programming	The task-list time-window algorithm	Integer programming	-
Path Topologies	Closed single-circle	Multi-circle	Segmented path topology	Linear array, ring, H-tree, star, 2D-mesh, n-cube, cube-connected cycles, complete graph,
Path Direction	Uni-directional	Bi-directional or Unidirectional	Bi-directional or Unidirectional	Bi-directional
Advantages	Routing control is very easy; no conflicts or deadlocks will occur; easy for implementation	Easy for routing control since every circle is served by a single vehicle;	An alternative design of that in [Lin & Dgen 1994]; relatively low value of flow's distance	Give the time and space complexities for basic routing functions which are upper- bounded by $O(n^2)$ and $O(n^3)$ respectively
Disadvantages	Low system throughput ; only suitable for small system	Low system throughput; additional cost needed for transit device between two adjacent circles; indirect transportation may cause delay	Low system throughput with one vehicle serving in a segment; additional cost for transit device; indirect transportation may cause delay	Routing control not given in detail; the assumption of arbitration capability for every buffer is too idealized

Table 3-6: Summary of Static and Dynamic Routing Algorithms for AGVs in General Path Topology [79]

	Static Routing Problem				Dynamic Routing Problem	
Authors	Broadbent et al. (1985)	Daniels (1988)	Huang et al. (1989)	Kim & Tanchoco (1991, 1993)	Taghaboni & Tanchoco (1995)	Langevin et al. (1996)
Problems Solved	Finding conflict-free shortest time routes for AGVs	Finding conflict-free shortest time routes for AGVs	Finding conflict-free shortest time routes for AGVs	Finding conflict-free shortest time routes for AGVs	Finding a conflict-free route for AGVs	Integrated solution for AGV dispatching, conflict-free routing and scheduling
Basic Algorithms	Dijkstra's shortest path algorithm	Partitioning shortest path algorithm	Labeling Algorithm	Dijkstra's shortest path algorithm; <i>conservative myopic strategy</i>	Incremental route planning	Dynamic programming
Computational Complexity	$O(N^2)$ (average case)	$O(N \times A)$ (average case)	$O((N+A)^2 \text{ Log}(N+A))$ (average case)	$O(V^2 \times N^2)$ (worst case)	Not available; Not guaranteed Optimality	Not available; Not guaranteed Optimality
Path Direction	Bi-directional	Bi-directional	Bi-directional	Bi-directional	Bi-directional & Uni-directional	Bi-directional
Advantages	Easy to execute	Easy to execute and faster than Broadbent's	Time windows are used for every node; the utilization of path segments are increased	Easy to execute and control; fast	Relatively fast in routing decision	Easy to execute and control
Disadvantages	Heavy computation; low utilization of path segments	Heavy computation; low utilization of path segments; may cause failure in finding routes that actually exist	Heavy computation; large amount of data of converted network to maintain	Heavy computation; large amount of data of path network to maintain	Low efficiency when the numbers of tasks and vehicles increase; also no optimal routing solutions could be guaranteed;	Since only two vehicles are allowed in the system, the system throughput and path utilization could be very low; only suitable for very small system with a few stations

N – The number of nodes in the path network; A – The number of arcs in the path network; V – the number of AGVs.

Furthermore, hyper-heuristic methods emerged to solve scheduling problems. Burke et al. (2003) defined hyper-heuristic idea based on the heuristics approach [10]. The main motivations behind development of the hyper-heuristic were to automate scheduling methods and to raise the level of generality. They suggested a framework for the hyper-heuristic and investigated it on various instances of two distinct timetabling and rostering problems. In the framework, heuristics compete using rules based on the principles of reinforcement learning. A Tabu list of heuristics was also maintained which prevented certain heuristics from being chosen at certain times during the search. In another paper [49], Kendall and Hussin (2005) investigated a Tabu search based hyper-heuristic for solving examination timetabling problems. They claimed that their approach is able to produce good quality solutions.

In recent years, agent systems have been used to solve scheduling problem. Cowling et al. (2004) presented a multi-agents system and used it as a case study for integrated dynamic scheduling of steel milling and casting [20]. In the system, a set of heterogeneous agents was used to integrate and optimize a range of scheduling objectives related to different processes of steel production, and could adapt to changes in the environment while still achieving overall system goals. In another papers [71, 72], Quelhadj et al. (2003, 2005) described a negotiation protocol in the multi-agent system. The purpose of that protocol was to allow the agents to cooperate and coordinate their actions in order to find globally near-optimal robust schedules, whilst minimising the disruption caused by the occurrence of unexpected real-time events.

In some situations when scheduling problem is dealing with imprecision and uncertainty, fuzzy sets are employed. Petrovic and Fayad (2004) described a fuzzy Shifting Bottleneck Procedure (SBP) hybridised with genetic algorithm for a real-world job-shop scheduling problem [75]. In each iteration, the SBP selects a machine and the genetic algorithm proposed a sequence of job's operations to be processed on that machine. In another paper, Petrovic et al. (2005) proposed an algorithm for a real-world job shop-scheduling problem, where both lot-sizing and batching processes were considered [74]. A fuzzy rule-based system was developed for determining lot sizes, where the input variables were workload on the shop floor, size of the job and its urgency. A fuzzy multi-objective genetic algorithm was developed to generate schedules of jobs whose processing times and due dates were imprecise and modelled by using fuzzy sets. A genetic algorithm took into consideration the determined size of lots for jobs, and considered batching together jobs of similar characteristics in order to reduce the required set-

up time. The objectives considered were to minimize average tardiness, number of tardy jobs, set-up times, idle times of machines and throughput times of jobs.

In order to evaluate the decisions, different indices may be considered to measure efficiency of the terminal. Liu et al. (2002) studied four automated container terminals, Port of Rotterdam, Port of Hamburg, Port of Hong Kong and Port of Singapore, and then evaluated their operations by simulation [57]. They evaluated ship turnaround time, throughput of terminals, gate utilization, idled time of yard crane and buffer cranes, dwelling times of containers and average cost of a container during the simulation time.

Another research group, Inoannou et al. (2001) proposed a microscopic simulation model [43]. In their paper an ACT (Automated Container Terminal) system was proposed. They collected data from a conventional terminal and simulated the ACT system for the same operational scenario in order to evaluate, and compare their performances. A cost model was also developed to calculate the average cost per container. They assessed the performance of the model by throughput (moves per hour per quay crane), throughput per acre, annual throughput per acre (number of processed TEUs per acre per year), ship turn-around time, truck turn-around time, gate utilization, container dwell time, idle rate of equipment.

Additionally, Duinkerken and Ottjes (2000) implemented a simulation model for automated container terminal and applied their model to Delta Sealand container terminal of ECT Rotterdam [23]. Their objectives was to determine the sensitivity concerning a number of parameters like number of AGVs, maximum AGV speed, crane capacity and stack capacity. They concluded that the most critical performance indicators are average number of moves per hour per quay crane, QC-utilization (percentage of time that the quay crane is not waiting for AGVs) and average trip duration ratio (the ration between the actual duration of a trip divided by the technical trip time-Distance/Speed) and averaged over all connections between the yard-side and quay-side.

Here, we provide some indices to evaluate the decisions as Table 3-7. The right column of the table lists corresponding indices to evaluate each of the five defined decisions.

Table 3-7: Some important indices to evaluate the decisions in the container terminals

Decisions	Indices
Allocation of berths to arriving vessels and quay cranes to docked vessels	<ul style="list-style-type: none"> ▪ Ship around time ▪ Throughput of Terminal (container/ship) ▪ Idle Time of QCs ▪ Total Waiting Time of QCs ▪ Berths and QCs Utilization ▪ Average cost per ship
Storage Space Allocation	<ul style="list-style-type: none"> ▪ Average size of block in the yard ▪ Largest and Smallest Block in the yard ▪ Average Cost of containers in the yard ▪ Container dwell time
RTGC deployment in the yard	<ul style="list-style-type: none"> ▪ Idle rate of yard cranes or RTGCs ▪ Maximum, minimum and average workload in the yard ▪ Average movement of RTGCs in the yard
Scheduling and Routing of Vehicles	<ul style="list-style-type: none"> ▪ IT or AGV turnaround time ▪ Average transportation cost per container ▪ Number of AGVs used ▪ Number/Percentage of idle AGVs ▪ Total Waiting Time of AGVs ▪ Total Delay Times of AGVs ▪ Route Utilization ▪ Average trip duration ratio ▪ Longest and Shortest trip ▪ Number of trips for each vehicle ▪ Percentage of Moving vehicle with/without container
Appointment times to XTs	<ul style="list-style-type: none"> ▪ Gate utilization ▪ Container dwell time

3.8 Summary and conclusion

In this chapter, we systematically surveyed the literature over decisions in container terminals. The literature also includes solutions, implementation and performance. The five scheduling decisions in Chapter 2 have been formulated as CSOPs. The solutions have been classified and summarized. Two frameworks for the implementation have been suggested. The latest researches around the decisions in some of the major container terminals have been summarized. From the summarized table (see Table 3-1), we observed that most container terminals have considered their vehicles in the research. Therefore, it is one of the most important and challenging problems in the ports.

For the next stage of this research, we will focus on scheduling problem of Automated Guided Vehicles in the port. It is clear that any implementation of those decisions requires additional studies where the assumptions should be refined and adapted with particular container terminal.

Chapter 4: Scheduling of AGVs and Its Problem Formulation

This chapter focuses on scheduling problem of Automated Guided Vehicles (AGVs) in the container terminals. The problem is to deploy several AGVs in a port to carry many containers from the quay-side to yard-side or vice versa. This problem is defined in Section 4.2 and is formulated as a Minimum Cost Flow (MCF) model in Section 4.5 of this chapter.

4.1 Reasons to choose this problem

In the past few decades, much research has been devoted to technology of AGVs system, both in hardware and software [79]. Nowadays they have been become popular over the world for automatic material-handling and flexible manufacturing systems. Qiu et al. (2002) surveyed the scheduling and routing algorithms for AGVs. One of their suggestions for future research is to develop more efficient algorithms for different path topologies, where AGVs are employed [79]. These unmanned vehicles are also increasingly becoming common mode of container transport in the seaport [79]. Moreover there are some other reasons for concentration on this decision including:

- The efficiency of a port is directly related to the amount of time that each vessel spends in the port. A major challenge in the port management is to reduce the turnaround time of the container ships. If the management can use the AGVs with full efficiency at minimum waiting and travelling times, the performance of the port is increased.
- However, most of existing scheduling and routing solutions are applicable to a small number of AGVs [79]. Although major of references in the paper were over use of AGVs in material handling systems, we investigated the latest research in container terminals. The number of AGVs in the problems, which have been experienced by Wook and Hwan (2000) [103], Böse et al. (2000) [8], Grunow et al. (2004) [31], Thurston and Hu (2002)

[90] and Cheng et al. (2003) [14] were 5, 5, 6, 12 and 80, respectively. The largest problem in the recent research experienced was a problem with 100 vehicles. This experiment has been done by Zhang et al. (2002) [108], but the problems were static. When the number of container jobs and AGVs increase, we need to find some efficient solutions to tackle over the huge search space of this problem.

- We believe that some of its solutions and algorithms can be applied to other transportation systems such as Pickup/Delivery system in real time.
- From Table 3-1, it can be seen that most container terminals have considered this problem in their research.
- Decreasing costs of the terminal, speed up the transportation system inside the port, rising customer demand and globalisation of trade outside the terminal are affected by making a good operational plan for the AGVs.

4.2 Assumptions

The problem is to transport many containers in the port from the storage areas to the berth or vice versa by AGVs in their appointment times. Each container job involves the loading of the container onto the AGV, the movement of the vehicle to the destination, and the unloading of the container by the QCs or RTGCs.

In order to define and formulate the decision, the following assumptions and notations are considered:

Assumption 4-1: The layout of a port container terminal can be visualized in Figure 4-1 [103]. In this example, there are five working positions of QCs in the berth (Seaside workplace) and five yard blocks in the storage area for containers (block A to block E). In the figure, the locations of RTGCs or yard cranes for unloading or picking up the containers are in front of each block. The path between two points is not necessary unique and the system controller may change the route of AGVs to designated points, due to congestion in the next lane or junction.

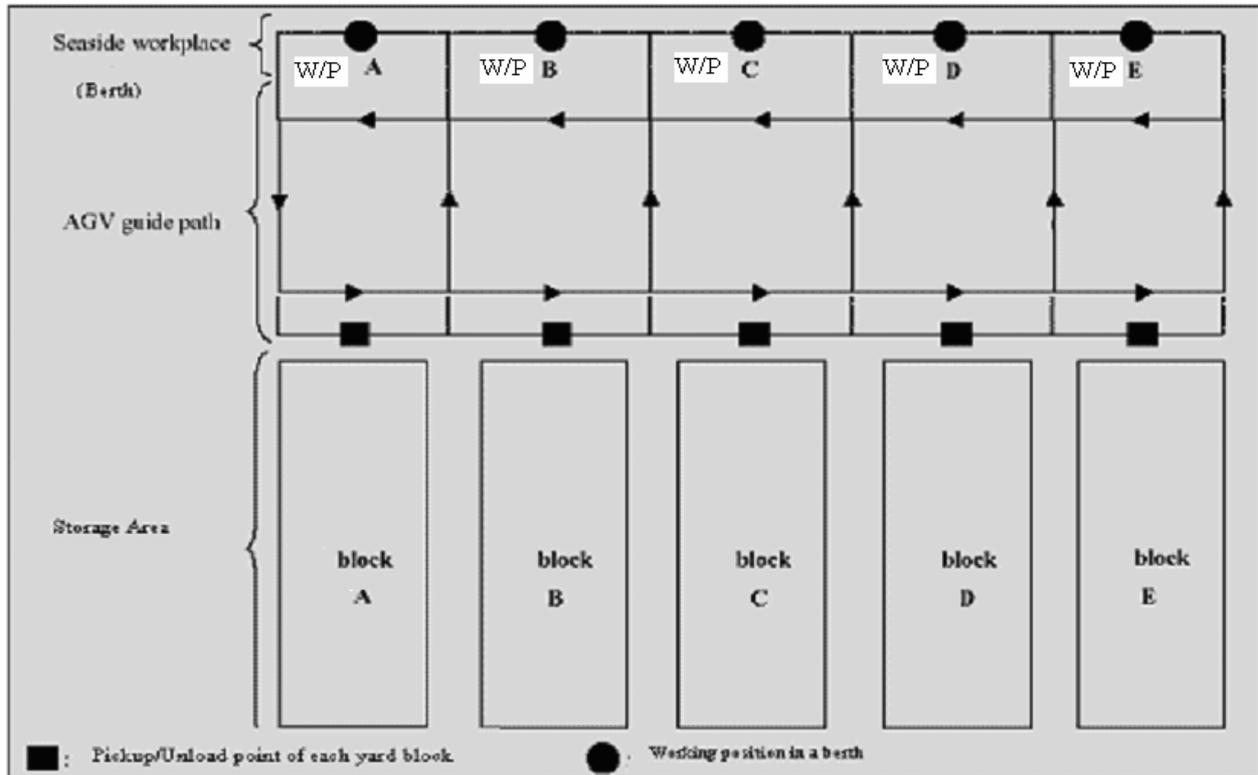


Figure 4-1: Layout of the container terminal

Assumption 4-2: We assume that the problem involves only one ship and therefore the number of QCs and their location don't change until all container jobs under consideration for the docked-ship are completed.

Assumption 4-3: Generally the following listed phenomena are happened when scheduling and routing AGVs are being studied [79]:

- **Collisions:** When more than one AGV attempt to occupy the same segment of the path at the same time, there is potentially a collision. Figure 4-2(a) shows two examples.
- **Congestion:** Congestion arises at a location where there is insufficient resource such that for a period of time there are too many vehicles in a path. Figure 4-2(b) depicts such a case. Congestion must be reduced or eliminated because it will produce a lower throughput of the system or even leads to deadlock.
- **Live-locks:** As shown in Figure 4-2(c), a live-lock may arise at the junction where the horizontal stream of traffic is given higher priority over the vertical one. In this case, the queue in the vertical line never moves.

- **Deadlocks:** A deadlock will arise when multiple AGVs mutually wait for the release (which will never occur) of the resource held by the others. Figure 4-2(d) shows two cases: local deadlock and non-local deadlock.

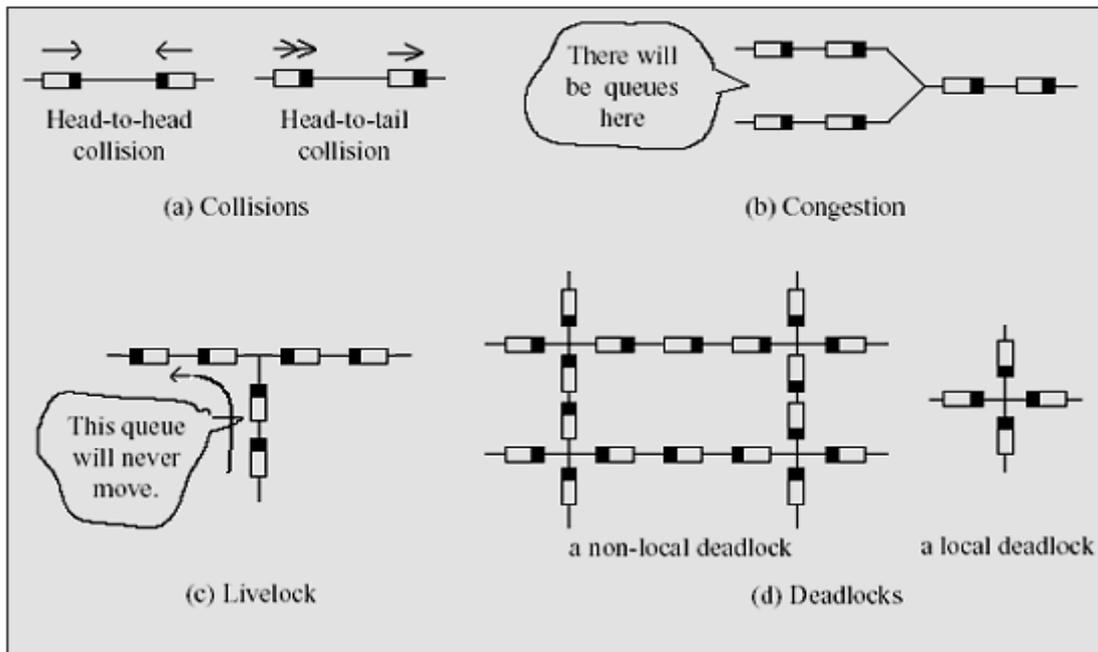


Figure 4-2: Phenomena arising in scheduling and routing of AGVs [79].

We assume that the AGVs are reliable and travel at certain predetermined average speed so that Collisions, Breakdowns, Live-Locks as well as Deadlocks can be eliminated in our model.

Assumption 4-4: There are several paths between every combination of Pickup (P) /Drop-off (D) points for the AGVs, according to our layout (see Figure 4-1). But we assume that at any time, the travel time between every two points is provided in a table like Table 4-1 [103]. In the table the notation W/P shows Working Position of the cranes in the berth.

Table 4-1: Example of traveling time (second) between two different points in the port

To → From ↓	Block A	Block B	Block C	Block D	Block E	W/P A	W/P B	W/P C	W/P D	W/P E
Block A	-	30	60	90	120	150	195	200	225	265
Block B	80	-	30	60	90	175	165	205	195	235
Block C	110	80	-	30	60	145	135	175	165	205
Block D	140	110	80	-	30	175	165	145	135	175
Block E	170	140	110	80	-	205	195	175	165	145
W/P A	205	175	145	175	205	-	50	90	80	120
W/P B	215	185	155	185	215	10	-	80	70	110
W/P C	225	205	175	145	175	30	20	-	50	90
W/P D	235	215	185	155	185	40	30	10	-	80
W/P E	265	235	205	175	145	60	50	30	20	-

Assumption 4-5: There are M AGVs in the container terminal. Every AGV can transport only one container. This simplification, however, ensures that the problem remains tractable and that an efficient operational plan can be devised and implemented in real time. In fact, most of the current literature focuses on AGVs with unit capacity. This is often the reality in container terminals [14]. Henceforth, we consider unit capacity for the AGVs.

Assumption 4-6: RTGCs or yard crane resources are always available [13], i.e., the AGVs will not suffer from delays in the storage yard location due to waiting for the yard cranes. This is not a restrictive assumption in the real implementation, since a good yard storage plan will be able to minimize the amount of congestion in a particular yard location, and hence reduce the amount of delays suffered by the AGVs. Furthermore, yard cranes or RTGCs are relatively much cheaper than QCs. Hence, yard cranes/RTGCs are assumed to be readily available when it is needed.

Assumption 4-7: There are N container jobs in the problem. The source and destination of them are given. Each job has an appointment time at its source/destination on the quay side. This appointment time is the time at which the job is to be unloaded/loaded from/on the vessel by a QC on the W/Ps. The appointment time, source and destination of jobs can be shown by a table like Table 4-2.

Table 4-2: Appointment time of containers jobs

Container Job (i)	Appointment time of Container Job i on the Quay side (t_i)	Source	Destination
1	00:30	W/P A	Block A
2	00:35	Block B	W/P B
3	00:40	W/P C	Block C
4	00:45	Block D	W/P A
..	..		
.	..		
N	..		

Assumption 4-8: There is a predetermined crane job sequence, consisting of loading jobs, or unloading/discharging jobs, or a combination of both for every QC. Given a specified job sequence, the corresponding drop-off (for loading) or pickup (for discharging) times of the jobs on the quayside depends on the work rate of the quay cranes. For example, assuming an average work rate of 5 minutes for one container (see Table 4-2), we need the horizontal transportation system to feed a container to the quay crane in every 5 minutes. This assumption for the cranes has the following two special properties that must be considered in developing any solution procedure:

- The container jobs must be carried out in the exact same order that is predetermined as in a sequence list. Planners in terminals make a discharging and loading sequence list before the ship operation begins. The sequence list is confirmed by the corresponding shipping company. Then, the ship operation is performed in the exact same order as specified in the sequence list.
- A delay in a quayside operation of a QC results in delays, by the same amount of time, to all succeeding seaside operations assigned to the same QC.

Assumption 4-9: The problem is divided into two types, static and dynamic. In the static problem, we assume that the number of vehicles, the number of jobs and the distance between every two points in the container terminal don't change. In the dynamic problem, we assume that the number of vehicles is fixed but the number of jobs, and the distance between the source and destination of the jobs may change (since the system controller may change the route of AGVs, due to congestion in the next lane or junction; see Figure 4-1). Note that in this problem each vehicle might be in different location of the port, on the quay side or in the yard side or in the middle of road between its source and destination.

Assumption 4-10: In this scheduling problem, our goal is to deploy the AGVs such that all the imposed appointment time constraints are met with minimum cost. Cheng et al. (2003) minimized waiting times of the AGVs [14]. Our objectives are to minimize (1) the total AGV waiting time on the quay side; (2) the total AGV travelling time in the route of port; (3) the total lateness times to serve the jobs. If our objectives are achieved by a deployment scheme for the AGVs, the terminal operates at the desired throughput rate.

4.3 Variables and notations

To make a model for the problem, the following variables and notations are used:

- a) t_i : Appointment time of job i at the quay side.

According to Assumption 4-7, the appointment time of the jobs are given. After the ship docked at the berth, the appointment time of the first jobs are calculated by the following expression:

$$t_i = \text{Ship_docked_time} + i \times W.$$

The *Ship_docked_time* is the time at which the ship is ready for discharge/loading at the berth. The time window W is the duration of discharging/loading a container. The appointment time of new jobs (after serving the first i jobs) is calculated by the following expression:

$$t_{i+k} = CT_i + k \times W$$

where CT_i denotes the actual completion time of the i -th job. Note that CT_i is available at the time of deployment of the job $(i+k)$.

b) RTA_m : Ready time of AGV m at the next location (either the quay-side or yard-side).

TTA_{mj} : Travel Time of AGV m from the next location to the location of job j on the quay side.

In the dynamic problems (see Assumption 4-9), the AGVs can be in different location and status. In reality, at any instant an AGV can be in one of the four states –*waiting* on the quay side, *Going* or *Idle* or *unloading/loading the job*. Each of these states, as the names suggest, corresponds to a different mode of operation for the AGV. The RTA_m for AGV m and calculation of its travelling time to the location of container job j , TTA_{mj} , is illustrated by Figure 4-3. As an example, consider the first case in the figure (case a). The RTA_m is the time for the AGV to get the yard and TTA_{mj} is the time distance from the yard to the destination of job j on the quay side. Thus, TTA_{mj} is the sum of the time needed for travelling from the next location of the vehicle to the source location of job j and from the source to the destination of the job. Other cases are calculated based on the next location of the vehicle and type of operation associated with job j .

c) DT_{ij} : the Distance Time between two distinct jobs i and j .

Given the source and destination of container jobs (see Table 4-2), the calculation of DT_{ij} is illustrated by Figure 4-4. It is calculated based on the type of operations associated with jobs i and j (unloading or loading). As an example, consider the first case in the figure (case a). In this case job i is unloaded from the ship and job j is loaded on to. In this case, DT_{ij} is the sum of distance from source location of job i to its destination, the distance between the destination of job i and source location of job j and the distance between the source and destination of job j . The other cases of the figure are calculated based on the type of operations associated with jobs i and j .

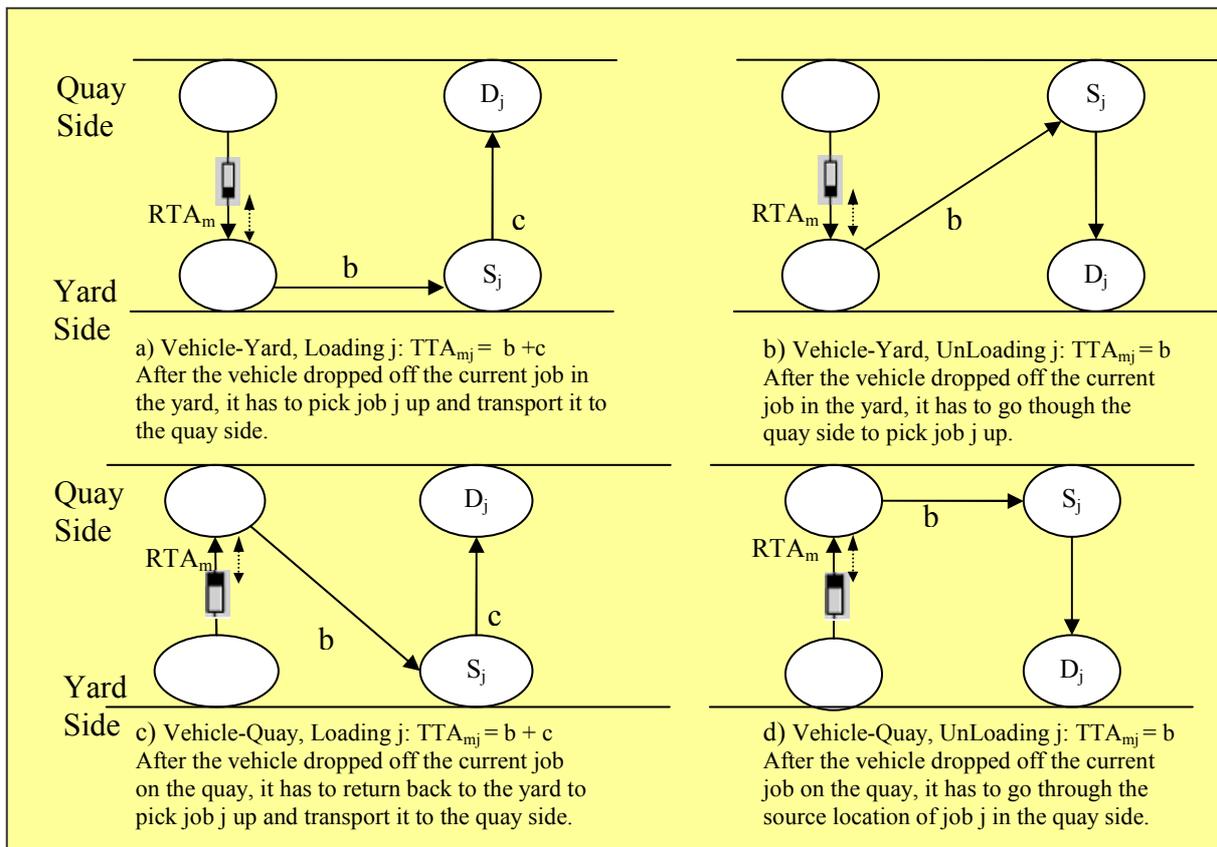


Figure 4-3: Travelling time computations between the next location of vehicle and the next job

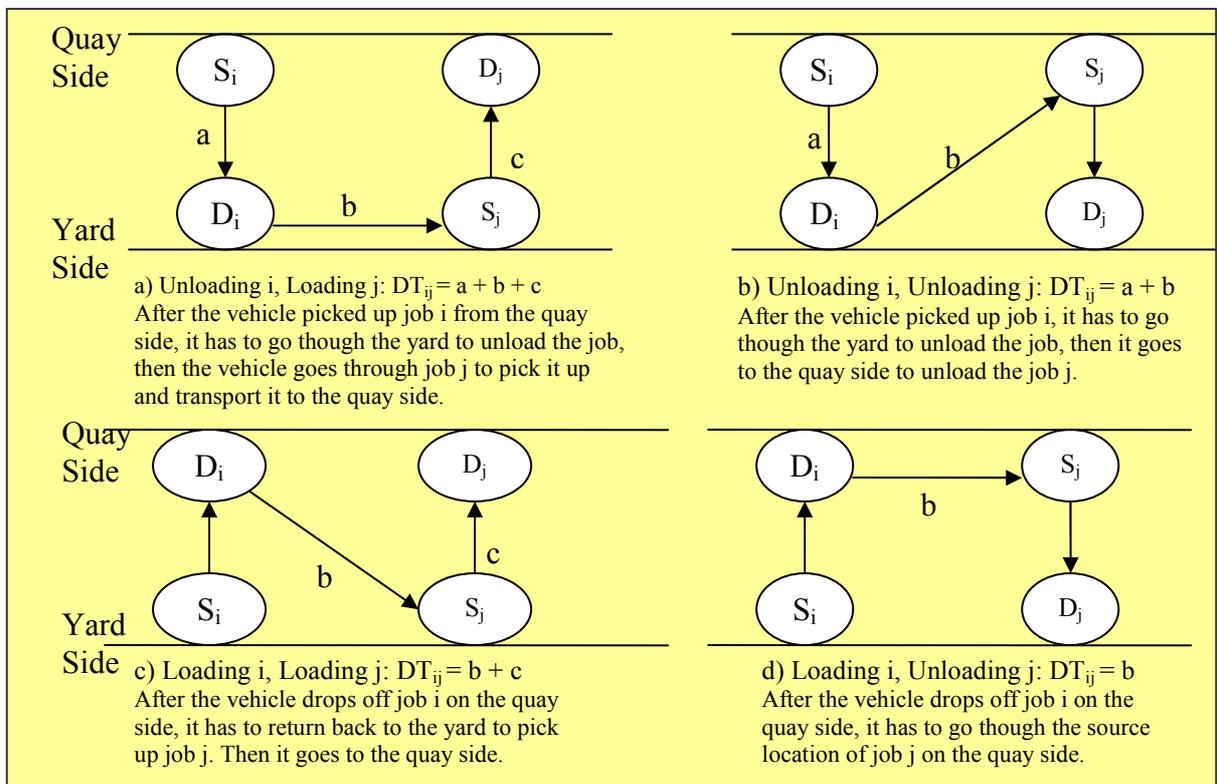


Figure 4-4: Travelling time computations between job i and job j

- d) w_1 : the weight of waiting time of the AGVs,
 w_2 : the weight of travelling time of the AGVs,
 P : the weight of the lateness time. P stands for Penalty of delay to serve the jobs.

According to assumption 4-10, these weights are required to be considered in the objective function.

4.4 The Minimum Cost Flow model

The scheduling problem of AGVs in the container terminal will be formulated as a Minimum Cost Flow (MCF) model [2]. In this section, we present the standard form of the MCF model with a few definitions, systematically. These definitions are related to Graph (G), the special Graph of G for the MCF model (G_{MCF}) and the MCF model itself.

4.4.1 Graph terminology

There are following standard definitions in graph theory (see Carre [11], Weber [101]).

Definition 4-1: A graph $G = (N, A)$ consists of a finite set of *nodes*, N , together with a finite set of *arcs*, A .

Definition 4-2: In an *undirected graph* the arcs are unordered pairs of nodes $\{i, j\} \in A, i, j \in N$.
 In a *directed graph* the arcs are ordered pairs of nodes (i, j) .

Definition 4-3: A *walk* is an ordered list of nodes i_1, i_2, \dots, i_t such that, in the case of an undirected graph, $\{i_k, i_{k+1}\} \in A$, or, in the case of a directed graph, that either $(i_k, i_{k+1}) \in A$ or $(i_{k+1}, i_k) \in A$, for $k = 1, \dots, t-1$.

Definition 4-4: A walk is a *path* if i_1, i_2, \dots, i_k are distinct, and a *cycle* if i_1, i_2, \dots, i_{k-1} are distinct and $i_1 = i_k$. A graph is connected if there is a path connecting every pair of nodes.

Definition 4-5: A loop in a directed graph is an arc which goes from a node to itself.

Definition 4-6: A network is a directed graph which is connected without loops.

Definition 4-7: A network is *acyclic* if it contains no cycles. A network is a tree if it is connected and acyclic. A network (N', A') is a sub-network of (N, A) if $N' \subset N$ and $A' \subset A$.

4.4.2 The standard form of the minimum cost flow model

The Minimum Cost Flow (MCF) model deals with a directed graph. In the graph, the problem is to send flow from a set of supply nodes, through a sub-network of the graph, to a set of demand nodes, at minimum total cost, and without violating the lower and upper bounds on flows through the arcs [2]. The MCF problem is defined as follows:

Definition 4-8 [2]: For the MCF problem, let graph $G = (N, A)$ be a directed network defined by a set of nodes, N , together with a set of arcs, A . Each arc $(i, j) \in A$ has an associated cost c_{ij} that denotes the cost per unit flow on that arc. It is assumed that the flow cost varies linearly with the amount of flow. The maximum and minimum amount of flow on each arc $(i, j) \in A$ are limited by M_{ij} and m_{ij} ($m_{ij} \leq M_{ij}$), respectively. A real number b_i is associated with each node, representing its supply/demand. If $b_i > 0$, node i is a supply node; if $b_i < 0$, the node i is a demand node with a demand of $-b_i$; and if $b_i = 0$, node i is a transshipment node. The decision variables in the MCF problem are arc flows, which is represented by f_{ij} for arc $(i, j) \in A$. The standard form of Minimum Cost Flow problem is as follows:

$$\begin{aligned} \text{MinCostFlow} &= \sum_{(i,j) \in A} c_{ij} \cdot f_{ij} \\ \text{Subject To} &\left\{ \begin{array}{l} \sum_{j:(i,j) \in A} f_{ij} - \sum_{j:(j,i) \in A} f_{ji} = b_i, \text{ for all } i \in N \\ m_{ij} \leq f_{ij} \leq M_{ij}, \text{ for all } (i, j) \in A \end{array} \right. \end{aligned}$$

These constraints state that flows must be feasible and conserve each node, i.e. the flow does not exceed the supply at a node and satisfies the demand. For the feasible flows to exist the MCF problem must also have $\sum_{i \in N} b_i = 0$, which means that the network is balanced. An important special case is that of incappeditated flows, $m_{ij} = 0$ and $M_{ij} = \infty$.

We now define a special graph for the MCF problem as follows:

Definition 4-9: A graph $G_{\text{MCF}} = (G, \text{NP}, \text{AP})$ consists of a graph G with a couple of properties for the nodes and arcs in G . The NP and AP are the Node's and Arc's Properties, respectively. The node property function $\text{NP}: N \rightarrow \mathbf{R}$ (Real numbers; possibly negative) gives the amount of supply/demand of the nodes. This function for each node is defined as follows:

$$\text{NP}(i) = \text{NP}_i = b_i \text{ where } \left\{ \begin{array}{l} b_i > 0 \text{ if node } i \text{ is a supply node} \\ b_i < 0 \text{ if node } i \text{ is a demand node} \\ b_i = 0 \text{ if node } i \text{ is a transshipment node} \end{array} \right\} \text{ so that } \sum_{i \in N} \text{NP}(i) = 0$$

The arc property function $AP: A \rightarrow \mathbb{R} \times \mathbb{R} \times \mathbb{R}$ (Real numbers; nonnegative) gives the lower bound, the upper bound and the cost of the arcs. This function for each element in A is defined as follows:

$$AP(i,j)=AP_{ij} = [m_{ij}, M_{ij}, c_{ij}]$$

Based on Definitions 4-8 and 4-9, we define the standard Minimum Cost Flow (MCF) problem, formally as follows:

Definition 4-10: a MCF model is defined as:

$$MCF = (G_{MCF}, f, D, CS, FC)$$

where $G_{MCF} = ((N,A), NP, AP)$ is a special graph for the MCF problem;

f = a finite set of decision variables on A (f stands for flow), $f = \{f_{ij} \mid (i, j) \in A\}$;

D = a function which determines a lower and upper bound for f ;

$D: f \rightarrow \mathbb{R} \times \mathbb{R}$ (to be pulled out from AP); We shall take $D_{f_{ij}}$ as the lower bound and

upper bound of f_{ij} by D (D stands for Domain);

CS = a finite set of Constraints on NP and f ;

FC = an objective function for the Flow's Cost on AP and f ;

The task in a MCF model is to assign a value to each f_{ij} that satisfy all constraints in CS with regard to the minimum value for FC .

For the standard form of the MCF model we have:

a) For each element D and f , $D_{f_{ij}} = [m_{ij}, M_{ij}]$, for $\forall (i, j) \in A$;

b) The CS is $\sum_{j:(i,j) \in A} f_{ij} - \sum_{j:(j,i) \in A} f_{ji} = NP_i$, for $\forall i \in N$

c) The FC is $\sum_{j:(j,i) \in A} c_{ij} \cdot f_{ij}$

4.5 The special case of the MCF model for Automated Guided Vehicles Scheduling

Here, we present a special case of the MCF model for the Scheduling problem of Automated Guided Vehicles (SAGV) in the container terminal. The problem differs primarily in the arrangement of nodes and arcs with their properties. In this special case, the property function of

nodes assigns integer value to every node. Additionally, the property function of arcs assigns integer values to the lower bound, the upper bound and the cost of each arc. Moreover, the lower bound and the upper bound of each arc take the binary values, 0 or 1. We present the special Graph of G_{MCF} for the Automated Guided Vehicles Scheduling ($G_{MCF-AGV}$) and the special case of the MCF model for the Scheduling problem of Automated Guided Vehicles (MCF-AGV).

Based on Definition 4-9, we introduce the following definition for the G_{MCF} in a special case:

Definition 4-11: A graph $G_{MCF-AGV} = (GS, NPS, APS)$ is a special case of $G_{MCF} = (G, NP, AP)$. The graph $GS = (NS, AS)$ is a Special case of $G = (N, A)$; the node and arcs properties of GS , NPS and APS , are also special cases of NP and AP , respectively ($NPS: NS \rightarrow N$ and $APS: AS \rightarrow N \times N \times N$; N is the set of Natural numbers). In this section, we formally describe the elements of $G_{MCF-AGV}$ in the two following sub-sections:

4.5.1 Nodes and their properties in the special graph

As we mentioned, there are three types of nodes in the standard form of a MCF model: supply nodes, transshipment nodes, and demand nodes [2]. Here, our problem is formalized with four different types of nodes: a supply node for each AGV, a couple of nodes for each container job as transshipment nodes (the reason is in the next section, see the Auxiliary Arcs) and a demand node. Given N jobs and M AGVs in the problem, the elements in each set, the sets themselves and the nodes properties are defined as follows:

a) $AGVN_m$: a supply node corresponding to AGV m with one unit supply ($AGVN$ stands for the AGV Node). There are M AGVs in the problem. Hence, there are M supply nodes in the $G_{MCF-AGV}$. We define the following set for these supply nodes along with their properties:

SAGVN: a set of M supply nodes in the $G_{MCF-AGV}$.

$$SAGVN = \{AGVN_m \mid m=1,2,\dots,M; NPS(m)=1\}$$

b) JIN_i : a node through which an AGV enters job i . It stands for the Job-Input Node. There is neither supply nor demand in this node, i.e. it is a transshipment node. We define the following set for these transshipment nodes along with their properties:

SJIN: a set of N Job-Input nodes in the $G_{MCF-AGV}$.

$$SJIN = \{JIN_i \mid i=1,2,\dots,N; NPS(i)=0\}$$

c) **JOUT_i**: a node from which an AGV leaves job i. It stands for the Job-Output Node. Like the previous nodes, there is neither supply nor demand in this node. We define the following set for these transshipment nodes along with their properties:

SJOUT: a set of N Job-Output nodes in the $G_{MCF-AGV}$

$$SJOUT = \{JOUT_i \mid i=1,2,\dots,N; NPS(i)=0\}$$

d) **SINK**: It stands for a Sink node or a demand node in the $G_{MCF-AGV}$ with M units demand. This node corresponds to the end state of the process, after all container jobs have been served. Hence, for the property of this node, we have:

$$NPS(SINK) = -M.$$

Therefore, there are $M+2 \times N+1$ nodes in the $G_{MCF-AGV}$ so that:

$$NS = SAGVN \cup SJIN \cup SJOUT \cup SINK$$

4.5.2 Arcs and their properties in the special graph

The following four types of arcs with their properties connect the nodes in the $G_{MCF-AGV}$:

1) **Inward Arcs**: There is a directed arc from every AGV node, to the Job-Input node of job i. We define the following notation for these arcs along with their properties:

ARC_{inward} : a set of arcs from SAGVN to SJIN.

$$ARC_{inward} = \{ (m, j) \mid \forall m \in SAGVN, \forall j \in SJIN, APS(m, j) = [0, 1, C_{mj}] \}$$

The number of these arcs in the $G_{MCF-AGV}$ is $M \times N$. Each arcs has the lower bound zero, and the upper bound one, i.e., only one AGV goes through each of these arcs. As we mentioned before (see Assumption 4-10), our objectives are to minimize waiting and travelling times of the AGVs and the lateness times of jobs. The cost between node m and node j is calculated as follows:

$$C_{mj} = \begin{cases} w_1 \times (t_j - (RTA_m + TTA_{mj})) + w_2 \times (RTA_m + TTA_{mj}) & \text{if } (t_j \geq RTA_m + TTA_{mj}) \\ P \times (RTA_m + TTA_{mj} - t_j) & \text{otherwise} \end{cases}$$

If AGV m could arrive on the quay side before the appointment time of the job associated with node j ($t_j \geq RTA_m + TTA_{mj}$), there is no lateness time to serve the job. Therefore the waiting and travelling times of AGV m to serve the job associated with node j are calculated as the cost. Otherwise, the lateness time to serving node j with a penalty (P) is considered. Note that there is neither waiting nor travelling time for the AGV in the second case.

- 2) **Intermediate Arcs:** There is a directed arc from every Job-Output node i to other Job-Input node j . We define the following notation for these arcs along with their properties:

$ARC_{intermediate}$: a set of arcs from SJOUT to SJIN.

$$ARC_{intermediate} = \{ (i, j) \mid \forall i \in SJOUT, \forall j \in SJIN, j \neq JIN_i, APS(m, j) = [0, 1, C_{ij}] \}$$

The number of these arcs in the $G_{MCF-AGV}$ is $N \times (N-1)$. Each arcs has the lower bound zero, and the upper bound one, i.e., only one AGV goes through from one job to another. The cost between node i and node j in the $G_{MCF-AGV}$ is calculated as follows:

$$C_{ij} = \begin{cases} w_1 \times (t_j - (t_i + DT_{ij})) + w_2 \times DT_{ij} & \text{if } (t_j \geq t_i + DT_{ij}) \\ P \times (t_i + DT_{ij} - t_j) & \text{Otherwise} \end{cases}$$

The first case shows that an AGV can serve the job associated with node j after serving the job associated with node i ($t_j \geq t_i + DT_{ij}$). In this case waiting and travelling times of the AGV are calculated without any lateness time. In the second case, there is neither waiting nor travelling time for the AGV and only the lateness time of serving node j with a penalty (P) is considered for the cost.

- 3) **Outward Arcs:** There is a directed arc from every Job-Output node i and AGV node m to SINK. We define the following notation for these arcs along with their properties:

$ARC_{outward}$: a set of arcs from SJOUT and SJAGVN to SINK.

$$ARC_{outward} = \{ (i, j) \mid \forall i \in SAGVN \cup SJOUT, j = SINK; APS(m, j) = [0, 1, 0] \}$$

These arcs show that an AGV can remain idle after serving any number of jobs or without serving any job. Therefore, a cost of zero is assigned to these arcs.

- 4) **Auxiliary Arcs:** There is a directed arc from every Job-Input node i to its Job-Output node.

We define the following notation for these arcs along with their properties:

$ARC_{auxiliary}$: a set of arcs from SJIN to SJOUT.

$$ARC_{auxiliary} = \{ (i, j) \mid \forall i \in SJIN, j = \text{an unique Job-Output node in SJOUT, correspond to the Input-Node } i; APS(i, j) = [1, 1, 0] \}$$

These arcs have unit lower and upper bounds. The transition cost across these arcs is zero.

These auxiliary arcs guarantee that every Job-Input and Job-Output node is visited once only so that each job is served.

Therefore, there are $M \times N + N \times (N-1) + M + 2 \times N$ arcs in the $G_{MCF-AGV}$ so that:

$$AS = ARC_{inward} \cup ARC_{intermediate} \cup ARC_{outward} \cup ARC_{auxiliary}$$

4.5.3 The MCF-AGV model for the Automated Guided Vehicles Scheduling

Now we present the special case of the MCF model for the Automated Guided Vehicles Scheduling with the following definition.

Definition 4-12: A MCF-AGV model is a special case of the MCF (Definition 4-10) for the Scheduling problem of Automated Guided Vehicles in the container terminals. A MCF-AGV model is defined as

$$\text{MCF-AGV} = (G_{\text{MCF-AGV}}, f, D, \text{CS}, \text{FC})$$

Where $G_{\text{MCF-AGV}} = (\text{GS}, \text{NPS}, \text{APS})$ is a graph for the MCF-AGV problem;

f = a finite set of integer decision variables on AS, $f = \{f_{ij} \mid (i, j) \in \text{AS}\}$;

D = a function which determines a lower and upper bound for f ; $D: f \rightarrow N \times N$ (to be pulled out from APS); For each element in D , corresponding to the type of arcs:

$$1) D_{f_{ij}} = [0,1] \text{ for } (i, j) \in \text{ARC}_{\text{inward}} \cup \text{ARC}_{\text{intermediate}} \cup \text{ARC}_{\text{outward}}$$

$$2) D_{f_{ij}} = [1,1] \text{ for } (i, j) \in \text{ARC}_{\text{auxiliary}}$$

CS = The constraints of the MCF-AGV are:

$$\left. \begin{array}{l} 1) \sum_{j:(i,j) \in \text{AS}} f_{ij} = 1; \quad \forall i \in \text{SAGVN} \\ 2) \sum_{j:(j,i) \in \text{AS}} f_{ji} = M; \quad \text{for } i = \text{SINK} \\ 3) \sum_{j:(i,j) \in \text{AS}} f_{ij} - \sum_{j:(j,i) \in \text{AS}} f_{ji} = 0; \quad \forall i \in \{\text{SJIN} \cup \text{SJOUT}\} \end{array} \right\}$$

The first constraint shows every node i ($i \in \text{SAGVN}$) sends one unit flow into the network. The second constraint ensures SINK node receives M units flow (the flows sent from nodes in SAGVN set). The third constraint shows the flow balance at every Job-Input and Job-Output node.

$$\text{FC} = \sum_{(i,j) \in \text{AS}} C_{ij} \cdot f_{ij}$$

Solving the MCF-AGV model generates M paths, each of which commences from a node in SAGVN and terminates at SINK. Each path determines a job sequence for every AGV. The decision variable f_{ij} for every arc $(i,j) \in \text{AS}$ (the flow between nodes i and j in the $G_{\text{MCF-AGV}}$) is either 1 or 0. $f_{ij} = 1$ means that an AGV goes from node i to node j . Otherwise, moving the AGV from node i to node j is not possible.

The MCF-AGV model of the problem can be illustrated by Figure 4-5 for two AGVs and four container jobs. According to our definitions for nodes and arcs in the MCF-AGV model, we have the following sets with their properties:

- $SAGVN = \{1,2\}$; $NPS(1)=NPS(2)=1$
- $SJIN = \{3,5,7,9\}$; $NPS(j)=0, \forall j \in SJIN$
- $SJOUT = \{4,6,8,10\}$; $NPS(j)=0, \forall j \in SJOUT$
- $SINK = \{11\}$; $NPS(11)=-2$
- $ARC_{inward} = \{(1,3),(1,5),(1,7),(1,9), (2,3),(2,5),(2,7),(2,9)\}$; $APS(m,j)=[0,1,C_{m,j}]$, $\forall m \in SAGVN, \forall j \in SJIN$
- $ARC_{intermediate} = \{(4,5), (4,7), (4,9), (6,3),(6,7),(6,9), (8,3),(8,5),(8,9),(10,3),(10,5),(10,7)\}$; $APS(i,j)=[0,1,C_{i,j}]$
 $\forall i \in SJOUT, \forall j \in SJIN$
- $ARC_{outward} = \{(1,11),(2,11),(4,11),(6,11),(8,11),(10,11)\}$; $APS(i,j)=[0,1,0]$, $\forall i \in SJOUT, \forall j \in SJIN$
- $ARC_{auxiliary} = \{(3,4),(5,6),(7,8),(9,10)\}$; $APS(i,j)=[1,1,0]$; $\forall i \in SJOUT, \forall j \in SJIN$

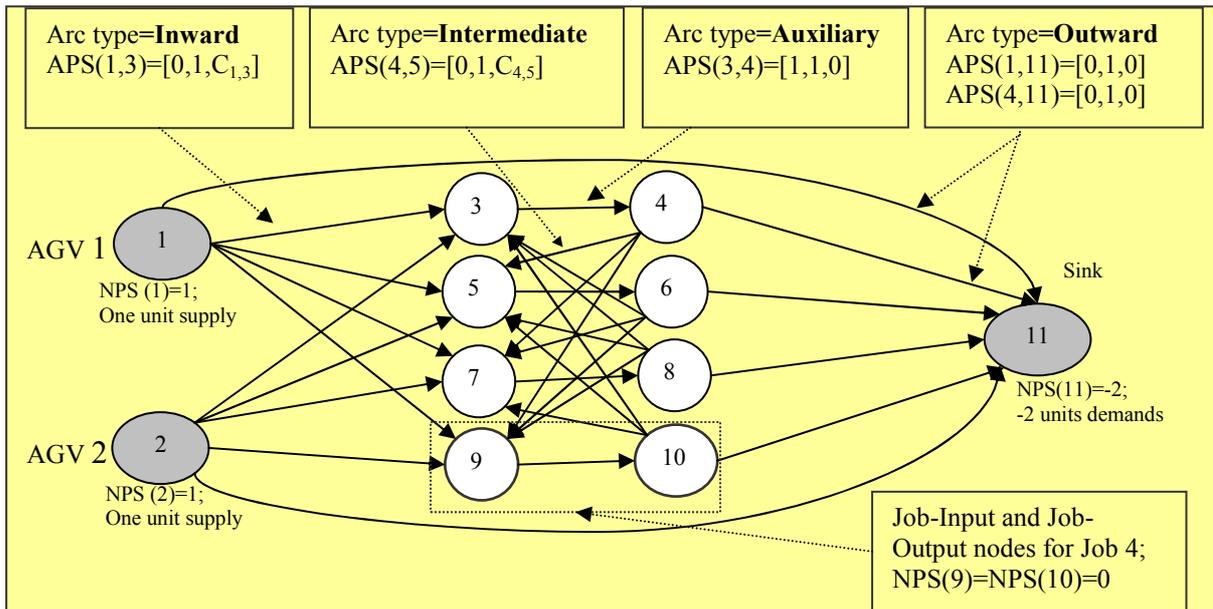


Figure 4-5: An example of the MCF-AGV model for 2 AGVs and 4 jobs

We showed one example for each type of the arcs with their properties (the lower bound, upper bound and cost, respectively) in the figure. Suppose that for some values of arc costs, the solution paths are $1 \rightarrow 3 \rightarrow 4 \rightarrow 9 \rightarrow 10 \rightarrow 11$ and $2 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 11$. This states that AGV 1 is assigned to serve container jobs 1 and 4, and AGV 2 is assigned to serve container jobs 2 and 3, respectively.

The MCF-AGV model has a huge search space and the solution should provide the optimal paths for each AGV from every node in SAGVN to SINK. As we mentioned before, there are $M+2 \times N+1$ nodes and $M+M \times N+N \times (N-1)+2 \times N$ arcs in the graph model where N and M specify the number of jobs and the number of AGVs in the problem, respectively. The number of paths in the search space is determined by the following equation:

$$\text{NumberOfPaths} = M + \binom{M}{1} \times N! \times 1 + \binom{M}{2} \times (N-1)! \times 2 + \dots + \binom{M}{M} \times (N-M+1)! \times M$$

$$\text{where } \binom{P}{Q} = \frac{P!}{Q! \times (P-Q)!}$$

The equation calculates every possible path in the search space. The first term represents paths from every node in SAGVN to SINK. The remaining terms shows the number of paths when 1, 2, .. , M ($M \leq N$) AGVs, respectively, is selected to serve the jobs.

4.6 Summary and conclusion

In this chapter, a scheduling problem in the container terminal was presented and formulated. The problem was to carry many container jobs from quay-side to yard-side or vice versa by several Automated Guided Vehicles. Each job has an appointment time on the quay-side and the jobs should be served in their appointment time by the AGVs.

The formulation was based on the Minimum Cost Flow (MCF) model. We introduced the $G_{MCF} = (G, NP, AP)$, a graph G with a couple of functions for the Node's Properties (NP) and the Arc's Property (AP) for the MCF model. After that, we presented a formal definition for the MCF model; $MCF = (G_{MCF}, f, D, CS, FC)$ where f , D , CS and FC were the decision variables, domain of f , constraints and objective function, respectively.

We established the scheduling problem of Automated Guided Vehicles on the MCF model. In order to do that, we defined a graph, $G_{MCF-AGV}$, for the problem. Then, we introduced the MCF-AGV model for the scheduling problem, as a special case of the MCF model. The decision variables with value one identified the path for the AGVs inside the graph $G_{MCF-AGV}$. There are always feasible and optimal solutions since the formulation is based on the standard form of the MCF model.

Chapter 5: Network Simplex Algorithm and Static Scheduling of AGVs

In Chapter 4, the scheduling problem of Automated Guided Vehicles (AGVs) in the container terminals was formulated as a special case of Minimum Cost Flow model. The model was introduced as the MCF-AGV. This chapter focuses on the standard Network Simplex Algorithm (NSA) to tackle the MCF-AGV in static aspect. In this aspect the number of jobs, the distance between the source and destination of the jobs, and the number of vehicles don't change (see Assumption 4-9).

5.1 Reasons to choose NSA

The main reasons to choose NSA are as follows:

- The Minimum Cost Flow (MCF) model has a rich history. This problem arises in almost all industries, including agriculture, communications, defence, education, energy, health care, manufacturing, medicine, retailing, and transportation [2]. NSA is a solution for the MCF model.
- The area of development algorithm to tackle the MCF model by NSA is under-researched and offers fertile research opportunities for large scale problems. Several researches have been devoted on this matter [1, 3, 24, 36, 46, 58, 67, 70] in the recent years.
- NSA is based on simple network operations. With simple network operations, the MCF model can be solved more than 100 times faster than equivalently sized Linear Programs¹. It is the fastest algorithm for solving the generalized network flow problem in practice [2].

5.2 The Network Simplex Algorithm

In Network Simplex Algorithm, the linear algebra of original simplex algorithm (in Operation Research) is replaced by simple network operations. Ahuja, Magnanti, and Orlin (1993) described the network simplex algorithm and gave pseudo-codes, implementation and hints [2].

¹ <http://mat.gsia.cmu.edu/classes/networks/node8.html> (Last check of the address: 3 Sep 2005)

Here, the standard form of Network Simplex Algorithm is presented. More details and several other algorithms for the MCF problem can be seen in the text book [2].

5.2.1 Spanning tree solutions and optimality conditions

Given graph $G_{MCF} = (N, A, NP, AP)$ for the MCF problem (see Definition 4-8), the standard form of Minimum Cost Flow problem [2] was as follows:

$$MinCostFlow = \sum_{(i,j) \in A} c_{ij} \cdot f_{ij}$$

$$Subject\ To \begin{cases} \sum_{j:(i,j) \in A} f_{ij} - \sum_{j:(j,i) \in A} f_{ji} = b_i, \text{ for all } i \in N \\ m_{ij} \leq f_{ij} \leq M_{ij}, \text{ for all } (i,j) \in A \end{cases}$$

In network simplex algorithm, it is assumed that the network is connected. Every connected graph has a spanning tree [2]. Some preliminary definitions related to the spanning tree are:

Definition 5-1: A spanning tree solution for the MCF problem is divided into three sets of arcs (T, L, U) of the graph. Given n as the number of nodes in the graph, $T \subset A$ is a set consist of $n-1$ arcs. The remaining arcs are divided into the two sets L and U . For these two sets, $f_{ij} = m_{ij}$ for each arc $(i, j) \in L$ and $f_{ij} = M_{ij}$ for each arc $(i, j) \in U$.

Definition 5-2: A spanning tree solution with $m_{ij} \leq f_{ij} \leq M_{ij}$ is a feasible spanning tree solution. In Figure 5-1, the spanning tree is a feasible spanning tree solution provided that for each dotted arc $m_{ij} \leq f_{ij} \leq M_{ij}$.

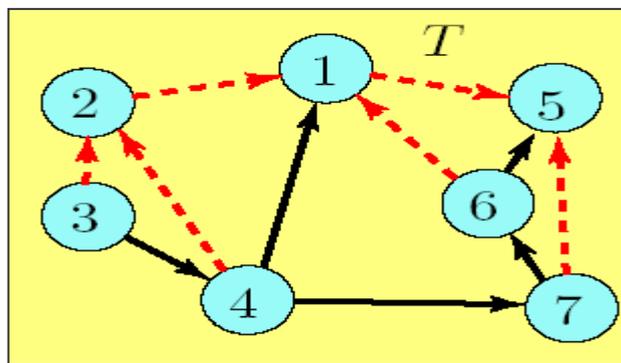


Figure 5-1: A feasible spanning tree solution (dotted)

Before stating the optimality condition of Network Simplex Algorithm, a couple of theorems and a property for the algorithm are presented.

Theorem 5-1 [2]: The Minimum Cost Flow problem is a special form of the Linear Program (in Operation Research). Given n nodes and k arcs in the graph model, the MCF problem can be represented as follows:

$$\begin{aligned} \text{MinCostFlow} &= c_{1 \times k} f_{k \times 1} \\ \text{Subject to} &\begin{cases} H_{n \times k} f_{k \times 1} = b_{n \times 1} \\ m_{k \times 1} \leq f_{k \times 1} \leq M_{k \times 1} \end{cases} \end{aligned}$$

In this formulation, the matrixes of b , c , f , m , M are the same as the MCF problem (see Section 4.4.2). The coefficient matrix, H , is called the node-arc incidence matrix. The elements of this matrix are defined as follows:

$$H_{ij} = \begin{cases} +1 & \text{If node } i \text{ is the start of } j^{\text{th}} \text{ arc} \\ -1 & \text{If node } i \text{ is the end of } j^{\text{th}} \text{ arc} \\ 0 & \text{otherwise} \end{cases}$$

Theorem 5-2 [2]: A flow vector of a basic solution for the Linear Program is a spanning tree solution of the MCF problem. Flows on non-basic arcs are either m_{ij} or M_{ij} .

Property 5-1 [2]: Suppose that a number $\pi(i)$ is associated with each node $i \in N$, which is referred to as the potential of that node. With respect to the node potentials $\pi = (\pi(1), \pi(2), \dots, \pi(n))$, the reduced cost \bar{C}_{ij} of an arc (i, j) is defined as follows:

$$\bar{C}_{ij} = C_{ij} - \pi(i) + \pi(j)$$

Theorem 5-3 (Necessary optimality conditions): The Optimality Conditions of the spanning tree solution (T, L, U) is obtained by the Lagrangian of the Minimum Cost Flow (MCF) problem. The Lagrangian of the minimum cost flow problem is:

$$\begin{aligned} L(f, \pi) &= \sum_{(i,j) \in A} c_{ij} \cdot f_{ij} - \sum_{i \in N} \pi_i \cdot \left(\sum_{j:(i,j) \in A} f_{ij} - \sum_{j:(j,i) \in A} f_{ji} - b_i \right) \\ &= \sum_{(i,j) \in A} (c_{ij} - \pi_i + \pi_j) f_{ij} + \sum_{i \in N} \pi_i \cdot b_i \end{aligned}$$

Minimizing $L(f, \pi)$ over $m_{ij} \leq f_{ij} \leq M_{ij}$ gives dual feasibility and complementary slackness conditions [101]. If the reduced cost is zero, f_{ij} could have any values between m_{ij} and M_{ij} .

Otherwise, the f_{ij} has the maximum (minimum) value when the reduced cost is negative (positive). Hence, the optimality conditions are:

$$\bar{C}_{ij} = C_{ij} - \pi(i) + \pi(j) > 0 \Rightarrow f_{ij} = m_{ij}; (i, j) \in L$$

$$\bar{C}_{ij} = C_{ij} - \pi(i) + \pi(j) < 0 \Rightarrow f_{ij} = M_{ij}; (i, j) \in U$$

$$\bar{C}_{ij} = C_{ij} - \pi(i) + \pi(j) = 0 \Rightarrow m_{ij} \leq f_{ij} \leq M_{ij}; (i, j) \in T$$

Given n nodes in the network, the spanning tree (T) has $n-1$ arcs. The potential of each node is calculated by the last equation ($\pi(i)-\pi(j)=C_{ij}$). The potential of one node is set arbitrarily. Usually it is the root of the tree with value 0 for its potential [2].

In NSA, it is worked with the reduced cost, instead of the actual cost [2]. It is important to determine the relationship between the objective functions $z(\pi) = \sum_{(i,j) \in A} \bar{C}_{ij} \cdot f_{ij}$ and $z(0) = \sum_{(i,j) \in A} C_{ij} \cdot f_{ij}$.

Suppose, initially, that $\pi=0$ and then we increase the potential of node k to $\pi(k)$. The definition of reduced costs implies that this change reduces the reduced cost of each unit of flow leaving node k by $\pi(k)$ and increases the reduced cost of each flow unit entering node k by $\pi(k)$. Thus the total decrease in the objective function equals $\pi(k)$ times the outflow of node k minus the inflow of node k . By the constraint for each node, the outflow minus inflow equals the supply/demand of the node. Consequently, increasing the potential of node k by $\pi(k)$ decreases the objective function value by $\pi(k) \times b(k)$ units. Repeating this argument iteratively for each node establishes that:

$$z(0) - z(\pi) = \sum_{i \in N} \pi(i) b(i) = \pi b$$

Given node potential π , $\pi.b$ is a constant. Therefore, a flow that minimizes $z(\pi)$ also minimizes $z(0)$. This result is used in Theorem 5-4.

Theorem 5-4 (Sufficient Optimality Conditions) [2]: Let f^* be the solution associated with the spanning tree structure (T, L, U). Suppose that some set of node potential π , together with the spanning tree structure (T, L, U) satisfy the optimality conditions.

It is needed to show that f^* is an optimal solution of the minimum cost flow problem. Previously, it was showed that minimizing $z(\pi) = \sum_{(i,j) \in A} \bar{C}_{ij} \cdot f_{ij}$ is equivalent to minimize $z(0) = \sum_{(i,j) \in A} C_{ij} \cdot f_{ij}$. The

optimality conditions stated as above imply that for the given node potential π , $z(\pi) = \sum_{(i,j) \in A} \bar{c}_{ij} \cdot f_{ij}$ is equivalent to minimizing the following expression:

$$\text{Minimize } \sum_{(i,j) \in L} \bar{c}_{ij} \cdot f_{ij} - \sum_{(i,j) \in U} |c_{ij}^+| \cdot f_{ij}$$

The definition of the solution f^* implies that for any arbitrary solution f , $f_{ij} \geq f_{ij}^*$ for all $(i,j) \in L$ and $f_{ij} \leq f_{ij}^*$ for all $(i,j) \in U$. The above expression implies that the objective function value of the solution f will be greater than or equal to that of f^* . \square

In economic aspect, the following interpretations can be stated [2]:

- \bar{c}_{ij} is the amount of change in the objective function, if there is one unit change in f_{ij} .
- π_i is the cost of sending one unit of flow from node i to the root along the tree path.
- $c_{ij} - \pi_i$ is the cost of obtaining one unit of the commodity at node i and then shipping it to node j .

5.2.2 The algorithm NSA

The network simplex algorithm maintains a feasible spanning tree structure at each iteration and successfully transforms it into an improved spanning tree structure until it becomes optimal. The algorithm in Figure 5-2 specifies steps of this method [2, 48].

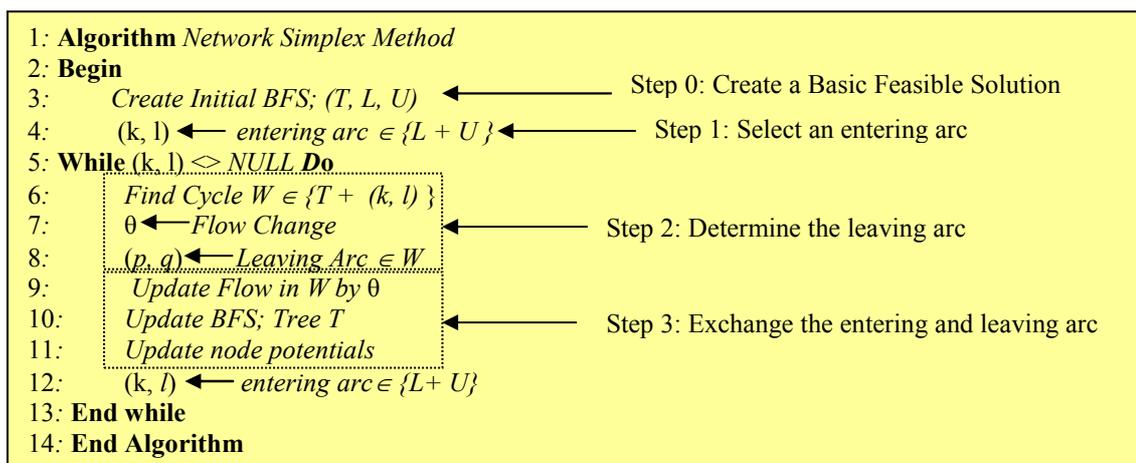


Figure 5-2: The Network Simplex Algorithm

Figure 5-2 shows four main steps in the algorithm:

- **Step 0:** Initial or create a Basic Feasible Solution (BFS).
- **Step 1:** Select an entering arc (which is appended to the spanning tree).
- **Step 2:** Determine the leaving Arc (which must be removed from the spanning tree).
- **Step 3:** Pivoting (exchange the entering and leaving Arc).

Step 0: To create an initial or Basic Feasible Solution, the graph has to be connected, which correspond to the MCF-AGV model in Chapter 4. In Line 3, creating an initial feasible spanning tree solution (see Definition 5-2) for every connected graph can be made by an easy way [2]. It is obtained by adding an artificial root node '0' to N and the artificial slack arcs $(i,0)$ and $(0,i)$, respectively, to A . Each artificial slack arc has the lower bound of zero, the upper bound of infinity, and a sufficiently large cost coefficient. The initial basic tree is consisting of all artificial arcs, each original arc becomes non-basic at its lower bound and no arc becomes non-basic at the upper bound. We examine each node j , other than '0', one by one. If $b(j) \geq 0$, we include $(j, 0)$ in T with a flow value of $b(j)$. If $b(j) < 0$, we include arc $(0, j)$ in T with a flow value of $-b(j)$. The set L consist of the remaining arcs, and the set U is empty.

Step 1: At each iteration of the algorithm, an entering arc is selected by some pricing scheme [48 46, 48]. This arc is selected from the non-basic arcs $(L + U)$. There are several schemes for selecting the entering arc, and these determine the speed of algorithm. A literature review over these schemes is presented later in this chapter. An arc may be admitted to the basis to improve the objective function if it violates the optimality conditions. Thus an arc $(i,j) \in A$, with the following conditions are admissible:

$$\begin{aligned} & \text{If } \bar{C}_{ij} < 0 \text{ and } f_{ij} = m_{ij} \\ & \text{or } \bar{C}_{ij} > 0 \text{ and } f_{ij} = M_{ij} \end{aligned}$$

If no admissible arc exists, then the current solution is optimal, and the algorithm terminates. Otherwise, Step 2 is performed.

Step 2: Appending the entering arc, (k, l) , to the spanning tree forms a unique cycle, W , with the arcs of the basis. In Line 6, the algorithm finds out the cycle. In order to eliminate this cycle in the tree, one of its arcs must leave the basis. By augmenting flow in a negative cost augmenting cycle, the objective value of the solution can be improved. The cycle is eliminated when there is

an augmented flow by a sufficient amount to force the flow in one or more arcs of the cycle to their upper or lower bounds. In Line 7, the flow change is determined by the following equation:

$$\theta = \text{Min } \{ \Delta f_{ij} \text{ for all } (i, j) \in W \} .$$

The leaving arc is selected based on cycle W and θ , in Line 8.

Step 3: In this step, the entering arc and the leaving arc are exchanged, and the new BFS is constructed. The construction of a new basis tree is called the pivot; adjusting flows, making the new spanning tree and updating the node potentials accordingly in the spanning tree solution (T, L, U) . These operations are performed in Lines 9, 10 and 11, respectively. We refer to cycle W (see Step 2) as the basis cycle. The algorithm sends a maximum possible amount of flow in the basis cycle without violating any of the lower and upper bound constraints on arcs. An arc which blocks further increase of flow in the basis cycle is called a blocking arc. The flow in every arc of the cycle W is increased or decreased by the amount θ depending on the orientation of the arc in relation to the orientation of the cycle. Generally, a basic arc is exchanged with a non-basic arc. The algorithm drops a blocking arc, say (p, q) , from T . This gives a new basis structure. Let $T1$ and $T2$ be the two sub-trees formed by deleting arc (p, q) from the previous basis T where $T1$ contains the root. In the new basis, potentials of all nodes in $T1$ remain unchanged and potentials of all nodes in $T2$ change by a constant amount. If (k, l) was a non-basic arc at its lower bound in the previous basis structure, then the amount of change is an increase by $\left| \bar{c}_{kl} \right|$, else it is decrease by an amount \bar{c}_{kl} .

5.2.3 The difference between NSA and original simplex

Those steps in Network Simplex Algorithm (NSA) can be compared with the Original Simplex Algorithm (OSA) (to solve Linear Program in Operation Research). Note that the main difference is that NSA is based on graph and operations in the graph while the OSA needs matrix and matrix manipulations. Step 0 is taken to finding an initial solution in both algorithms. An initial basic spanning tree is created by adding the artificial node and arcs in NSA. In the similar way, OSA uses the artificial variables to generate an initial basic solution. Steps 1 and 2 in the both algorithms are choosing the entering and leaving arc in NSA, which are similar to choosing the entering and leaving variable in OSA. Constructing a new spanning tree in NSA and new basic

solution in OSA are Step 3 for the both algorithms. In this way, OSA needs some matrix manipulation and inversion, whereas a new spanning tree can be easily constructed by some operation in the graph without any multiplication and division. Both algorithms continue Steps 1-3 until they meet the optimality conditions. Obviously, the matrix manipulations are different from graph operations, which have significant negative impacts on the performance of OSA.

5.2.4 A short literature over pricing rules

In order to find out an entering arc for the basic solution, there are different rules, which called pricing schemes. The performance of the Network Simplex Algorithm is affected by these schemes. A literature review over these schemes is given below:

The standard textbook [2] provided a detailed account of the literature on those schemes. We now briefly review this literature. Bradley, Brown and Graves (1977), used a dynamic queue, containing the indices of so-called ‘interesting’ nodes and admissible arcs. Their method is called *BBG Queue pricing scheme*. An ‘interesting’ node is a node whose incident arcs have not been re-priced in recent iterations. At each iteration, the entering arc is selected from the queue. Another candidate list scheme has been described by Mulvey (1978). In the *Mulvey scheme*, there is a major and minor loop to select the entering arc. A limited number of favourably priced entering arcs are collected by scanning the non-basic arcs in a major iteration. In the minor iteration, the most favourably priced arc in the list is chosen to enter the basis. Grigoriadis (1986) describes a very simple arc *block pricing scheme* based on dividing the arcs into a number of subsets of specified size. At each iteration, the entering arc is selected from a block with most negative price. Only the arcs of one block are re-priced at any iteration. Taha (1987) suggested *the most negative pricing scheme* for the algorithm. At each iteration, all non-basic arcs are re-priced, and the arc with the most negative price is selected as the entering arc. Kelly and Neill (1993) implemented a variation of the arc block pricing scheme, which is called *arc sample* [48]. Instead of selecting the entering arc from among the required number of consecutive arcs, this method considers arcs at constant intervals, called the skip factor, from throughout the entire arc set. Andrew (1993) studied practical implementation of minimum cost flow algorithms and claimed that his implementations worked very well over a wide range of problems [6].

Istvan reviewed a collection of some known pricing schemes in the original simplex algorithm [46]. They are *First improving candidate*, *Dantzig rule*, *Partial pricing*, *Multiple pricing* and *Sectional pricing*. These schemes can be applied to NSA. First improving candidate chooses the first violate arc as the entering arc. It is cheap but it usually leads to a very large number of iterations. In Dantzig rule all non-basic arcs are checked (full pricing) and one which violates the optimality condition the most is selected. This rule is quite expensive but overall is considerably better than the previous method. The Partial pricing scans only a part of the non-basic arcs and the best candidate from this part is selected. In the next step, the next part is scanned, and so on. In Multiple pricing, some of the most profitable candidates (in terms of the magnitude) are selected during one scanning pass. They are updated and a sub-optimization is performed involving the current basis and the selected candidates using the criterion of greatest improvement. The Sectional pricing behaves as a kind of partial pricing, but in each iteration sections or clusters of arc are considered.

In recent years, several researches have been devoted on network simplex algorithm. Muramatsu (1999) used a *primal-dual symmetric pivoting rule* and proposed a new scheme in which the algorithm can start from an arbitrary pair of primal and dual feasible spanning tree [67]. Eppstein (1999) presented a *clustering technique* for partitioning trees and forests into smaller sub-trees or clusters [24]. This technique has been used to improve the time bounds for optimal pivot selection in the primal network simplex algorithm for minimum-cost flow problem. Lobel (2000) developed and implemented the *multiple pricing rules* to select an entering arc, a mixture of several sizes for the arc block [58]. A *general pricing scheme* for the simplex method has been proposed by Istvan (2001). His pricing scheme is controlled by three parameters. With different settings of the parameters, he claimed that it creates a large flexibility in pricing and applicable to general and network simplex algorithms [46]. Ahuja et al. (2001) developed a network simplex algorithm with $O(n)$ consecutive degenerate pivot [3]. They presented an *anti-stalling pivot rule*, based on concept of strong feasible spanning tree, which is described in the following section. Their rule uses a negative cost augmenting cycle to identify a sequence of entering variables.

5.2.5 Strongly feasible spanning tree

The definition of strongly feasible solution for Network Simplex Algorithm and a property are given below:

Definition 5-3 [2]: The basis structure (T, L, U) is strongly feasible if we can send a positive amount of flow from any node to the root along arcs in the spanning tree without violating any of the flow bounds. An equivalent way of stating this property is that no upward pointing arc of the spanning tree can be at its upper bound and no downward pointing arc can be at its lower bound. An example of a strongly feasible basis is given in Figure 5-3. Note that the current flow and upper bound of every arc are given on each arc in the figure. The Lower bound of the arcs is zero.

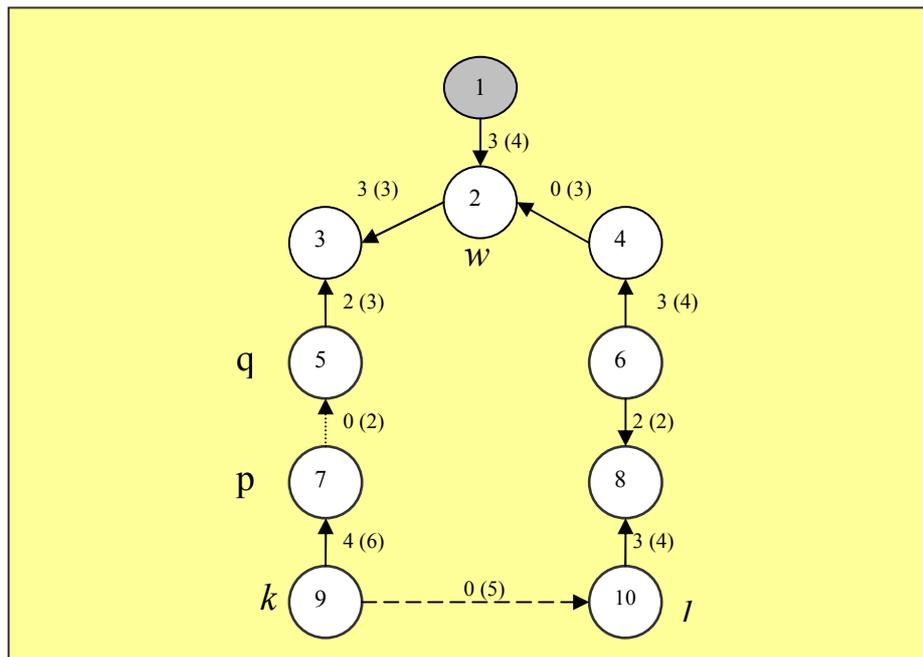


Figure 5-3: An example of strongly feasible spanning tree [2]

The network simplex algorithm can maintain a strongly feasible basis at every iteration. In order to do this, the initial basic solution, which was described in the previous section, should be strongly feasible. The algorithm may also select the leaving arc appropriately so that the next basis would be also strongly feasible. Suppose that the entering arc (k, l) is at its lower bound and node w is the first common predecessor of nodes k and l . Let W be the basis cycle formed by adding arc (k, l) to the basis tree. This cycle consists of the basis path from node w to node k , the arc (k, l) , and the basis path from node l to node w . After updating the flow, the algorithm identifies blocking arcs. If the blocking arc is unique, then it leaves the basis. If there are more than one blocking arcs, then the algorithm should select the leaving arc to be the last blocking arc encountered in traversing W along its orientation starting at node w . For example, in Figure 5-3, the entering arc is $(9, 10)$, the blocking arcs are $(2, 3)$ and $(7, 5)$, and the leaving arcs is $(7, 5)$. It

can be shown that the above rule guarantees that the next basis is strongly feasible [2]. A strongly feasible basis has the following property.

Property 5-2 [2]: Due to degeneracy, cycling may occur in the network simplex algorithm. By maintaining strongly feasible basis due to Cunningham (1976, 1979), cycling can be prevented without restrictions on the entering variable.

5.3 Simulation software

In order to evaluate our model and the employed algorithms in this thesis, we developed a piece of software. Our software is called DSSAGV (Dynamic Scheduling Software of Automated Guided Vehicles).

The main objectives of the software were:

- To define a few terminal ports and their layout.
- To simulate a Job Generator.
- To test and measure the efficiency of the algorithms.
- To solve the scheduling problem (defined in Chapter 4) in both static and dynamic aspects.
- To produce a system for Dynamic Scheduling of Automated Guided Vehicles.
- To produce a set of benchmarks for the future research.

We implemented the software in C++ programming language along with Borland Database Engine (BDE) for its database [39]. In this section, the features of our software are described briefly. Then the detail implementation of the standard version of Network Simplex Algorithm is presented. After that, we explain the input and output of the algorithm for an example.

5.3.1 The features of our software

Figure 5-4 shows the main screenshot of the software. It shows a couple of vessels, six Quay Cranes (QCs), one Rubber Tyred Gantry Crane (RTGC) in each block of the Storage Area and several AGVs. The figure also shows the main menu as well as several buttons including ‘Port’, ‘Route’, ‘Containers’, ‘Vehicles’ and ‘Process’. These buttons have been shown under the main menu and designed as hotkeys to facilitate the software execution.



Figure 5-4: The main screenshot of the software.

Some important features of DSSAGV are described briefly as follows:

- The user can define a few ports, a number of blocks in the yard, a number of working positions or cranes in the berth and a number of Automated Guided Vehicles in each port. The 'port' button activates this feature.
- A facility to generate a random distance between every two points in the yard or berth has been considered. The user can change the distance. The 'route' button activates this feature.
- At the beginning of the process, the start location of each vehicle may be any point of the port. The user can define or change the ready time of the vehicles at the start location and the location as well. But at the first stage, the software generates them randomly. The 'vehicle' button activates this feature.
- A Job Generator was designed and implemented in the software. For static and dynamic fashion, a few container jobs may be generated to transport from their source to their destination. Either the source or destination of each job is the quay side, which can be chosen

randomly by the Job Generator. There are three options for quay cranes: single crane and multiple cranes randomly and circular. In the first option, crane number 1 is selected to handle the jobs whereas in the second option one crane, among different cranes in the berth, will be selected to handle the jobs. In the last option, choosing the crane number will be circular; the first job for the first crane, the second job for the second crane and so on. After the next job is allocated for the last crane, the turn goes to the first crane.

- The initial time of the operation, time window for the cranes and vehicles are defined by the user. The first parameter plays a role as the ship arrival's time; the second one determines the processing time of a container job by the crane; namely the time between two consecutive jobs. The last one is the time taken by a vehicle to pick-up (drop-off) the job from (to) the crane. We assume some default values for these parameters.
- The user can monitor some indices to measure the efficiency of the model and algorithm. The waiting or delay time for every job, the number of jobs and the total travelling and waiting times for every vehicle are calculated in the static and dynamic problems. The 'process' button activates another screenshot of the software. In the screenshot², several panels and facilities for verification and validation of the software have been designed and implemented to help the user. These panels are "Static", "Model", "Dynamic", "Result", "Graph", "Algorithm" and "Performance". The 'Static' and 'Dynamic' panels are used for the static and dynamic problems. The input and output of the algorithm, before and after solving a problem, can be observed by the user. The 'Model' panel shows the input and output of the algorithm. The 'Algorithm' panel shows the employed algorithms from which the user can choose one. The 'Performance' panel shows the CPU-Time and the number of iterations required to solve the model. The 'Graph' panel shows and compares the 'Quay crane time' (when the crane is ready to pick-up/drop-off the job from/on the vehicle), the 'Vehicle time' (when the vehicle is ready to deliver/pick-up the job to/from the crane) and the 'Actual time' (the maximum of 'Quay crane time' and 'Vehicle time').
- A real time analogue clock has been designed and implemented. In dynamic aspect, the performance of different parts of the software can be monitored by the clock.
- A relational database has been designed and implemented, along side the software by Borland Database Engine (BDE). The relationships between tables or the Entity Relationship Diagram (ERD) of the database have been illustrated by Figure 5-5.

² <http://privatewww.essex.ac.uk/~hrashi>

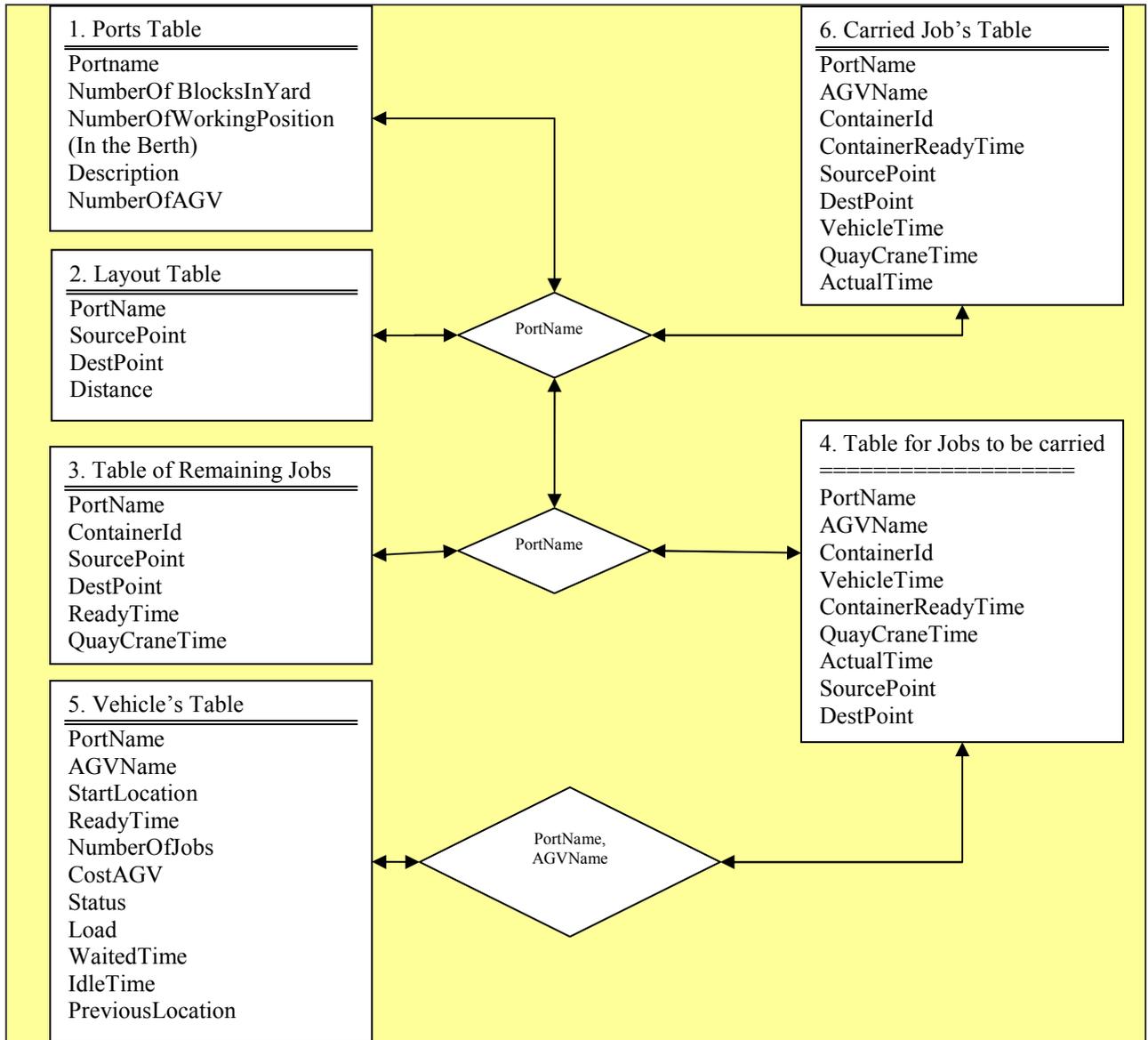


Figure 5-5: Relationships between the tables of the Database.

There are six tables in the database and their fields are shown in Figure 5-5. The relationships between the tables have been illustrated by one or two fields into the diamond. The Table 1 is considered to store port specification, including the name of ports, the number of blocks in the yard, the number of cranes or working positions, the number of AGVs and a description for the port. The distance between every different two points either in the yard or in the berth will be stored in Table 2. While the system is doing its processes, the remaining jobs, the jobs to be carried for each vehicle and the vehicles status are updated in Tables 3, 4 and 5 respectively. The start location, previous location, time travelled and waited of the vehicles are stored and updated in Table 5. The ready time of the vehicles to pick-up (deliver) the job from (to) the

crane, the time that the crane picks-up (delivers) the job from (to) the vehicle and the ‘Actual time’ of the job served (maximum of the two former times) are stored in Table 4.

5.3.2 The implementation of NSA in our software

Before going to the detail of our implementation, Unimodularity theorem in network flow problem is stated.

Theorem 5-5 (Unimodularity theorem) [2]: For every network flow problem with integer data, every basic feasible solution and, in particular, every basic optimal solution assigns an integer value to the flow of every arc.

To get a higher performance in our software, we considered Theorem 5-5 in the implementation. There is no multiplication, division and floating point variable during the process.

We implemented the standard version of Network Simplex Algorithm (see Figure 5-2). The operations of the algorithm were described in Section 5.2.2. As we mentioned, the pricing rule or scheme to choose the entering arc in Step 1 determines the speed of algorithm. In the literature, we reviewed the pricing rules. Actually, there is the trade-off between time spent in pricing at each iteration and the ‘goodness’ of the selected arc in terms of reducing the number of iterations required to reach the optimal solution. The *First improving candidate* and *Dantzig rule* represent two extreme choices for the entering arc. Other pricing schemes strike an effective compromise between these two extremes and have proven to be more efficient in practice [2]. Kelly and Neill [48] implemented several pricing schemes and ran their software for different classes of minimum cost flow problems. In their results, the *block pricing scheme* provided a better performance compared with others. We therefore chose the block pricing scheme. This scheme is based on dividing the arcs of the graph into a number of subsets of specified size. A block size of between 1% and 8.5% of the size of the arcs in the graph has been recommended by Grigoriadis [48], for large MCF problems. We set the number to 5% by the try and error. In our software, there is a procedure to select the entering arc. The flowchart of this procedure is depicted by Figure 5-6.

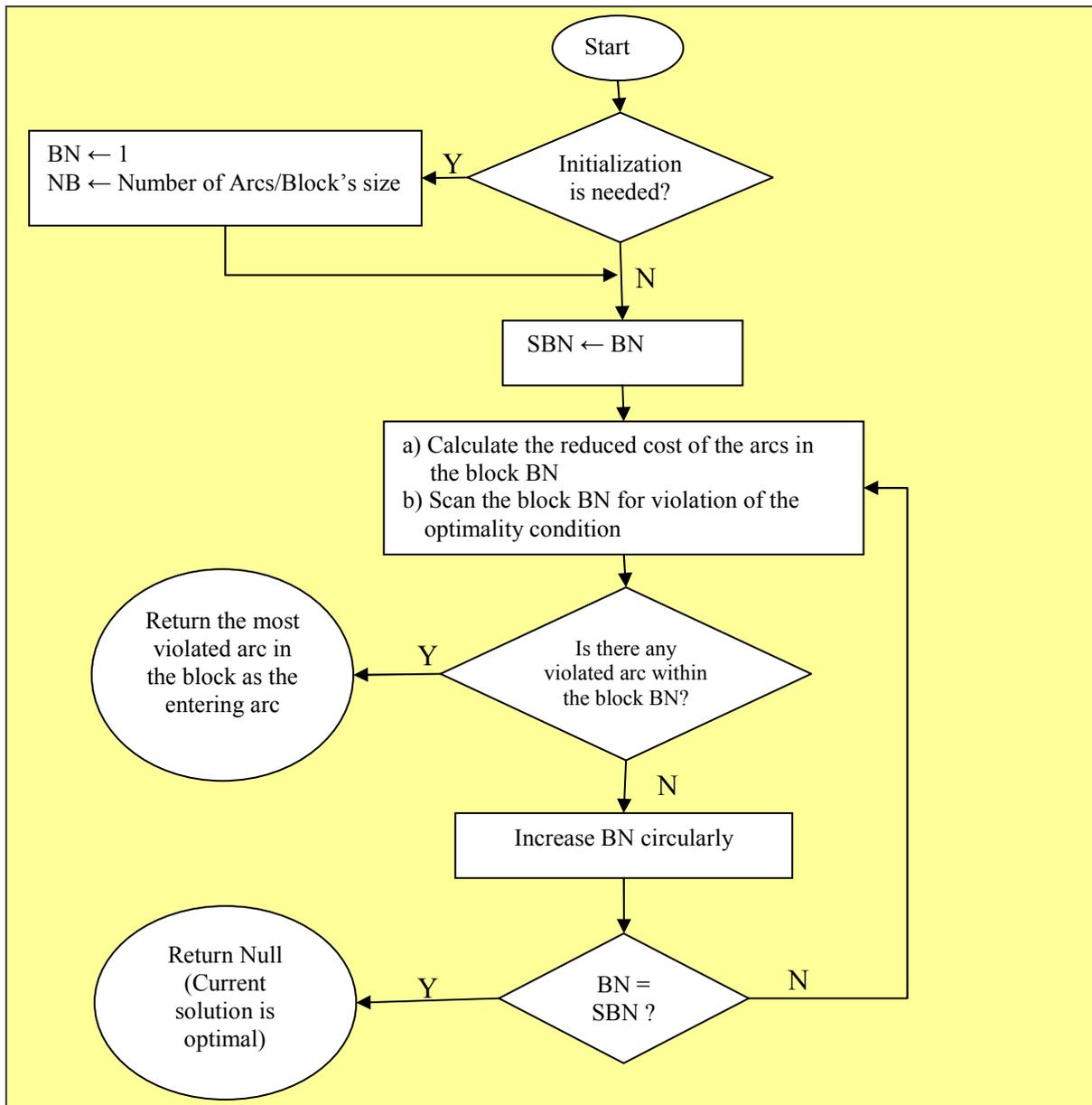


Figure 5-6: Flowchart of Network Simplex Algorithm (*Block Pricing Scheme*) to select an entering arc.

We now explain the flowchart. To solve every problem, it is needed to initialize the block number (BN) to 1 and to calculate the number of blocks (NB). At each iteration, the reduced cost of the arcs in a block, identified by BN, is calculated and the optimality condition is checked. Only the arcs of one block are re-priced. Then, the most violated arc within the block is selected as the entering arc. If there is no violated arc in the block, the block number (BN) is increased circularly (1, 2, ..., NB, 1,...). If there is no violated arc in the graph (BN=SBN), then the current solution is optimal.

5.3.3 How the program works

As we mentioned in Section 4.5, the container jobs to be served and the AGVs to be deployed were considered as nodes in the MCF-AGV model. There were M AGV nodes, $2 \times N$ job nodes and a sink node, all together $M+2N+1$ nodes in the model. The AGV nodes were considered as supply nodes and the sink was a demand node. Each job was considered with a couple of nodes, Job-Input and Job-Output nodes (see Section 4.5).

A graph $G_{MCF-AGV} = (GS, NPS, APS)$ is made by the software. In the graph, NS is a set of nodes and AS is a set of arcs; NPS , APS are the properties of the nodes and arcs, respectively. We defined the NS and AS and their elements (see Section 4.5 of Chapter 4) as below:

$$NS = SAGVN \cup SJIN \cup SJOUT \cup SINK$$

$$AS = ARC_{inward} \cup ARC_{outward} \cup ARC_{auxiliary} \cup ARC_{intermediate}$$

As an example, assume there are 2 AGVs to be deployed and 2 jobs to be served. The nodes and arcs with their properties are:

- $SAGVN = \{1,2\}$; $NPS(1)=NPS(2)=1$
- $SJIN = \{3,5\}$; $NPS(3)=NPS(5)=0$
- $SJOUT = \{4,6\}$; $NPS(4)=NPS(6)=0$
- $SINK = \{7\}$; $NPS(7)=-2$
- $ARC_{inward} = \{(1,3),(1,5),(2,3),(2,5)\}$; $APS(1,3)=[0,1,132]$, $APS(1,5)=[0,1,400]$, $APS(2,3)=[0,1,80]$, $APS(2,5)=[0,1,360]$
- $ARC_{intermediate} = \{(4,5), (6,3)\}$; $APS(4,5)=[0,1,280]$, $APS(6,3)=[0,1,10000]$
- $ARC_{outward} = \{(1,7),(2,7),(4,7),(6,7)\}$; $APS(1,7)=APS(2,7)=APS(4,7)=APS(6,7)=[0,1,0]$
- $ARC_{auxiliary} = \{(3,4),(5,6)\}$; $APS(3,4)=APS(5,6)=[1,1,0]$

Figures 5-7 to 5-9 illustrate the MCF-AGV model, the input and output of the algorithm for above example, respectively. In Figure 5-7, nodes 1 and 2 are AGV nodes, nodes 3 and 5 are Job-Input nodes, nodes 4 and 6 are Job-Output nodes, and node 7 is the Sink node.

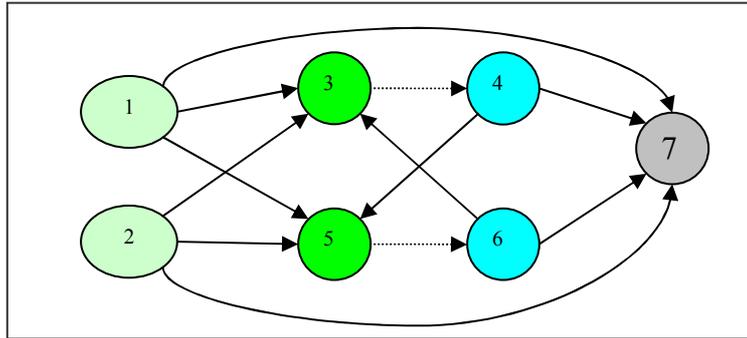


Figure 5-7: An example of the MCF-AGV model for 2 AGVs and 2 jobs in our software.

```

1: p min 7 12
2: c Create Supply nodes
3: n 1 1
4: n 2 1
5: c Create Demand node
6: n 7 -2
7: c Create Inward arcs from every vehicle node to every Job-Input node
8: a 1 3 0 1 132
9: a 1 5 0 1 400
10: a 2 3 0 1 80
11: a 2 5 0 1 360
12: c Create Outward arcs from every vehicle nodes to the Sink node
13: a 1 7 0 1 0
14: a 2 7 0 1 0
15: c Create Auxiliary arcs from every Job-Input node to its Job-Output node
16: a 3 4 1 1 0
17: a 5 6 1 1 0
18: c Create Outward arcs from every Job-Output node to the Sink node
19: a 4 7 0 1 0
20: a 6 7 0 1 0
21: c Create Intermediate arcs from every Job-Output node to others Job-Input nodes
22: a 4 5 0 1 280
23: a 6 3 0 1 10000

```

Figure 5-8: The input of the algorithm (NSA) in DIMACS³ format

In Figure 5-8, the prefixes of ‘p’, ‘c’, ‘n’ and ‘a’ identify defining the problem, comments, nodes and arcs in the graph, respectively. The first line in the figure defines a problem with 7 nodes and 12 arcs, which has to be minimized. Lines 3 and 4, define supply nodes with amount of flow to be sent into the network. Line 6 defines the Sink node with amount of its demand. Other lines in the figure specify the arcs with their tail and head nodes, lower and upper bounds, and transition cost.

³ Centre for Discrete Mathematics and Theoretical Computer Science

Figure 5-9 shows the output of the algorithm. In the figure, the prefixes of 's' and 'f' identify the objective function and solution for the problem. The numbers after prefixes of 'f' determine which arcs have been chosen as the optimal paths for the vehicles. According to the solution for this example, jobs 1 and 2 are served by AGV 2 and there is no job for AGV 1.

```

1 : c Output to minimum-cost flow problem.
2 : c The problem was solved with the
3 : c standard version of network simplex
4 : c algorithm.
5 : c
6 : c It needed 6 iteration(s) in 0 second(s).
7 : s Objective function: 360
8 : f 2 3 1
9 : f 1 7 1
10: f 3 4 1
11: f 5 6 1
12: f 6 7 1
13: f 4 5 1
14: c
15: c All other flow variables are zero

```

Figure 5-9: The output of the algorithm (NSA) in DIMACS format

5.3.4 The circulation problem

There is a special case for the MCF model, which have no supply nodes and demand nodes. Given $G = (N, A)$ and b_i ($i \in N$) as the amount of supply/demand flow at node i , for the Minimum Cost Flow problem (see Definition 4-8), the circulation problem is defined as follows:

Definition 5-4 [2]: The circulation problem is a Minimum Cost Flow problem with only transshipment nodes; that is, $b_i=0$ for all $i \in N$.

The circulation problem may be occurred in the solutions for the MCF-AGV model. If every AGV could not arrive before the appointment times of the Job-Input nodes (the transition cost from every AGV to the Job-Input node incurs the Penalty) and the cost between any two distinct jobs has not Penalty, the circulation will be happened. This problem can be demonstrated by an example in Figure 5-10. In the figure, the number on each arc is its cost and P shows the penalty (see Section 4.5.2 for the cost and Penalty).

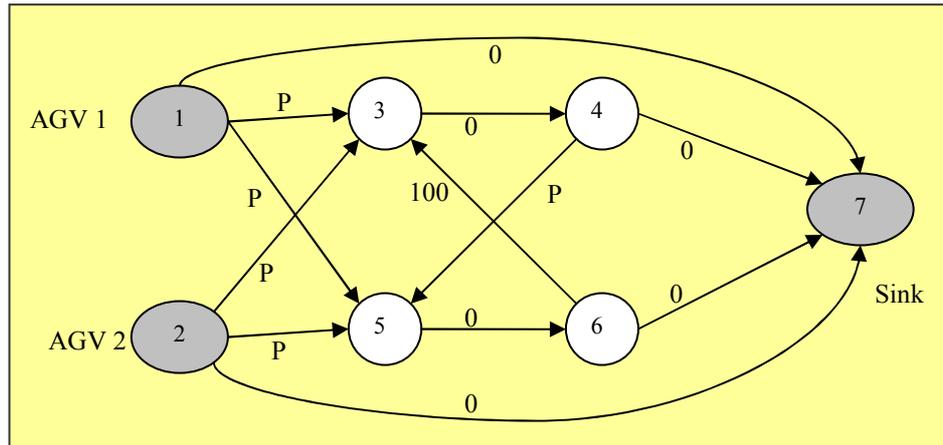


Figure 5-10: An example of the circulation problem ($P = \text{Penalty}$)

In the figure, there are two AGVs and two container jobs. In order to send two units flow from AGV nodes 1 and 2 to the Sink node with minimum cost, the solution is $1 \rightarrow 7$ and $2 \rightarrow 7$. Other nodes have one unit input and one unit output flow ($3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 3$), according to the constraints. The cost of problem is $100 + P$, which is less than any other possible solution in the network. In this case, neither job1 nor job2 is served.

Although the circulation problem never has happened in our experience (in static aspect), the following operations are performed to fix the problem. When the solution became ready, status of every job is checked to see whether it was assigned to a vehicle or not. There are two solutions for the problem. The first solution is to assign the remaining jobs to an idle vehicle. This scheme has a higher priority because moving the vehicles is preferred over their stopping. Among the idle vehicles, a vehicle with minimum cost is assigned to the job. This process continues until there is no remaining job. If the first solution could not solve the problem (no idle vehicle), the second solution is to distribute the remaining jobs among the vehicles randomly.

5.4 Experimental results

In this section, the results of our implementation and running the algorithm, to tackle the static problem of the MCF-AGV model, are presented. In the static problem the number of jobs, the distance between source and destination of the jobs, and the number of vehicles don't change (see Assumption 4-9). The values in Table 5-1 were used as parameters in the objective function, for the port specification and to generate the jobs. We considered ECT (European Container

Terminal) [23] for the port specification. It includes 7 quay-cranes, 32 automatic stacking cranes, and a maximum of 50 AGVs is in operation.

Table 5-1: Values of parameters for the simulation

Parameters	Values
Weight of Waiting Times for the AGVs (W_1 in the costs of the objective function)	1
Weight of Travelling Times for the AGVs (W_2 in the costs of the objective Function)	5
Number of AGVs in the port	50
Number of Quay Cranes	7
Number of Blocks in the yard (Storage area inside the port)	32
Time Window of the Cranes (the duration of discharging/loading a container)	120 second
The Distance Table (see Table 4-1)	Uniform Random Distribution between 1 and 100
Time Window of the Vehicles, time to unload/load a job	2 Second
P as a penalty (see the costs of the MCF-AGV model in Chapter 4)	10000

Some outputs of running the program in static fashion were taken. Table 5-2 shows the result, including the number of jobs, the number of nodes and the number of arcs in the MCF-AGV model. The CPU-time required to solving the MCF-AGV problems also is shown in the table.

Table 5-2: Experimental results of Network Simplex Algorithm in static fashion

Problem	Number of Jobs	Number of Nodes	Number of ARCS	CPU-Time (Second)
1	500	1,051	275,550	1
2	700	1,451	525,750	2
3	1,000	2,051	1,051,050	4
4	1,200	2,451	1,501,250	6
5	1,300	2,651	1,756,350	7
6	1,400	2,851	2,031,450	6
7	1,500	3,051	2,326,550	9
8	1,500	3,051	2,326,550	11
9	1,600	3,251	2,641,650	13
10	1,700	3,451	2,976,750	15
11	1,800	3,651	3,331,850	17
12	2,000	4,051	4,102,050	27
13	2,100	4,251	4,517,150	28
14	2,200	4,451	4,952,250	33
15	2,300	4,651	5,407,350	47
16	2,500	5,051	6,377,550	49
17	2,700	5,451	7,427,750	59
18	2,710	5,471	7,482,360	64
19	2,715	5,481	7,509,740	65
20	2,718	5,487	7,526,192	66
21	2,800	5,651	7,982,850	68
22	2,900	5,851	8,557,950	86
23	2,930	5,911	8,734,380	100
24	2,940	5,931	8,793,590	99
25	3,100	6,201	9,768,150	122
26	3,200	6,401	10,403,250	136
27	3,300	6,601	11,058,350	137

Note that those results have been collected by running our software on Pentium PC with 2.4 GHz processor and 1GB RAM. Obviously on different computers the CPU-time is different.

The CPU-time required to solve the MCF-AGV model is demonstrated by Figures 5-11 and 5-12, according to both the number of jobs and number of arcs in the graph model. Based on our observations the estimated values by a polynomial equation for the CPU-time are also shown on the figures. We assumed degrees 3 and 2 for the polynomial equations, respectively in Figure 5-11 and Figure 5-12.

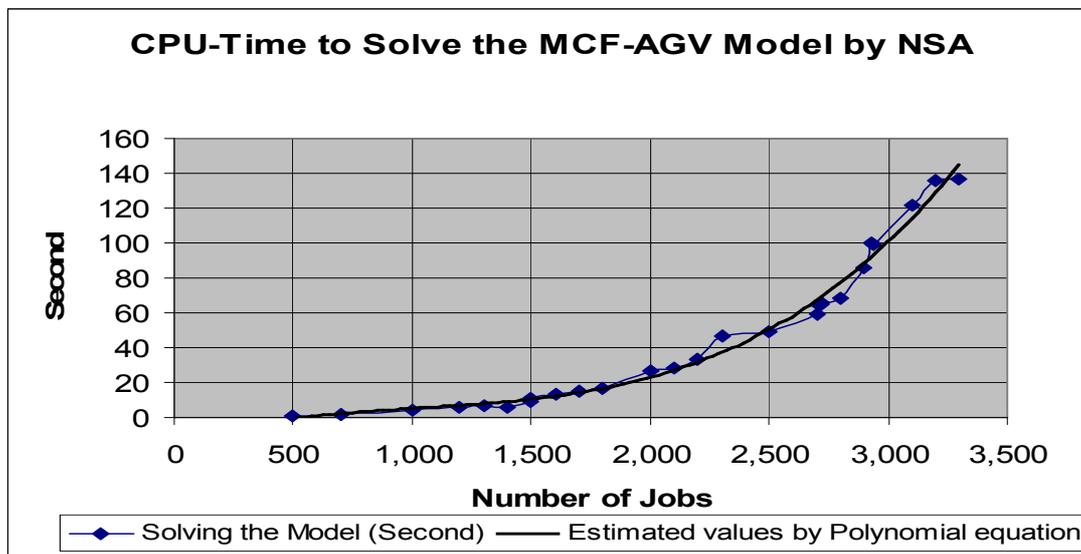


Figure 5-11: CPU-Time required to solve the problem by Network Simplex Algorithm, based on the number of jobs

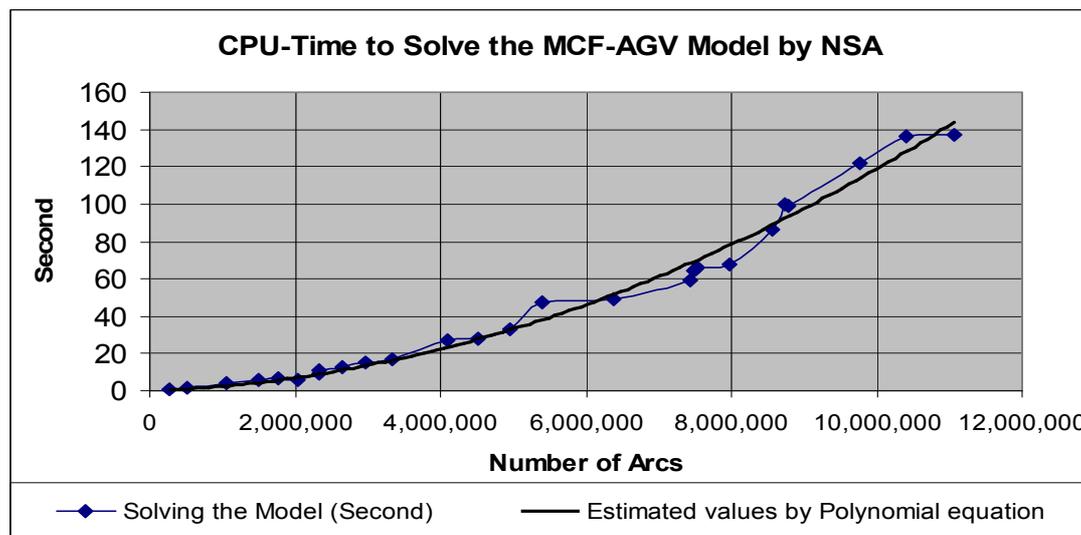


Figure 5-12: CPU-Time required to solve the problem by Network Simplex Algorithm, based on the number of arcs

From the figures, we can observe that:

Observation 5-1: The Network Simplex Algorithm (NSA) is fast and efficient. It could find out the global optimal solution for the problem of 3,000 jobs and ten millions arcs in the MCF-AGV model within two minutes.

Observation 5-2: From the figures, it seems that NSA is run in polynomial time to solve the MCF-AGV model, in practice.

There are two different types of iteration in NSA, degenerate and non-degenerate [2]. In every non-degenerate iteration, the value of the objective function is decreased whereas degenerate iterations do not change the objective function's value. In the degenerate iterations, a flow change of zero causes cycling. In the literature, Grigoriadis experienced that cycling is rare in practical application [48]. Observations 5-1 confirms the experience.

In order to confirm that NSA is run in polynomial time to solve the MCF-AGV model (Observations 5-2), we estimated complexity of the algorithm in the next section.

5.5 An estimate of the algorithm's complexity in practice

The time complexity can be expressed in CPU-Time required to solve the MCF-AGV model. The CPU-Time is estimated based on the number of jobs and number of arcs in the graph model. Based on Observations 5-1 and 5-2, we considered the following equations to estimate the CPU-Time:

$$\begin{aligned} CPU - Time_{NSA} &= a_1 \times \text{NumberofJobs}^3 + a_2 \times \text{NumberofJobs}^2 + a_3 \times \text{NumberofJobs} \\ CPU - Time_{NSA} &= b_1 \times \text{NumberofArcs}^2 + b_2 \times \text{NumberofArcs} \end{aligned}$$

The experimental results in Table 5-2 were used to estimate the parameters of 'a1', 'a2', 'a3', 'b1' and 'b2' in the equations. The estimation's results for the parameters have been shown in Table 5-3 and Table 5-4. The Coefficient section of the tables specify values for 'a1', 'a2', 'a3', 'b1' and 'b2' in the equations.

Table 5-3: Regression result for CPU-Time required to solve the problem by NSA (Based on the number of jobs)

Multiple R	R-Square	Adjusted-R-Square	Standard Error	Observations		
0.99	0.99	0.94	5.24	27		
	<i>DF</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance-F</i>	
Regression	3.00	47310.45	15770.15	574.17	0.00	
Residual	24.00	659.18	27.47			
Total	27.00	47969.63				
	<i>Coefficients</i>	<i>Standard-Error</i>	<i>t-Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>
X Variable 1	1.54E-02	6.14E-03	2.50	1.95E-02	2.71E-03	2.81E-02
X Variable 2	-1.83E-05	5.52E-06	-3.31	2.94E-03	-2.96E-05	-6.87E-06
X Variable 3	8.11E-09	1.19E-09	6.83	4.58E-07	5.66E-09	1.06E-08

Table 5-4: Regression result for CPU-Time required to solve the problem by NSA (Based on the number of arcs)

Multiple R	R-Square	Adjusted-R-Square	Standard-Error	Observations		
0.99	0.99	0.95	5.07	27		
	<i>DF</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance-F</i>	
Regression	2	47327.497	23663.75	921.29	2.0415E-23	
Residual	25	642.1324	25.68			
Total	27	47969.63				
	<i>Coefficients</i>	<i>Standard-Error</i>	<i>t-Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>
X Variable 1	1.40E-06	6.30E-07	2.23	3.50E-02	1.07E-07	2.70E-06
X Variable 2	1.05E-12	7.41E-14	1.41	1.98E-13	8.95E-13	1.20E-12

Based on the Coefficients in the tables for values of the parameters, we have the following equations for the CPU-Time to solve the MCF-AGV model:

$$CPU - Time_{NSA} = 8.11 \times 10^{-9} \times \text{NumberofJobs}^3 - 1.83 \times 10^{-5} \times \text{NumberofJobs}^2 + 0.154 \times \text{NumberofJobs}$$

$$CPU - Time_{NSA} = 1.05 \times 10^{-12} \times \text{NumberofArcs}^2 + 1.4 \times 10^{-6} \times \text{NumberofArcs}$$

The degree of the second equation is less than the first one's, because the number of arcs is extremely greater than the number of jobs.

Note that for any prediction, the equation for the CPU-Time depends on other factors such as the speed of processor, other active programs when the problem is being solved in multi-task operating system and so on, in practice. Our program has been run on Windows-2000 computer with Pentium 2.4 GHz processor in the normal situation.

More details about information in the two tables and their explanations are as follow:

- **DF:** It stands for the degrees of freedom. There are 27 samples in this experiment.
- **SS:** It refers to the Sum of Squares differences between the values of curve fitted and the average of dependent variable (for Regression), and the Sum of Squares differences between the actual values of dependent variable and the values of curve fitted (for Residual).
- **MS:** It stands for the Mean Square, which is calculated by dividing the SS over the DF.
- **F:** This is a test statistic. A large value indicates that the estimated equation is significant in the sense, i.e. it is unlikely to have resulted from random variation.
- **Significance-F:** It gives us the probability that we would get this result by random chance. This value for the both estimations is zero.
- **R-Square:** This is the percentage of the SS of Regression over the SS of Total. It reveals how closely the values of the estimated curve correspond to the actual data. Its value is 0.99 for the both estimations.
- **Multiple-R:** This is the square root of the R-Square. It is the correlation between the dependent variable and curve fitted.
- **Adjusted-R-Square:** This indicates the percentage of the variations explained by the model. Its value for the both estimations is 0.99. This is useful because we assumed two independent variables in the model.
- **Standard-Error:** This is the square root of the Residual Mean Square discussed above. It is essentially the standard deviation of the points around the regression curve. This is very useful in evaluating how big of a mistake we are likely to make when using the model for prediction. Its value is 5.24 and 5.07, respectively for the both estimations.
- **t-Stat:** This is a statistic for a null hypothesis that the coefficient is zero. The ‘t-Stat’ value is calculated by dividing the coefficient by its standard error. The large value of ‘t-Stat’ indicates that it is low probability to have occurred by chance. Usually a ‘t-Stat’ greater than 2 is considered to indicate a model is significant.
- **P-value:** This gives a probability that the coefficient is zero. Its value for each coefficient of the both estimations is almost zero.
- **Lower and Upper 95%:** These give an upper and lower bound on a 95% confidence interval for the coefficients. Given α as a coefficient, S_α as its standard error and t as the critical value of the t distribution at 95% confident limit, the values are calculated as follows:

$$\alpha \pm t \times (S_\alpha)$$

5.6 Limitation of the NSA in practice

The question is how big (the number of vehicles, the number of jobs) of a problem can be solved by NSA within time t , a minute for example? The answer is that there is no limitation in NSA, theoretically. In practice, the answer is based on the platform and implementation. Given the number of jobs and number of vehicles, N and M respectively in our formulation, there are $M+2\times N+1$ nodes and $M+M\times N+N\times(N-1)+2\times N$ arcs in the MCF-AGV model. The limitation is due to available memory to put the MCF-AGV model into. The largest problem, which has been solved by our software, was a MCF-AGV model consists of 11,058,350 arcs ($M=50$; $N=3,300$; see Table 5-2). Based on this maximum number of arcs and the related formula, the number of vehicles (M) and number of jobs (N) can be had different values. Hence, we have another observation from the experiment:

Observation 5-3: Although NSA is efficient and provides the optimal solution, it can only work on problem with certain limits in size. The limitation is due to available memory to put the MCF-AGV model into.

5.7 Summary and conclusion

In this chapter, the steps of network simplex algorithm were reviews. To select the next basic solution at each step of the algorithm, the literature over different pricing schemes was presented. Then, the standard version of Network Simplex Algorithm (NSA) with the block pricing scheme was applied to the MCF-AGV model (defined in Chapter 4). To test the program, Random data were generated and fed to the model for fifty vehicles.

Based on our experiment, now we can conclude that with simple network operation in the graph and specializations, Network Simplex Algorithm is efficient. Our software, which has been implemented in Borland C++ and run on a 2.4 GHz Pentium PC, could find the global optimal solution for 3,000 jobs and ten millions arcs in the MCF-AGV model within two minutes. Although the algorithm is efficient and provides the optimal solution, it can only work on problems with certain limits in size. When the size of problem goes beyond the limit, incomplete solution methods should be used.

Chapter 6: Network Simplex plus Algorithm and Dynamic Scheduling of AGVs

In Chapter 5, the static scheduling problem of Automated Guided Vehicles (AGVs) in container terminals was solved by the standard version of Network Simplex Algorithm (NSA). In this chapter, some modifications are applied to NSA to be obtained a novel version of the algorithm. The new algorithm then is applied to the dynamic scheduling problem of Automated Guided Vehicles in container terminal (the problem defined in Chapter 4 and modelled as the MCF-AGV).

6.1 Motivation

Although NSA is efficient, cycling may occur in the algorithm. Additionally, to tackle the dynamic scheduling problem in Chapter 4, we need more efficient algorithms. In dynamic problems, new jobs arrive continually, the fulfilled jobs are removed, and the distance between the source and destination of jobs may be changed. The objective of this chapter is to develop a new version of NSA, which avoids cycling and is faster. We call it Network Simplex plus Algorithm (NSA+). Like NSA, NSA+ is a complete algorithm, which means it guarantees optimality of the solution if it finds one within the time available.

6.2 The Network Simplex plus Algorithm (NSA+)

NSA+ is an extension of NSA. Compared with the standard version of NSA, it has two features. Firstly, it deals with the concept of strongly feasible solution [2]. Secondly, a mixture of heuristic approach and memory technique are used in NSA+. These features are explained below.

6.2.1 Anti-Cycling in NSA+

The first feature is related to maintaining the strongly feasible basis at each iteration (see Definition 5-3 in Chapter 5). At the beginning, NSA+ chooses a strongly feasible solution (see Step 0 of NSA in Chapter 5). In each pivot, the leaving arc is selected appropriately by the last

blocking arc in the cycle so that the next basis is also strongly feasible (see Figure 5-3 as an example). We avoid cycling in NSA+ by this feature (see Property 5-2 in Chapter 5).

6.2.2 Memory technique and Heuristic approach in NSA+

The second feature of NSA+ is concerned with the entering arc (Step 1 in Figure 5-2). In order to find the entering arc, there is a procedure in our software. The flowchart of this procedure is depicted by Figure 6-1.

The arcs in the graph are divided into several blocks with the same size. At each iteration, a packet of the violated arcs are collected. The capacity of the packet is more than the block's size and the most violated arcs are kept at the top of the packet. The number of most violated arcs may be a percentage of block's size. We set the block's size and number of most violated arcs to 200 and 25, respectively. For each problem, the number of blocks depends on the number of arcs in the graph and the block's size. In our software, DSSAGV, the blocks are identified by a Block-Number and the first one is chosen Randomly or by a Heuristic method (based on location of the largest cost in the graph, for example). To solve every problem, we need to initialize the Block-Number and calculate the number of blocks. At the initial stage, the packet is empty. Then, scanning of the arcs for violation of the optimality conditions among the blocks is performed circularly. At each scan, one violated arc (at most) from each block is put in the packet.

At the beginning of the entering arc procedure, the reduced costs of the most violated arcs in the previous stage are recalculated. If they violate the optimality conditions again, they are kept in the packet. Otherwise they could be replaced by new violated arcs. Then, some new violated arcs, based on scanning of arcs from the blocks, are put into the packet so long as it has empty place. At the end of the procedure if the packet is empty (there is no violated arc in the graph), then the current solution is optimal. Otherwise the packet will be sorted decreasingly, based on the absolute value of the reduced costs, and the most violated arc (at the top of the packet) will be chosen as the entering arc.

As we mentioned, there are two options to choose the first block, Randomly and Heuristically. With this aspect, NSA+ has two extensions:

- **NSA+R:** The entering arc procedure chooses the first block by Random selection.

- **NSA+H:** The entering arc procedure chooses the first block by a Heuristic method (based on location of the largest cost in the graph).

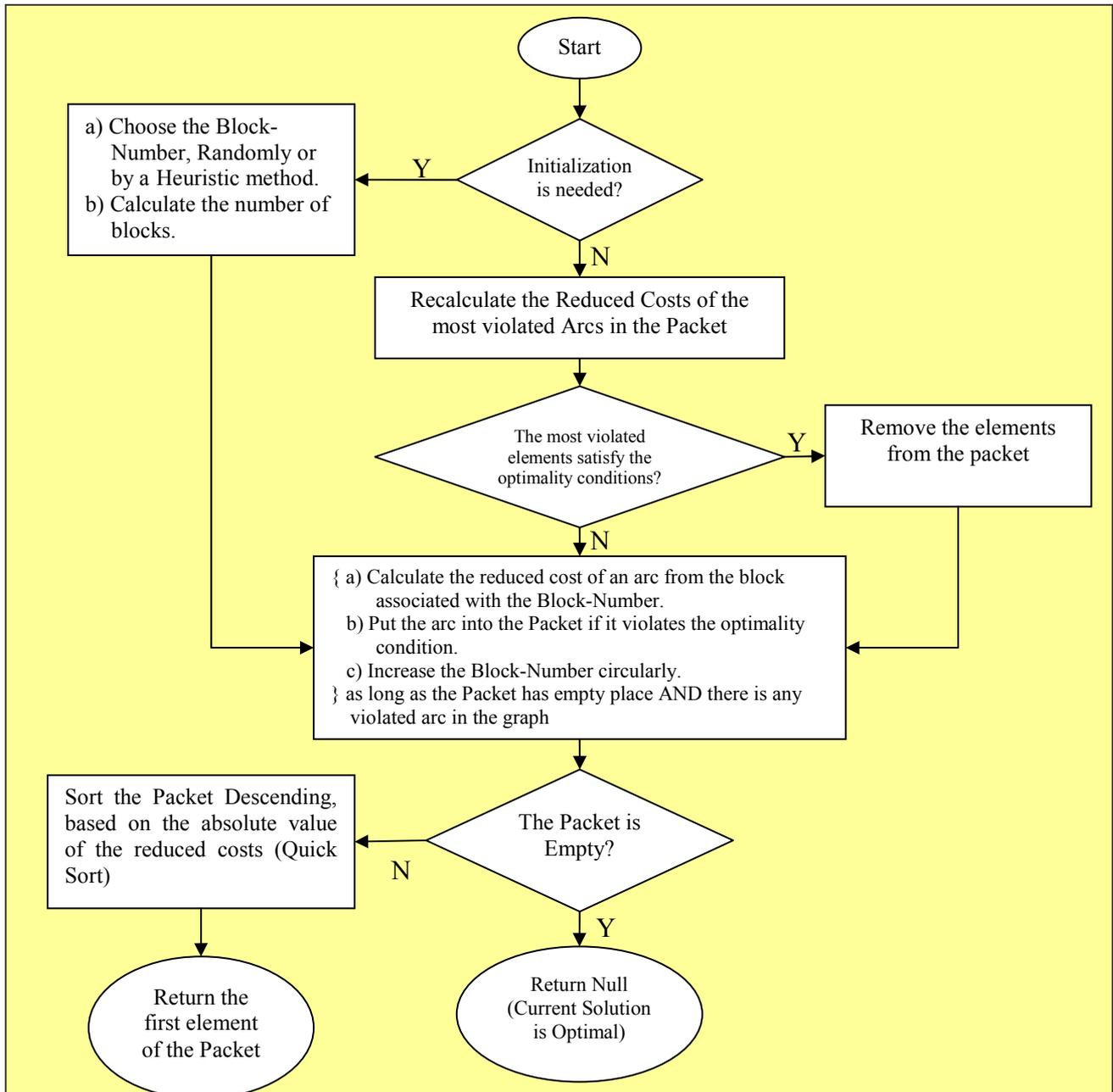


Figure 6-1: Flowchart of Network Simplex plus Algorithm to select an entering arc.

6.2.3 The differences between NSA and NSA+

The main difference between NSA and NSA+ are in the pricing scheme and the entering arc procedure. As we mentioned (see Section 5.2.4), the role of the pricing scheme is that how the

entering arc to be selected from the violated arcs in the graph. In this way, the flowcharts of Figures 6-1 and 5-6 can be compared. The differences between NSA and NSA+ are as follows:

- At each iteration, a packet of violated arcs from different blocks is collected in NSA+ and the most violated arc is selected as the entering arc, whereas NSA selects the most violated arc from one block.
- There is no memory technique in NSA while NSA+ uses a few elements at the top of the packet for the next iteration. It benefits from the current violated arcs for the next iteration.
- The first block is selected Randomly or by a Heuristic method in NSA+, whereas NSA always chooses the first block for scanning the violated arcs.
- In NSA, the leaving arc is selected by Step 2 (see Figure 5-2), while NSA+ considers a restriction on the step. NSA+ selects the leaving arc appropriately so that the spanning tree is strongly feasible at each iteration.

6.3 A comparison between NSA and NSA+

In order to compare the performances of the two algorithms, several static problems of the MCF-AGV model were generated randomly and solved by NSA and NSA+. This experiment was run on Windows-XP computer with 2.2 GHz Pentium processor and 1GB RAM when the number of vehicles is 50. Table 6-1 shows the results.

Table 6-1: Experimental results for a comparison between NSA and NSA+

Problem	Number of Jobs	CPU-Time by NSA (second)	CPU-Time by NSA+H (second)	CPU-Time by NSA+R (second)	Problem	Number of Jobs	CPU-Time by NSA (second)	CPU-Time by NSA+H (second)	CPU-Time by NSA+R (second)
1	50	0.005	0.005	0.005	17	1100	6.741	3.2532	4.644
2	60	0.005	0.005	0.005	18	1200	8.217	3.5577	6.885
3	70	0.009	0.0047	0.005	19	1300	11.996	5.1795	8.180
4	80	0.009	0.0048	0.005	20	1400	11.039	12.3	10.500
5	90	0.009	0.0045	0.010	21	1500	12.548	7.092	7.092
6	100	0.024	0.0093	0.011	22	1600	15.980	14.208	17.208
7	150	0.033	0.0327	0.033	23	1700	20.592	10.734	18.246
8	200	0.061	0.0468	0.050	24	1800	26.462	12.342	18.426
9	300	0.103	0.117	0.113	25	1900	36.526	17.081	27.294
10	400	0.600	0.225	0.525	26	2000	30.951	21.651	30.810
11	500	0.394	0.3795	0.381	27	2100	37.152	23.301	24.816
12	600	1.415	0.6984	0.745	28	2200	48.683	25.546	28.242
13	700	1.003	0.8298	0.950	29	2300	46.588	36.069	39.609
14	800	1.307	0.9981	1.298	30	2400	57.050	33.613	35.113
15	900	4.566	1.7718	3.272	31	2500	64.084	40.018	61.179
16	1000	5.259	2.5359	3.849	32	2600	70.553	62.952	55.735

Figure 6-2 shows the CPU-Time required to solve the MCF-AGV model by the both algorithms. From the figure and Table 6-1, we can observe that:

Observation 6-1: The average CPU-Time required to solve the problems by NSA+ is less than NSA.

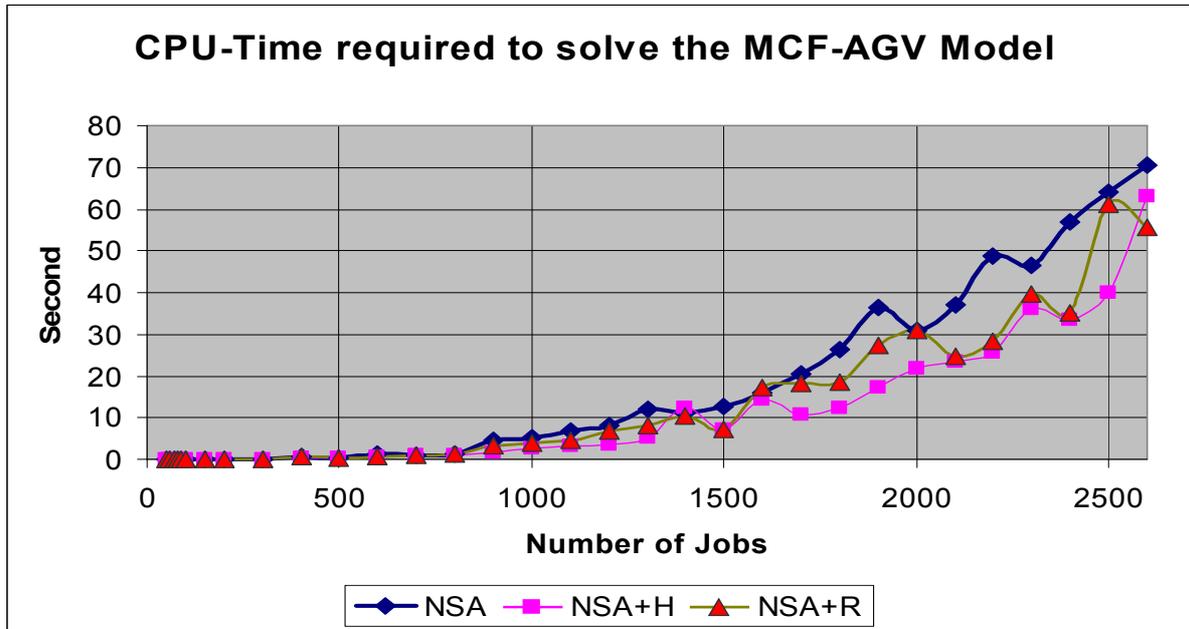


Figure 6-2: A comparison of CPU-Time required to solve the same problems by NSA and NSA+

In order to calculate the average CPU-Time required to solve the problems and to compare performance of the algorithms in this experiment, we introduce the following terms:

T_i^{NSA} : The CPU-Time used to solve the problem i by NSA.

T_i^{NSAH} : The CPU-Time used to solve the problem i by NSA+H.

T_i^{NSAR} : The CPU-Time used to solve the problem i by NSA+R.

PIH_i: The Percentage of Improvement in CPU-time used to solve the problem i by NSA+H compared with NSA.

PIR_i: The Percentage of Improvement in CPU-time used to solve the problem i by NSA+R compared with NSA.

TPIH: The Total Percentage of Improvement in CPU-Time used to solve the problems by NSA+H compared with NSA.

TPIR: The Total Percentage of Improvement in CPU-Time used to solve the problems by NSA+R compared with NSA.

W_i: The Weight of improvement for the problem i. In this experiment we consider the number of arcs in the MCF-AGV model for the weight. Given N jobs and M AGVs in the problem, the number of arcs is $M+M\times N+N\times(N-1)+2\times N$.

Now we calculate the percentage of improvements in the CPU-Time used for problem i by the following terms:

$$PIH_i = \frac{100 * (T_i^{NSAH} - T_i^{NSA})}{T_i^{NSA}}$$

$$PIR_i = \frac{100 * (T_i^{NSAR} - T_i^{NSA})}{T_i^{NSA}}$$

The total percentages of improvement in the CPU-Time used to solve the problems by NSA+H and NSA+R, compared with NSA, are calculated by the following equations:

$$TPIH = \frac{\sum_{i=1}^{32} W_i \times PIH_i}{\sum_{i=1}^{32} W_i} = -35.16\%$$

$$TPIR = \frac{\sum_{i=1}^{32} W_i \times PIR_i}{\sum_{i=1}^{32} W_i} = -21.28\%$$

In order to determine which factor, from the two features, made these improvements, we disabled the first feature (maintaining the strongly feasible spanning tree) and ran the software for some problems. We got the following observation:

Observation 6-2: There was no significant change in the improvement for the non-strongly feasible spanning tree. In the literature, Grigoriadis had experienced that cycling is rare in practical application [48]. Therefore, the second feature has significant impact on the CPU-Time required to solve the problems. In fact, the memory technique and scanning method are the most important features of NSA+.

6.4 Statistical test for the comparison

The CPU-time required to solve the problems by the two algorithms, NSA and NSA+, were analysed statistically. We tested the null hypothesis that the means produced by the two algorithms were statistically indifferent. Table 6-2 provides the test's result along with the critical

values of T-distribution for the particular degree of freedom. The T-test confirms that NSA+ is significantly better than NSA with 95% degree of confidence.

Table 6-2: The result of T-Test for the two algorithms, NSA and NSA+

Statistical Parameters	NSA+H vs. NSA	NSA+R vs. NSA
Observations	32	32
T-Test (Paired Two Sample For Means)	-4.1799	-3.3617
Degree of Freedom	31	31
Critical T-Value	-1.6955	1.6955

The values and hypotheses to do the test between the means of NSA+H and NSA are demonstrated by Figure 6-3. The hypotheses are the mean CPU-Time for NSA+H is greater than NSA or not. The value of ‘T-test’ and ‘Critical t-value’ are shown in the figure. As we can see the result of the T-Test is inside the reject region. The same examination is performed to do statistical test analysis between the means of NSA+H and NSA. The result of this test also shows the mean of CPU-Time for NSA+H is less than NSA.

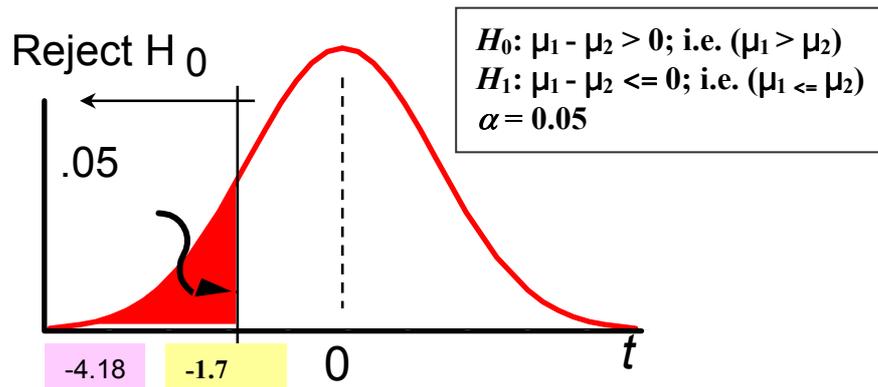


Figure 6-3: The T-Test acceptance and reject regions (NSA and NSA+H).

6.5 Complexity of Network Simplex plus Algorithm (NSA+)

Assume that the Maximum Flow, MF, in each of the m arcs, at maximum cost, C , for the minimum cost flow model. So there is an upper bound on the value of the objective function. This upper bound is given by $m \cdot C \cdot MF$. There are two different types of pivots in the algorithm, non-degenerate and degenerate pivots. The former is bounded by $m \cdot C$ because the number of non-degenerate pivots in the algorithm is bounded by $m \cdot C \cdot MF$ ($MF=1$ in the MCF-AGV model). The number of degenerate pivots is determined by the sum of nodes potential and maintaining the strongly feasible spanning tree. Given n as the number of nodes in the graph model, the sum of nodes potential is bounded by $n^2 \cdot C$. It is decreased at each iteration when the spanning tree is

strongly feasible [2]. A series of degenerate pivots may occur between each pair of non-degenerate pivots, and thus a bound on the total number of iterations is $m \cdot n^2 \cdot C^2$. Find the entering arc is $O(m)$ and sorting the packet is $O(K \cdot \text{Log}K)$ operation (K is size of the packet, $K=225$). Finding the cycle, amount of flow change, leaving arc and updating the tree are $O(n)$ operations. Hence the complexity of each pivot is $O((m + n) K \cdot \text{Log}K)$. Based on the complexity of the number of iterations and the complexity of each pivot, the total complexity of this algorithm is determined as follows:

$$O((m + n)mn^2C^2K\text{Log}K)$$

Given N and M ($M < N$), respectively, as the number of jobs and AGVs in the MCF-AGV model (see Section 4.5 in Chapter 4), we have the following results:

$$m=O(N^2) ; n=O(N)$$

Therefore, the total complexity of NSA+ Algorithm to tackle the MCF-AGV model is:

$$O(N^6)$$

We estimated the performance of NSA+ by the experimental results of Table 6-1 (see Section 5.5). The results support this complexity.

6.6 Software architecture for dynamic aspect

The architecture of main part of the software for the dynamic scheduling problem of Automated Guided Vehicles is demonstrated by Figure 6-4.

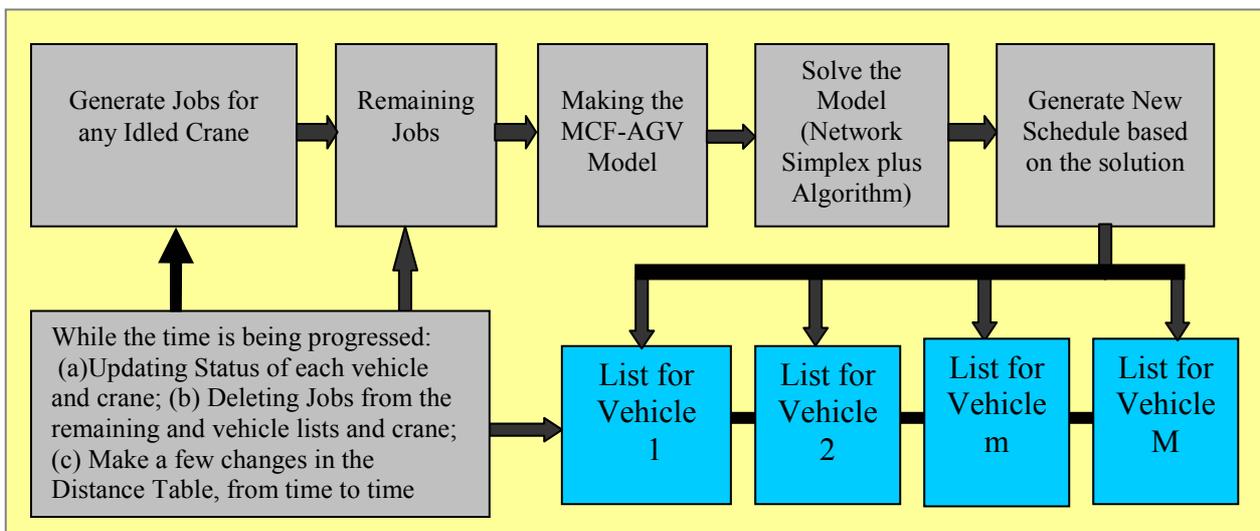


Figure 6-4: Block diagram of the software and algorithm (NSA+) for dynamic aspect

At the start of the process, the Job Generator generates a few jobs for each crane. These jobs will be appended to the remaining jobs, which is empty at the beginning. The remaining jobs are used to make up a MCF-AGV model. Then the model will be tackled by NSA+. The output of this algorithm is a few job sequences for the vehicles. Based on these sequences the software will prepare a job list for each vehicle.

Flowchart of Figure 6-5 demonstrates what is done in the real time processing and dynamic aspect while the time is being progressed. Note that the termination condition for the end of simulation is determined by meeting a specific time, ten hours or a day, for example.

At the beginning, based on the solution to the current problem, a job is assigned to each vehicle and crane. During the simulation, handling of the jobs by the cranes and vehicles are executed in parallel.

Briefly, the software does two tasks. The first task is related to updating status of the vehicles and cranes whereas the second one takes influence from any change in the problem or any idle crane. As depicted in the flowchart, the status of each crane and the travelling and waiting times of every vehicle are updated while the time is being progressed. At the same time, if the vehicles pick up the job from the quay side, the job will be removed from the crane, list of jobs for the vehicles and the remaining jobs. After that, the new job will be assigned to the vehicles and cranes. If a job has to be delivered to the crane on the quay side, it could not be removed until the meeting time between the crane and the vehicle. Note that, the appointment place of the jobs is on the quay side, not the yard side.

The second task refers to any change in the problem or status of the cranes. In the both cases, a new MCF-AGV model will be made by the remaining jobs (except the current job for every vehicle) and the new jobs (if there is any). The new model will be tackled by Network Simplex plus Algorithm from scratch. Then, the new solution will be used for updating the list of jobs for every vehicle. Every 5 minutes, the software makes a few random changes in the distance table in order to produce dynamic problems (see Table 4-1). Additionally, when the Job Generator finds out any idle crane, it has to generate a few jobs for the crane.

Note that generating jobs for any idle crane, making the MCF-AGV model, solving the model and generating new schedule for the vehicles are performed sequentially. These tasks are non-preemptive, i.e. when a task starts execution on the processor, it finishes to its completion.

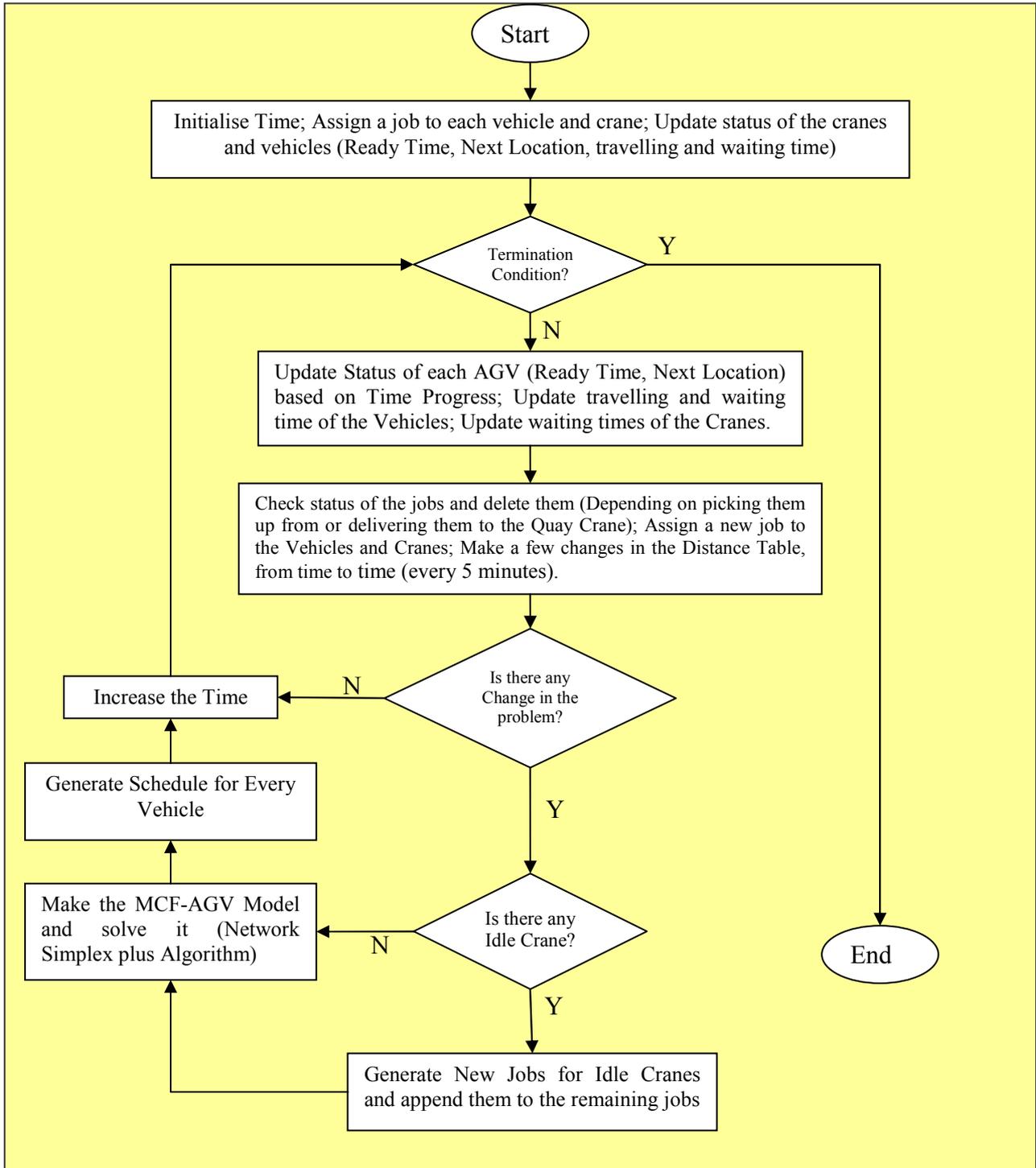


Figure 6-5: Operations of the software in dynamic aspect

One important question is remained, how many new jobs should be generated for any idle crane and what is the best situation for the problem? The answer of this question depends on the stability of the schedule and the software's performance. Generally, the first factor is related to any change in the problem and traffic in the routes such as congestion, collision, live-lock and deadlock. Since we assume that the vehicles are moving with an average speed so that there is no traffic problem, the answer to this question is determined by the rate of change in the problem and software's performance.

6.7 Experimental results from the dynamic aspect

In order to evaluate the result of Network Simplex plus Algorithm for the Scheduling problem of Automated Guided Vehicles in dynamic aspect, we did a simulation for six hours. In this simulation, the distance between every two points in the port as well as the source and destination of jobs were chosen randomly. During the simulation, the Job Generator generated 5 jobs for any idle crane. Other parameters were the same as Table 5-1.

We put some parts the simulation's results in Figures 6-6 and 6-7. Figure 6-6 shows the travelling and waiting times of the vehicles as well as the waiting times of the cranes.

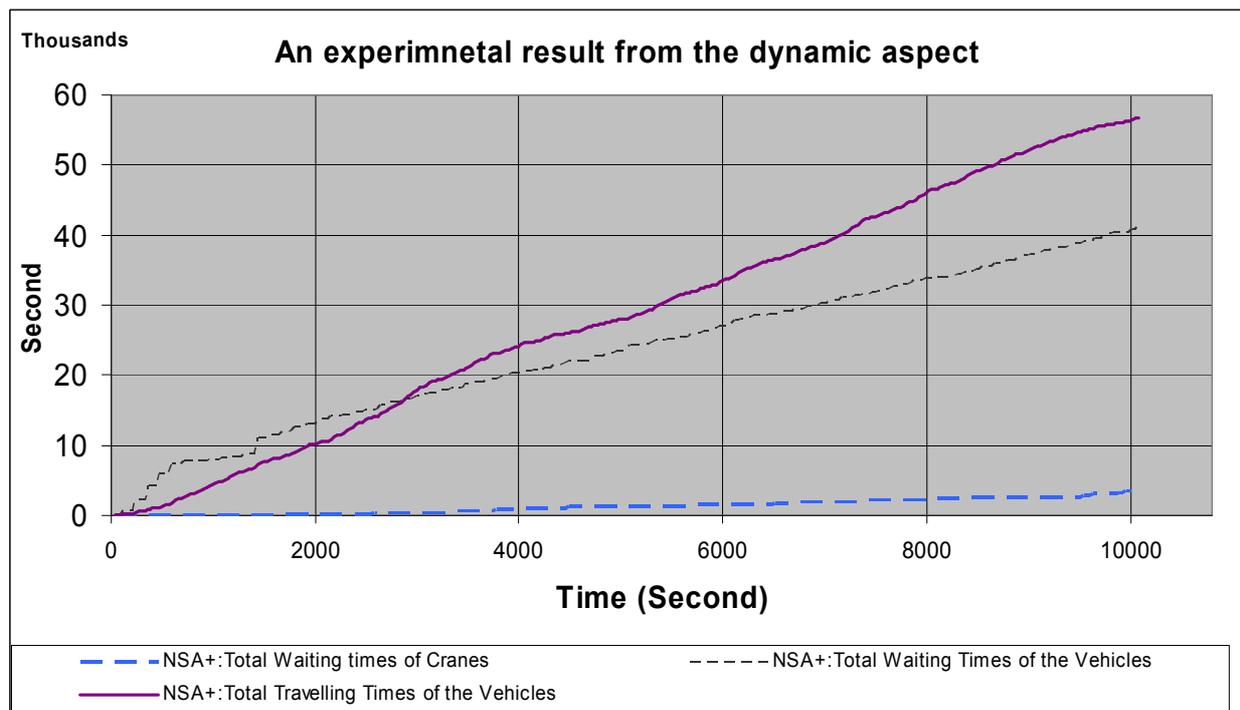


Figure 6-6: An experimental result from the dynamic scheduling problem of AGVs (NSA+ solved the problem).

Figure 6-7 shows three attributes of the carried jobs based on ‘Appointment time’ of jobs. These attributes are ‘QCraneTime’ (when the crane is ready to pick-up/drop-off the job from/on the vehicle), ‘VehicleTime’ (when the vehicle is ready to deliver/pick-up the job to/from the crane) and ‘ActualTime’ (when the job has been served).

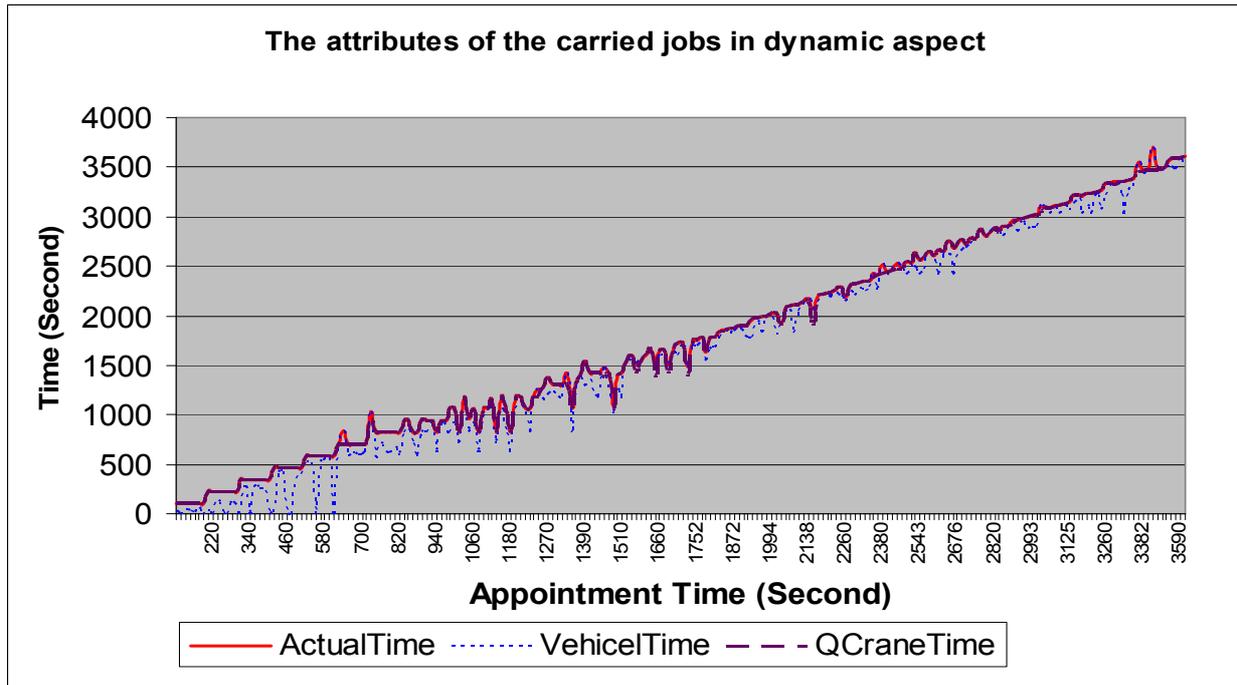


Figure 6-7: The attributes of the carried jobs in the dynamic scheduling problem of AGVs.

In this experiment, our observations were:

Observation 6-3: As we can see in Figure 6-6, the travelling times of the vehicles is significantly greater than their waiting times after 2,700 seconds. This indicator shows the vehicles were used efficiently to handle the container jobs. The waiting times of cranes are at a reasonable level. Since the cranes are a critical resource in the container terminal, their waiting times should be kept at minimum level.

Observation 6-4: In Figure 6-7, ‘ActualTime’ is the maximum of ‘QCraneTime’ and ‘VehicleTime’. If we draw a straight line between the left-down and the right-up corners of the figure, it can be seen that the ‘ActualTime’ has a good fitting with the ‘Appointment times’. Hence, the jobs were served efficiently.

Note that in the experiment, we assumed the distance table is in range of the time window of cranes (see Table 5-1). We did some changes in the values of Table 5-1 and ran the software. Our observations from that experiment were:

Observation 6-5: If the time window of cranes (e.g. 200 seconds) was significantly greater than the distance between every two points in the container terminal (e.g. 50 seconds, on average), then waiting times of the cranes would be less than our results. In this situation, the vehicles waited for the cranes more and therefore the jobs were served with more delay.

Observation 6-6: If the time window of cranes (e.g. 20 seconds) was significantly less than the distance between every two points in the container terminal (e.g. 200 seconds, on average), then waiting times of the cranes would be greater than our results. In this situation, the cranes waited for the vehicles more and therefore the jobs were served with more delay.

6.8 Summary and conclusion

In this chapter, some modifications were applied to NSA to have obtained a new version of the algorithm, Network Simplex plus Algorithm (NSA+). The main features of NSA+ deals with the entering arc and leaving arc. In order to find an entering arc, the algorithm uses a mixture of memory technique and heuristic method. Additionally, the leaving arc is chosen appropriately so that the spanning tree of the graph always becomes strongly feasible. NSA+ prevents cycling by this feature.

Then, the same static problems were solved by both algorithms NSA and NSA+, and CPU-Time required to solve the problems has been compared. Our experiments showed that NSA+ can solve the problems faster than NSA.

NSA+ is a complete and polynomial algorithm. We employed NSA+ to solve the dynamic scheduling problem of Automated Guided Vehicles in container terminal (defined in Chapter 4 and presented by the MCF-AGV). The result of a six-hour simulation showed the ‘Actual time’ of jobs, at which they have been handled by the vehicles and cranes, had a good fitting with their ‘Appointment times’. Based on our experiments, NSA+ is a practical algorithm for dynamic Automatic Vehicle Scheduling.

Chapter 7: Dynamic Network Simplex Algorithms and Dynamic Scheduling of AGVs

In this chapter, we extend Network Simplex Algorithm in dynamic aspect. In this aspect Dynamic Network Simplex Algorithm (DNSA) and Dynamic Network Simplex plus Algorithm (DNSA+) are presented. Then, NSA+ and DNSA+ are applied to the dynamic scheduling problem of Automated Guided Vehicles in container terminals (the problem defined in Chapter 4) and their results are compared.

7.1 Motivation

The objectives of Dynamic Network Simplex Algorithm are to solve the new problem faster, to use some parts of the previous solution for the next problem and to respond to changes in the problem. These objectives are explained below:

Firstly, although Network Simplex Algorithm is much faster than the traditional simplex algorithm for Linear Programs, for dynamic scheduling with large scale problems it still takes time to make a new MCF-AGV model and to solve it. The dynamic problem arises when new jobs are introduced, fulfilled jobs are removed, and the distance between the source and destination of the jobs are changed. The dynamic problems need more efficient algorithms.

Secondly, in most practical environments, scheduling is an ongoing reactive process where the presence of real time information continually forces reconsideration and revision of pre-established schedules. The second goal of DNSA is to repair the solution based on dynamic changes, rather than having to resolve it from scratch each time.

Thirdly, in many applications of graph algorithms, including communication networks, graphics, assembly planning, and scheduling, graphs are subject to discrete changes, such as additions or deletions of arcs or nodes. In the last decade there has been a growing interest in such dynamically changing graphs, and a whole body of algorithms and data structures for dynamic

graphs has been discovered [82]. In a typical dynamic graph problem one would like to respond to the changes in the graph that are under-going a sequence of updates, for instance, insertions and deletions of arcs and nodes.

7.2 Classification of graph algorithms

Given the powerful versatility of dynamic algorithms, it is not surprising that these algorithms and dynamic data structures are often more difficult to design and analyse than their static counterparts. Rauch (1992) classified dynamic graph problems according to the types of updates allowed [82]. A graph is said to be fully dynamic if the update operations include unrestricted insertions as well as deletions of arcs and nodes. A graph is called partially dynamic if only one type of update, either insertions or deletions, is allowed. If only insertions are allowed, the graph is called incremental; if only deletions are allowed it is called decremental. In this chapter our graph is fully dynamic.

7.3 The Dynamic Network Simplex Algorithm

In this section, Dynamic Network Simplex Algorithm with some examples is presented. The data structures of the problem and graph are basic components for the algorithm. Additionally, efficient memory management plays an important role for the algorithm. Before presenting details of the algorithm, the data structures and memory management are explained.

7.3.1 Data structures

The defined problem in Chapter 4 is considered to be solved by the algorithm. We formulated the problem and presented it as the MCF-AGV model (see Section 4.5). The MCF-AGV model was established on a directed graph. There are three dynamic data structures for the algorithm and problem. The memory is allocated for these structures based on the maximum number of jobs in the dynamic problem. These main structures are explained briefly below:

a) The first structure maintains the status of nodes in the graph model and its spanning tree. For each node we considered the Node number, Predecessor, first Child, Right sibling (next Child of the Predecessor), Left sibling (previous Child of the Predecessor), Balance (amount of supply or

demand of the node), Sub-tree's size, Basic arc of the node, Orientation of the Basic arc, Flow value of the Basic arc and Potential of the node. We explain these attributes with an example.

Figure 7-1 shows an example of the spanning tree [58, 2] for a small problem like Figure 4-5 when nodes 9 and 10 (the Job-Input and Job-Output nodes for Job 4) have been deleted. Given a graph $G = (N, A)$, let $T_t \subset A$ be a spanning tree in G at time t . The Root is identified with node '0'. Consider some node $v \in N - \{0\}$:

- There is a unique (undirected) path, denoted by $P(v)$, from v to the Root node '0'. The arc in $P(v)$, which is incident to v , is called the Basic arc of v .
- The Orientation of the Basic arc is called Upward (Downward) if v is the tail (head) node of its Basic arc.
- The other terminal node u of the Basic arc is called the Predecessor (node) of v . If v is the Predecessor of some other node u , we call u a Child (node) of v .
- The number of nodes in the sub-tree, rooted by v , including itself, is called the Sub-tree size of v .
- Every node may have a Right and/or Left sibling, but it has at most one Child reference. The other children of the node are accessible by traversing the Siblings.
- The Sub-tree's size and Predecessor variables are used to find a cycle and pivoting. The Orientation, Child, and Sibling variables are used for the computation of the node Potentials. (see the main loop in Figure 5-2)

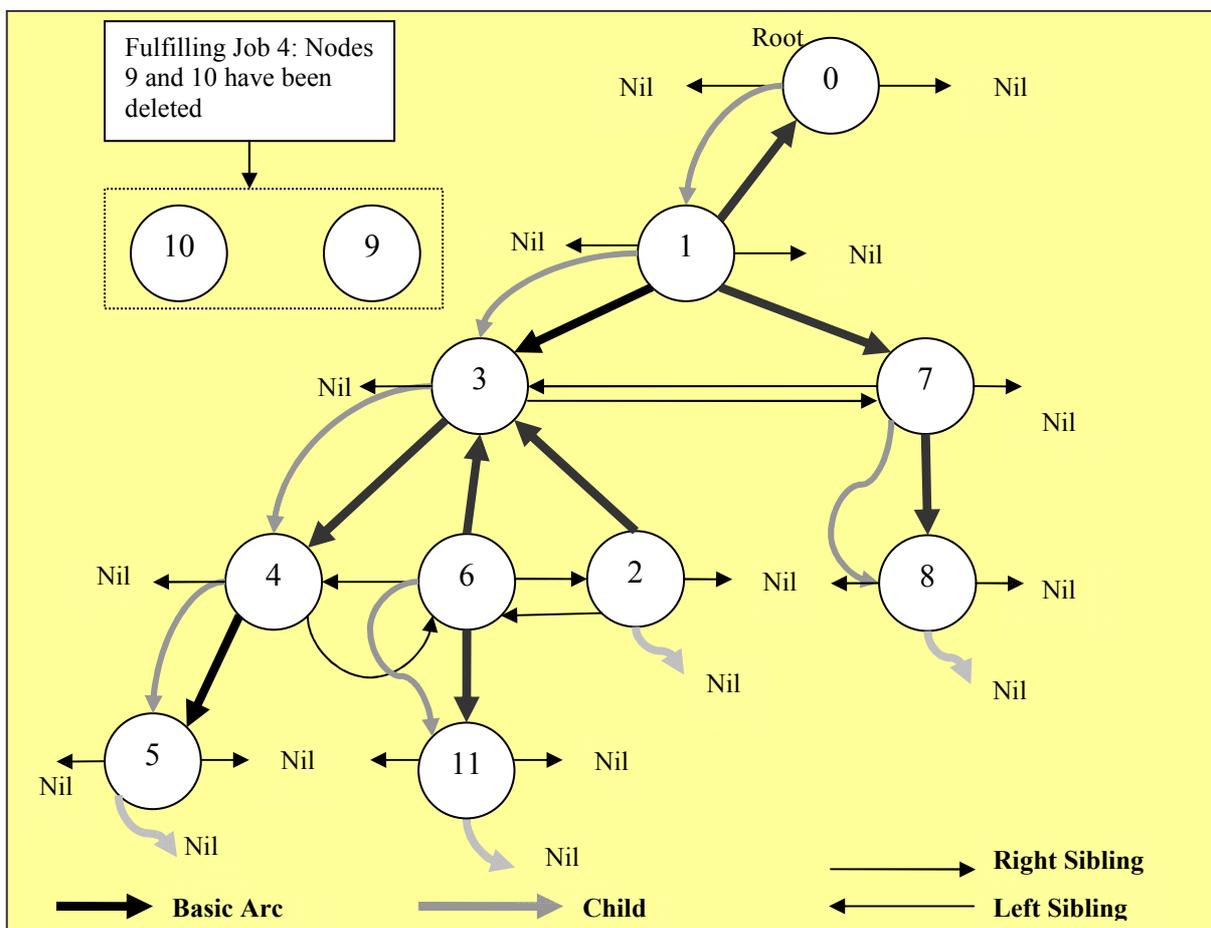
The Predecessor, Child, Left sibling, Right sibling, Sub-tree's size, Basic arc of each node, Orientation of the Basic arc are shown in a table, below Figure 7-1. For the status of the nodes, we introduce the following property.

Property 7-1: Every node has an Identification flag. At any time, the Identification of a node specifies whether the node belongs to the model or not. There are two cases for the Identification of nodes, 'FIXED' and 'UNFIXED'. At each stage of the dynamic problem, the 'FIXED' nodes are considered by the algorithm whereas the 'UNFIXED' nodes are ignored. We introduce the following notations for these sets:

FN_t : The set of 'FIXED' nodes of the current graph model at time t .

DN_t : The set of 'UNFIXED' nodes after repairing the solution at time t .

At each stage of the dynamic problem, a few existing jobs are fulfilled and a few new jobs are arrived. Based on the fulfilled jobs, a set of nodes for deletion is collected (we call it the ‘DELETION’ nodes). The elements of this set are a couple of nodes associated with every fulfilled job (we called them the Job-Input and Job-Output nodes; see Section 4.5). The nodes of this set have to be removed from the graph model in the next stage. Additionally, when a new job arrives, a set of new nodes associated with the job are collected (we call it the ‘INSERTION’ nodes). These nodes have to be inserted into the graph model in the next stage of the dynamic problem.



Node number	0	1	2	3	4	5	6	7	8	9	10	11
Predecessor	Nil	0	3	1	3	4	3	1	7	-	-	8
Child	1	3	Nil	4	5	Nil	11	8	Nil	-	-	Nil
Right sibling	Nil	Nil	7	7	6	Nil	2	Nil	Nil	-	-	Nil
Left sibling	Nil	Nil	6	Nil	Nil	Nil	4	3	Nil	-	-	Nil
Sub-tree' size	9	8	1	6	2	1	2	2	1	-	-	1
Orientation	-	Up	Up	Down	Down	Down	Up	Down	Down	-	-	Down
Identification	FIX	FIX	FIX	FIX	FIX	FIX	FIX	FIX	FIX	UFD	UFD	FIX

Figure 7-1: A sample of the spanning tree and its attributes (FIX='FIXED', UFD='UNFIXED').

When a node is removed from the graph model, the arcs associated with the node are marked as the ‘DELETION’ arc (we use the notation D to show these arcs after repairing the solution). When a node must be inserted into the model, the arcs associated with the node are marked as the ‘INSERTION’ arc.

b) The second data structure is considered for arcs in the MCF-AV model, including the Tail node, Head node, Lower bound, Upper bound, Cost and Value of the arcs. For the status of the arcs, we introduce another property below.

Property 7-2: Every arc has an Identification flag. The Identification of an arc specifies the arc is in which set of the spanning tree structure. There are four cases for the Identification of an arc at time t ; the arc is in the T_t set, the L_t set, the U_t set (according to the spanning tree structure (T , L , U); see Definition 5-1 in Section 5.2.1) or in the D_t set.

Suppose that the paths in the solution for the problem in Figure 4-5 are $1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 11$ and $2 \rightarrow 7 \rightarrow 8 \rightarrow 11$. According to Figure 7-1 for the solution, those sets at time t are as follows:

$$T_t = \{(1,0), (1,3), (3, 4), (4,5), (6,3), (6,11), (2,3), (1,7), (7,8)\}$$

$$L_t = \{(1,5), (2,5), (1,11), (2,11), (4,7), (4,11), (6,7), (5,6), (8,3), (8,5), (2,0), (0,3), (4,0), (0,5), (6,0), (0,7), (8,0), (0,11)\}$$

$$U_t = \{(2, 7), (8,11)\}$$

$$D_t = \{(1,9), (9,10), (2,9), (10,3), (10,5), (10, 7), (10, 11), (4, 9), (6,9), (8,9), (0,9), (10,0) \}$$

Note that the flow on every Basic arc in the spanning tree is between the Lower bound and the Upper bound of the arc. The flow of every arc in the set L is at the Lower bound of the arc. The flow of every arc in the set U is at the Upper bound of the arc. Moreover, we considered the Artificial arcs that connect the Root to the other nodes in the sets. These Artificial arcs were explained in Section 5.2.2 (Step 0).

c) The third data structure is a Job Buffer. There is a direct mapping between the Job-Input and Job-Output nodes in the MCF-AGV model and a particular location in the buffer for every job. For example, the nodes 3 and 4 in Figure 7-1 are associated with the first location in the Job Buffer. When a job is fulfilled, its location in the Job Buffer is marked as empty or hole. According to Figure 7-1, we have a hole in the Job Buffer. The nodes 9 and 10 associated with the job 4 were the ‘DELETION’ nodes in the MCF-AGV model. When a new job is arrived, it is put into a hole of the Job Buffer.

7.3.2 Memory management

A small memory management facility has been designed, implemented and embedded in the software. The objectives of this facility are to make independent software, to get a higher performance (most programming languages, including C/C++, have the Garbage Collection facility in dynamic memory allocation; It has negative impacts on the efficiency) and prevent any missing job (when the Job Generator generates a job and the memory could not be allocated).

There are two aspects of memory management in the software. The first one is relevant to the jobs whereas the second one refers to the graph model. There is a buffer for the jobs, which is allocated at the start of operation. Once a job is fulfilled, a hole will be created in the buffer and when the Job Generator generates a job, it puts the job into the first hole. For the arcs and nodes in the graph model, an Identification flag has been considered. The Identification flag associated with each arc identifies whether the arc is in the T_t set, L_t set, U_t set, or D_t set (see Property 7-2) at time t . There is the one-to-one mapping between every location in the Job Buffer and the nodes associated with the job in the graph model. When a job is fulfilled, the nodes associated with this job are marked for 'DELETION'. For each node belonged to the fulfilled jobs, the node and the relevant arcs are removed from the spanning tree of the graph. In order to make a new spanning tree, we use a '**Remove-Node-Algorithm**', which will be presented in the next section. When a new job arrives the relevant nodes, which has been deleted from the graph model, will be marked for 'INSERTION'. The 'INSERTION' nodes and the arcs associated with the new jobs are inserted into the spanning tree consistently. This task is performed by '**Insert-Node-Algorithm**', which will be presented later in the next section.

As stated before (Section 4.5 in Chapter 4), given N jobs and M AGVs in the problem, there are $M+2 \times N+1$ nodes and $M+M \times N+N \times (N-1)+2 \times N$ arcs in the MCF-AGV model. The challenge here is to control them correctly. The memory management routine allocates the memory based on the maximum number of jobs. This parameter is determined by the user and here is represented as MNJ . Table 7-1 shows a memory map of the allocated space. There were four different types of arcs in the MCF-AGV model: Inward Arcs, Outward Arcs, Auxiliary Arcs, and Intermediate Arcs (see Figure 4-5). Additionally, we needed the Artificial Arcs to generate initial Basic Feasible Solution (see 'Step 0' in Section 5.2.2). Two blocks of the memory are allocated for these arcs and two pointers are used to access them; the first one is for arcs in the MCF-AGV

model and the second one is for the Artificial Arcs. In order to address on a certain type of arc, it is needed to have an offset. The offset is the difference in the address from the beginning of the block.

Table 7-1: Memory allocation for the arcs of the MCF-AGV model and its algorithm

Type of Arcs	Specification	Offset	Size (the number of arcs)	Example for 2 AGVs and 2 Jobs (See Figure 5-7)
ARC_{inward}	Arcs from every vehicle node to Job-Input nodes	0	$M \times MNJ$	(1,3);(1,5);(2,3);(2,5)
$ARC_{outward}$	Arcs from every vehicle node to the sink	$M \times MNJ$	M	(1,7);(2,7)
	Arcs from every Job-Output node to the sink	$M \times MNJ + M$	MNJ	(4,7);(6,7)
$ARC_{auxiliary}$	Arcs from every Job-Input node to its Job-Output node	$M \times MNJ + M + MNJ$	MNJ	(3,4);(5,6)
$ARC_{intermediate}$	Arcs from every Job-Output node to other Job-Input node	$M \times MNJ + M + MNJ + MNJ$	$MNJ \times (MNJ - 1)$	(4,5);(6,3)
$ARC_{artificial}$	Artificial Arcs to generate initial feasible solution	0	$2 \times MNJ + M + 1$	(1,0);(2,0);(0,3); (4,0);(0,5);(6,0);(0,7)

7.3.3 The algorithms DNSA and DNSA+

We base the Dynamic Network Simplex Algorithms on the Network Simplex Algorithm. DNSA is a standard dynamic form of NSA and DNSA+ is DNSA with the features of NSA+ (see Section 6.2). The inputs of the dynamic algorithms are:

- **s**: Stage for the dynamic problem, which is increased by the algorithm.
- **Set of ‘DELETION’ Nodes**: it determines which nodes have to be removed from the model.
- **Set of ‘INSERTION’ Nodes**: it determines which nodes have to put into the new model.

Figure 7-2 shows the Dynamic Network Simplex Algorithm. At the beginning, when the container or Job buffer became full and the software made a MCF-AGV model, an initial feasible solution is generated by the ‘Generate-Initial BFS’ procedure. The operation of this procedure was described in Section 5.2.2 (Step 0). In fact, an initial feasible spanning tree solution (T_0, L_0, U_0) is created (see Definition 5-1). The difference between NSA and DNSA is the ‘Reconstruct New BFS’. When the ‘s’ is zero, the ‘Generate Initial BFS’ is called. Otherwise, the ‘Reconstruct New BFS’ procedure repairs the current solution and spanning tree at time t; (T_t, L_t, U_t) is reconstructed. The main body of the algorithms, NSA and DNSA, are the same. The operation of the main body was described in Section 5.2.2. Here, we describe the ‘Reconstruct New BFS’.

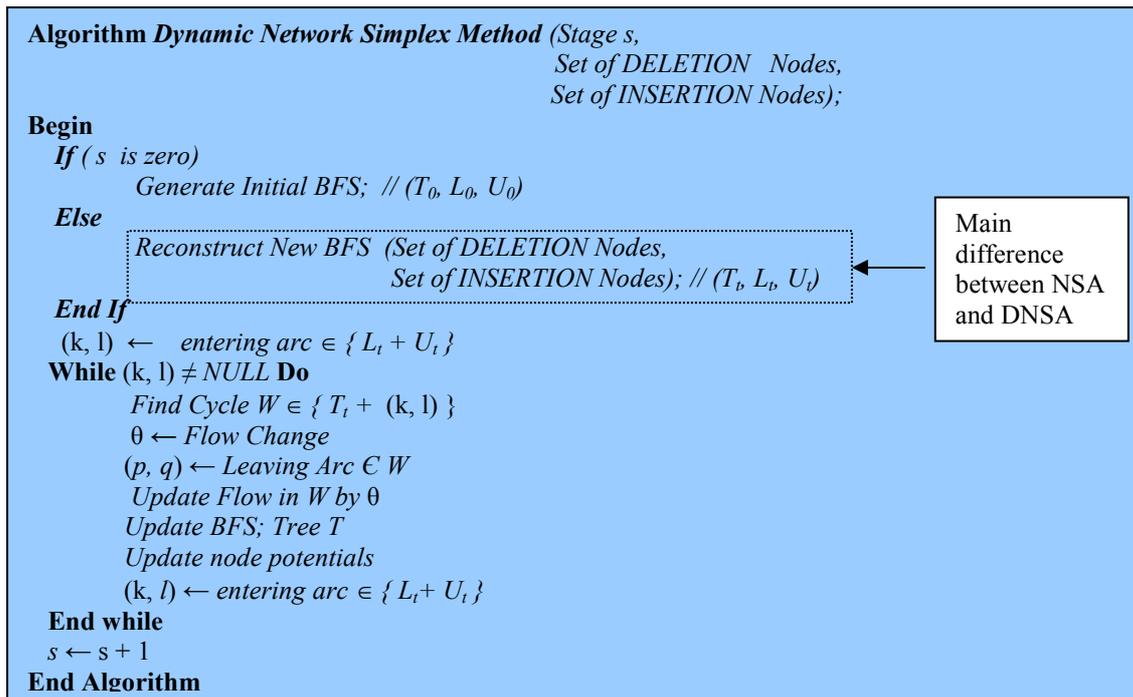


Figure 7-2: The Dynamic Network Simplex Algorithm.

Figure 7-3 shows the algorithm of reconstructing the new spanning tree. There are three main steps in the algorithm; Step 01, Step 02 and Step 03.

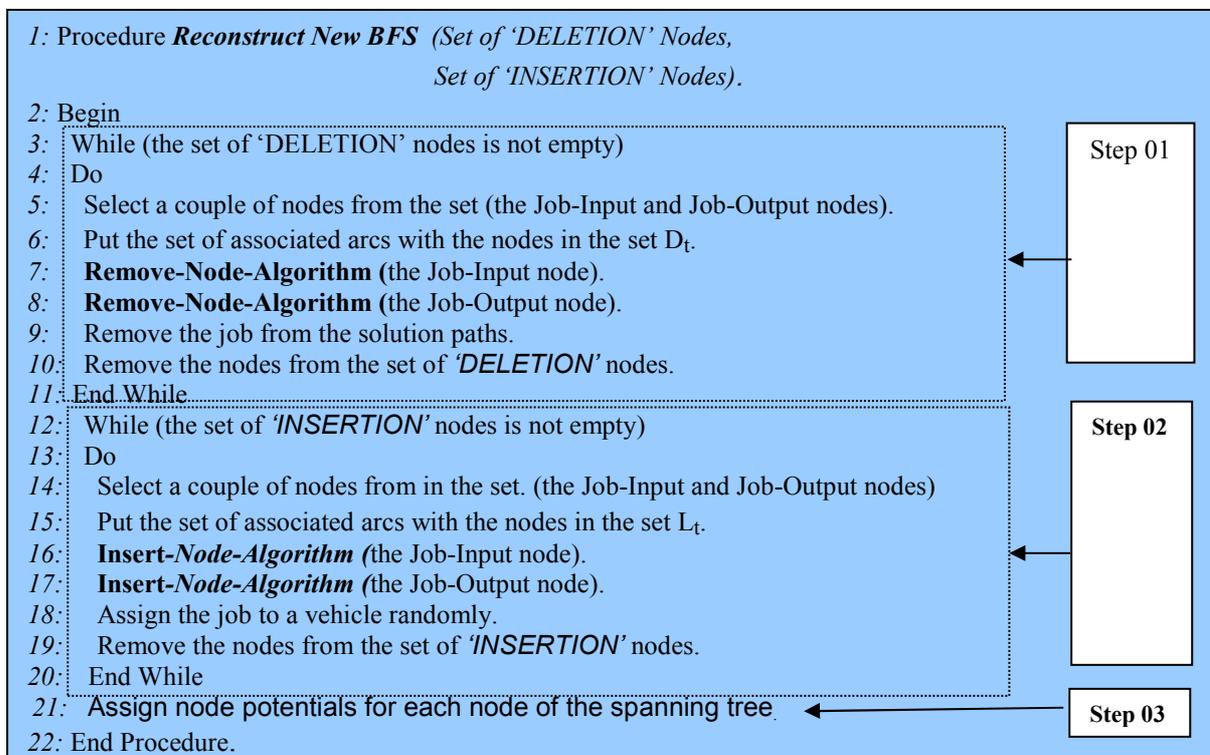


Figure 7-3: The pseudo code of reconstructing the spanning tree in Dynamic Network Simplex Algorithm.

In Step 01, all ‘DELETION’ nodes and their arcs are removed from the model, spanning tree of the graph and the solution paths. After that, the ‘INSERTION’ nodes and their arcs are put into the model, its spanning tree and solution in the second step (Step 02). In Step 03, according to the current solution a value is assigned to the potential of each node in the new spanning tree. There is no challenge in Step 03 since it is easy task. The Steps 01 and 02 are elaborated by some examples as follows:

Step 01: There is a loop for this step. At first, a couple of nodes associated with every fulfilled job (from the ‘DELETION’ set) are selected and transferred into the D_t set. These two tasks are performed in Lines 5 and 6, respectively. Then, in Lines 7 and 8 a procedure, which called ‘**Remove-Node-Algorithm**’, is used to remove the nodes from the spanning tree consistently. After that in Line 9, the fulfilled job associated with the nodes is removed from the solution paths. Based on removing the job from the solution, some arcs may be transferred into the set L_t or U_t .

Figure 7-4 shows the operation of the ‘**Remove-Node-Algorithm**’. Removing a node from the spanning tree, splits T_t into several clusters, say T_1, T_2, \dots and so on. Depending on whether the deleted node has a Child or not, there is a branch in the algorithm. Based on the location of the deleted node in the spanning tree, appropriate operations are done.

```

1: Procedure Remove-Node-Algorithm (Node)
2: Begin
3:   If (the Node has not any Child)
4:     Set the Child of the Predecessor of node to Nil (if its parent had only one Child).
5:     Set the Right sibling, Left sibling of the other nodes if it is necessary.
6:     Set the Sub-tree's size of the new spanning Tree.
7:   Else
8:     Set the Right sibling, Left sibling of the other Nodes.
9:     Find the last Child of the root.
10:    Set the Sub-trees (Children of the deleted node) as the new Children for the root.
11:    Set a Basic-arc for every root-node of the sub-trees using Artificial arcs.
12:    Set Predecessor, Sub-tree's size of the nodes in the new spanning tree.
13:  End If
14: End Procedure.

```

Figure 7-4: The pseudo code of removing a node from the spanning tree in Dynamic Network Simplex Algorithm.

If the ‘DELETION’ node has not any Child (Line 3) and its Predecessor has not any other Child (Line 4), then the Child of the Predecessor is set to Nil. After that in Lines 5 and 6, the Right and left siblings of other nodes are adjusted and the Sub-tree’s size of the new spanning tree is updated. If the ‘DELETION’ node has a Child, the Right and left siblings of other nodes are

adjusted in Line 8. In Lines 9 and 10, the last Child of the Root is found out and the sub-trees (Children of the ‘DELETION’ node) are connected to the root with the Artificial arcs. Then in Lines 11 and 12, the Basic arc of the root in the sub-trees, and their Predecessor as well as the sub-tree’s size is adjusted. Some examples for a ‘DELETION’ node are demonstrated below:

Example 7-1: Suppose that the job associated with nodes 7 and 8 is fulfilled. Imagine the node 8 in Figure 7-1 must be deleted, first. In this case, the ‘DELETION’ node has not any Child, Right sibling and Left sibling. In this case T1 is the rooted spanning tree and T2 is empty. What is necessary to do is to delete the Child of its Predecessor and then update the Sub-tree’s size from the Predecessor of the deleted-node to the Root. The spanning tree after removing node 8 is shown in Figure 7-5. The same operation is done for deleting the node 7.

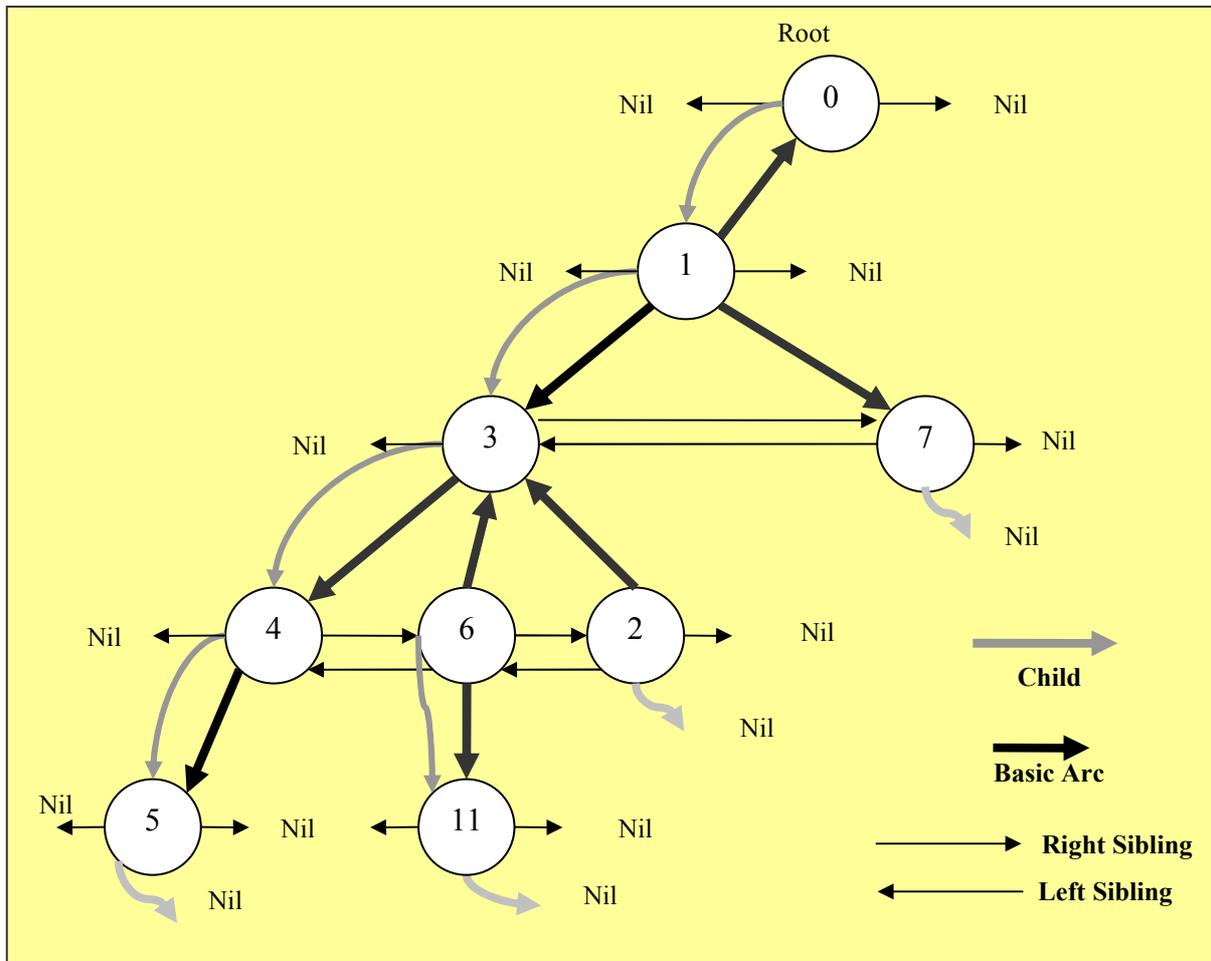


Figure 7-5: The new spanning tree after removing nodes 8 (See Figure 7-1).

After these operations the solution paths are $1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 11$ and $2 \rightarrow 11$. According to Property 7-1, the sets of nodes in the graph at time t are:

$$FN_t = \{1, 2, 3, 4, 5, 6, 11\}$$

$$DN_t = \{9, 10, 7, 8\}$$

According to Property 7-2, the sets of arcs in the current graph are:

$$T_t = \{(1,0), (1,3), (3,4), (4,5), (6,3), (6,11), (2,3)\}$$

$$L_t = \{(1,5), (2,5), (1,11), (4,7), (4,11), (5,6), (2,0), (0,3), (4,0), (0,5), (6,0)\}$$

$$U_t = \{(2,11)\}$$

$$D_t = \{(1,9), (9,10), (2,9), (10,3), (10,5), (10,7), (10,11), (4,9), (6,9),$$

$$(8,9), (0,9), (10,0), (0,11), (1,7), (2,7), (7,8), (0,7), (6,7), (7,8), (8,11), (8,0), (8,3), (8,5)\}$$

Example 7-2: Suppose that the job associated with nodes 3 and 4 is fulfilled in Figure 7-1. Imagine the node 3 is deleted first. The spanning tree after removing node 3 and the reconstruction operation is shown in Figure 7-6.

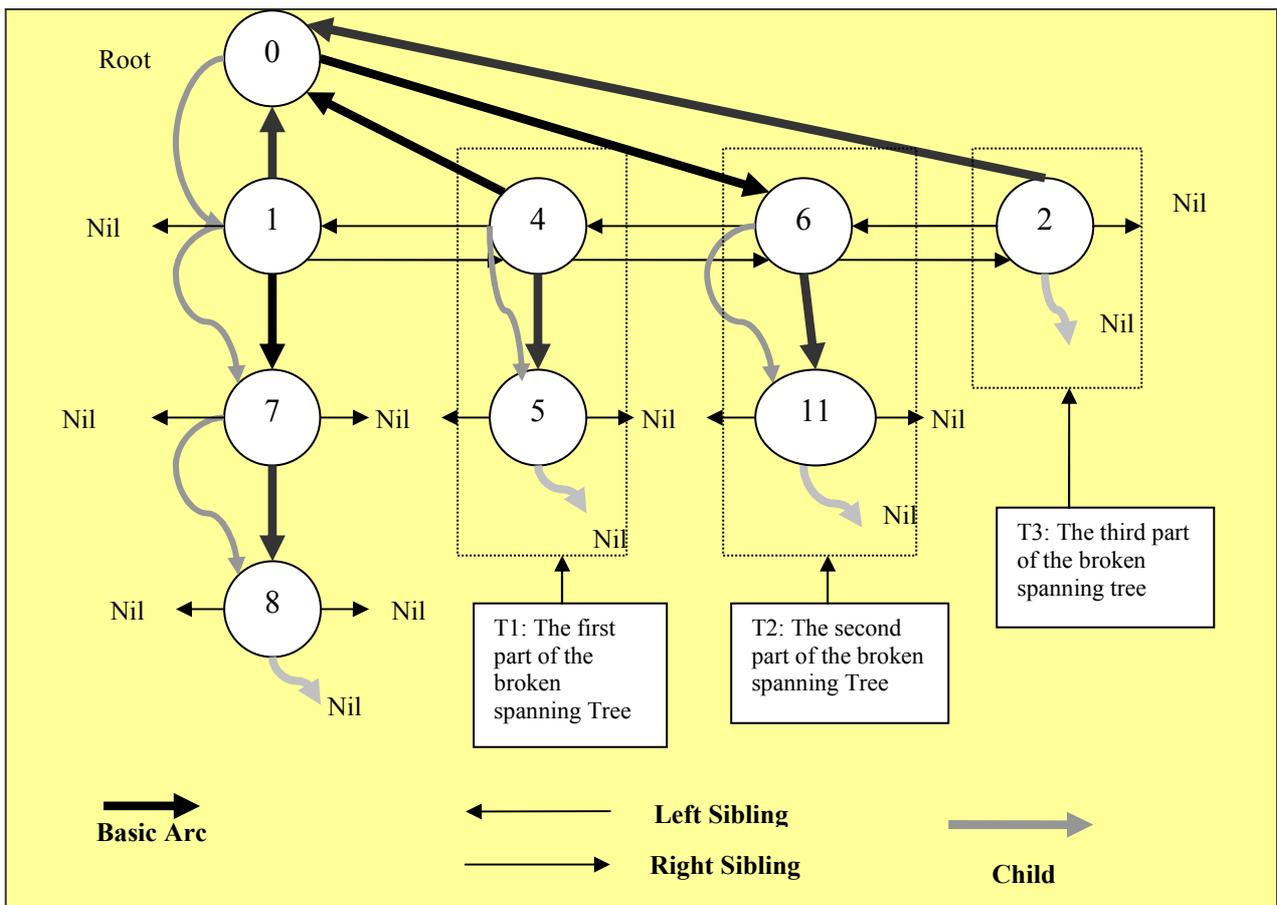


Figure 7-6: The new spanning tree after removing node 3 (See Figure 7-1).

In this case, the ‘DELETION’ node had a Child and its Right sibling exists. The following operations were necessary for this case:

- Adjust the Child of node 1 to node 7.
- Connect the sub-trees T1 (node 4 and 5), T2 (node 6 and 11) and T3 (node 2) to the Root.
- Adjust the Right and Left siblings from the most left side (node 1) to the most right side (node 2).
- Recalculate the Sub-tree size for the node 1 and Root.

Note that, the best and fastest way to recover the spanning tree is to connect the minor fragmented sub-trees to the Root, in our experience. We used the artificial arcs for reconnecting T1, T2 and T3 to the Root or main part of the spanning tree. The Orientation of the Artificial-Basic arc depends on the amount of supply/demand of the node. For the node j , if j is a Job-Output node we include $(j, 0)$ in T_t . If j is a Job-Input node, we include arc $(0, j)$ in T_t .

After deleting the node 3, the node 4 in Figure 7-6 must be deleted. The spanning tree after removing node 4 and the reconstruction operation is shown in Figure 7-7.

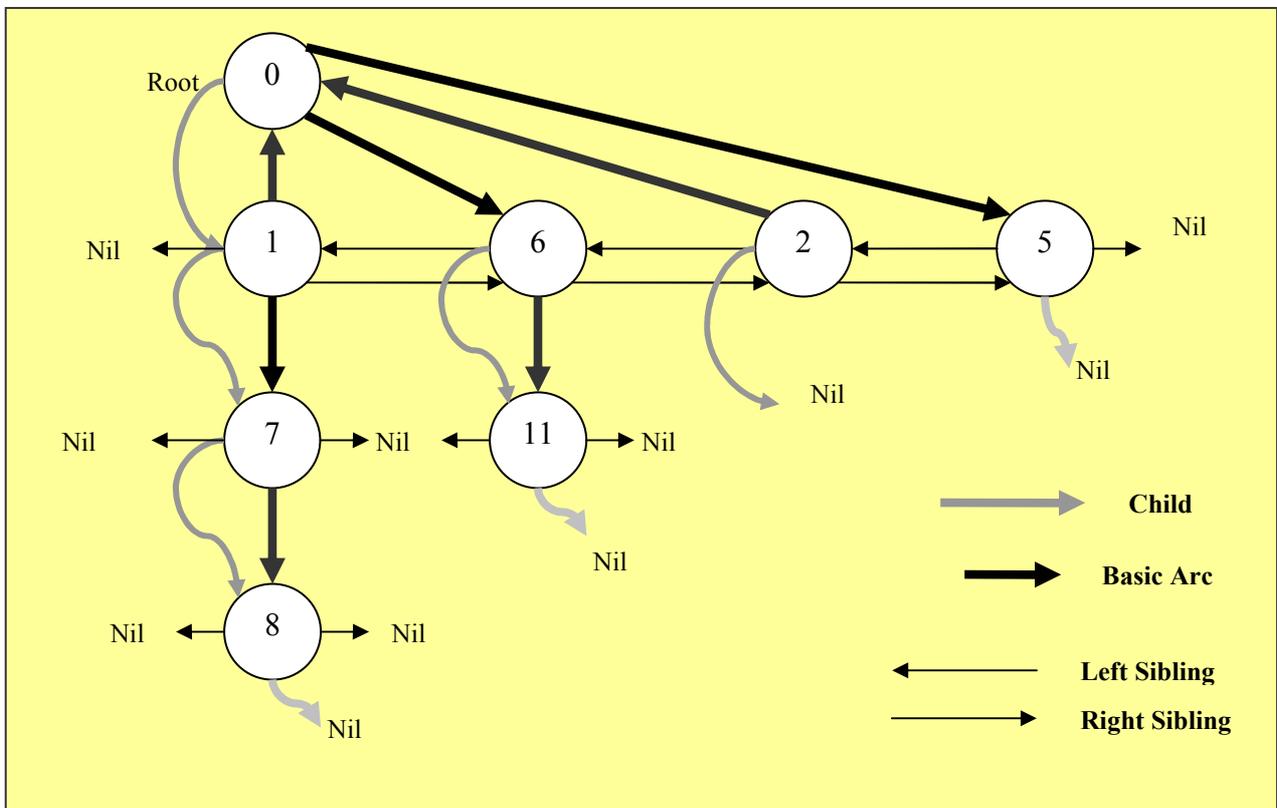


Figure 7-7: The new spanning tree after removing node 4 (See Figure 7-6).

After these operations the solution paths are $1 \rightarrow 5 \rightarrow 6 \rightarrow 11$ and $2 \rightarrow 7 \rightarrow 8 \rightarrow 11$. According to Property 7-1, the sets of nodes in the graph at time t are:

$$FN_t = \{1, 2, 5, 6, 7, 8, 11\}$$

$$DN_t = \{9, 10, 3, 4\}$$

According to Property 7-2, the sets of arcs in the graph are:

$$T_t = \{(1,0), (6,11), (1,7), (7,8), (2,0), (0,5), (6,0)\}$$

$$L_t = \{(2,5), (1,11), (2,11), (4,7), (4,11), (6,7), (5,6), (8,3), (8,5), (0,7), (8,0), (0,11)\}$$

$$U_t = \{(1,5), (2,7), (8,11)\}$$

$$D_t = \{(1,9), (9,10), (2,9), (10,3), (10,5), (10,7), (10,11), (4,9), (6,9), (8,9), (0,9), (10,0), (1,3), (3,4), (4,5), (6,3), (0,3), (4,0), (2,3)\}$$

Step 02: In this step every new job is inserted into the spanning tree and the solution paths. At first, a couple of nodes associated with a new job (from the ‘INSERTION’ set) are selected and transferred into the L_t set. Then, a procedure, which called ‘**Insert-Node-Algorithm**’, is used to insert the nodes into the spanning tree. After that, the new job associated with the nodes is assigned to a vehicle randomly. This job is inserted into a solution path. Based on the insertion, some arcs may be transferred into the set L_t or U_t . This process is repeated for each new job.

Figure 7-8 shows the operations of the ‘**Insert-Node-Algorithm**’. The input of the algorithm is a node, which is appended to the new spanning tree by an Artificial arc. The attributes of these arc is the same as the Artificial arcs in the Basic Feasible Solution (see Step 0 in Figure 5-2). Firstly, the most Right sibling of the Root’s Child is found and the new node is put at the right side of the existing Children of the Root. These operations are performed in Lines 3-5. Then in Line 6, the Basic-arc, Predecessor, Child, Right-sibling, Left-sibling and Sub-tree’s size of this node are adjusted.

```

1: Procedure Insert-Node-Algorithm (Node)
2: Begin
3:   Find the Child of the root.
4:   Find the most Right sibling of the Child.
5:   Set the node as a new Child for the Root by an Artificial arc.
6:   Set Basic-arc, Predecessor, Child, Right-sibling, Left-sibling and Sub-tree’s size for this node and the
   root.
7: End Procedure.

```

Figure 7-8: The pseudo code of inserting a node into the spanning tree in Dynamic Network Simplex Algorithm

Example 7-3: Suppose that node 9 and 10 have to be inserted into the spanning tree of Figure 7-1. The spanning tree after the insertion is demonstrated by Figure 7-9. According to the algorithm in Figure 7-8, the Artificial arcs connect the nodes to the spanning tree. In this example, we assumed that the new job is inserted in the second path for AGV 2.

After these operations the solution paths are $1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 11$ and $2 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 11$.

Now, the sets of nodes in the graph at time t are:

$$FN_t = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$$

$$DN_t = \{\}$$

According to Property 7-2, the sets of arcs in the current graph are:

$$T_t = \{(1,0), (1,3), (3, 4), (4,5), (6,3), (6,11), (2,3), (1,7), (7,8), (0,9), (10,0)\}$$

$$L_t = \{(1,5), (2,5), (1,11), (2,11), (4,7), (4,11), (5,6), (2,0), (0,3), (4,0), (0,5), (6,0), (10,3), (10,5), (10,7), (0,7), (6,7), (7,8), (8,11), (8,0), (8,3), (8,5), (1,9), (2,9), (4, 9), (6,9)\}$$

$$U_t = \{(2,7), (8,9), (9,10), (10, 11)\}$$

$$D_t = \{\}$$

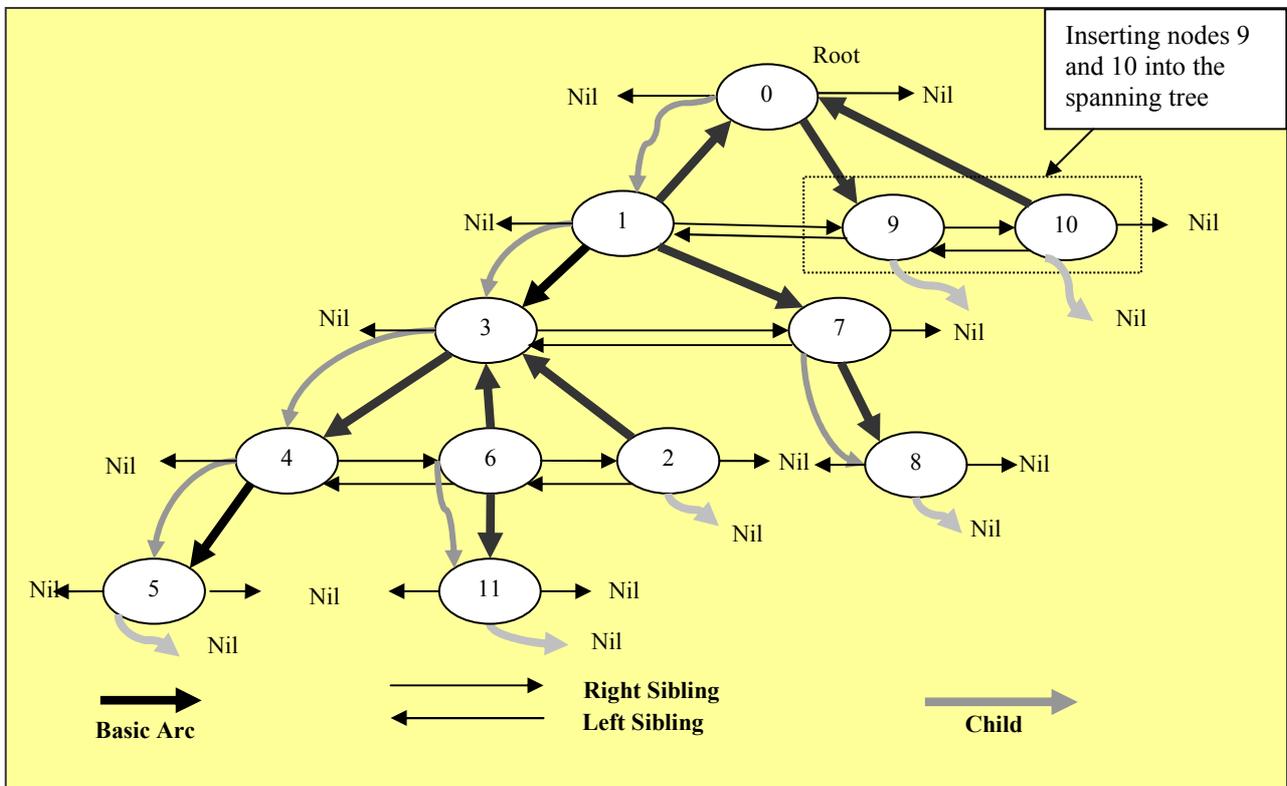


Figure 7-9: The new spanning tree after inserting node 9 and 10 (See Figure 7-1).

7.4 Software architecture for dynamic aspect

At the start of the process, a few jobs are generated for each crane and the memory for the jobs and graph are allocated. Then, the MCF-AGV model is made and tackled by Network Simplex plus Algorithm. The output of this algorithm is a few job sequences for the vehicles. Based on these sequences, the software will prepare a job list for each vehicle.

The main architecture of the software is demonstrated by Figure 7-10 for dynamic aspect. Note that this architecture is for the time when 's' > 0 (see the algorithm in Figure 7-2).

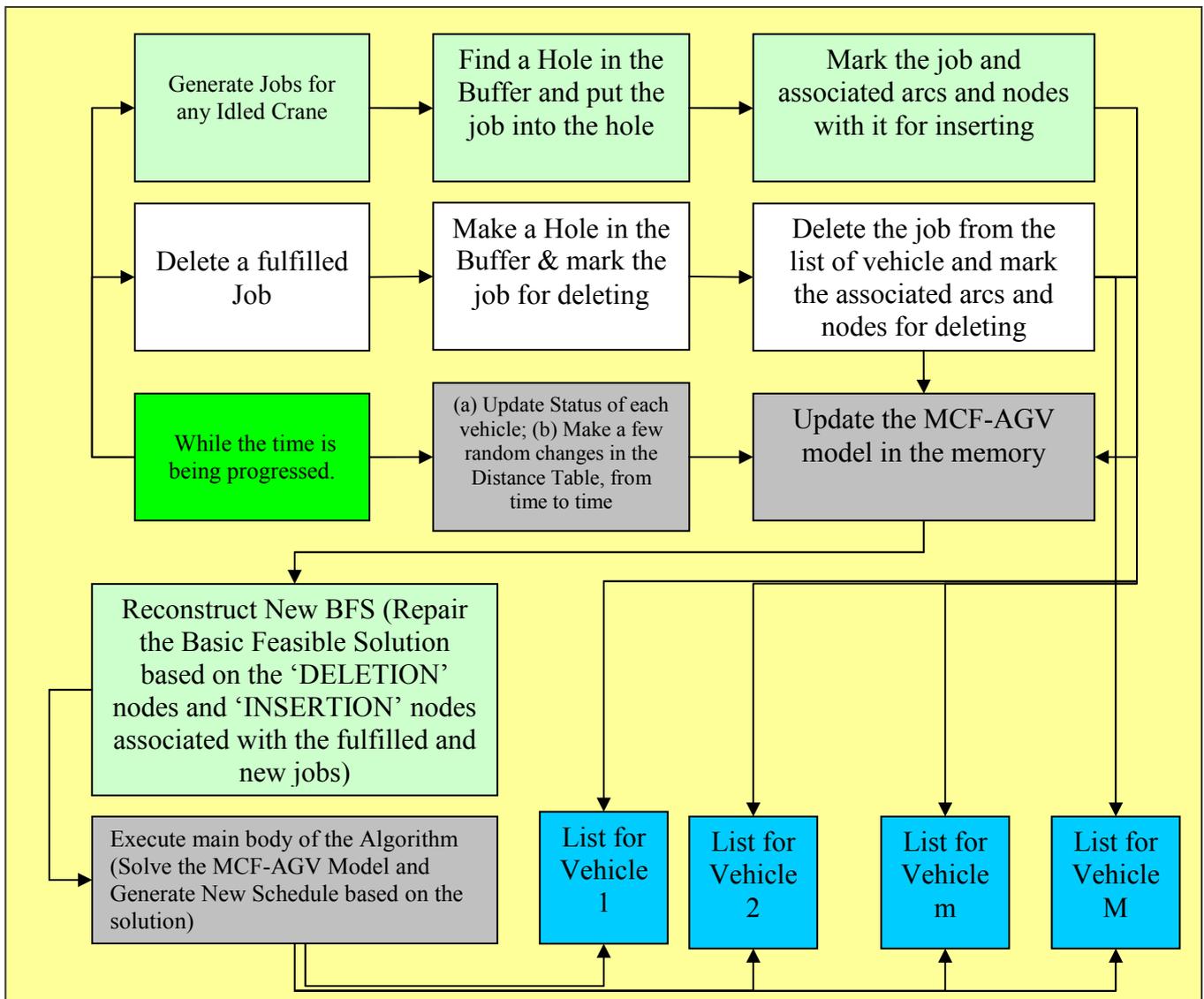


Figure 7-10: Block diagram of the software and algorithm (DNSA+) in the dynamic aspect

While the time is being progressed, the vehicles and cranes are carrying and handling the containers. From time to time, the software makes a few random changes in the distance table (see Table 4-1) in order to produce dynamic problems. The Job Generator has to generate a few new jobs, when it finds out any crane is in idle state.

As we see in the figure, every event is recorded in order to be processed latter. The events include modification of the vehicle's position, the fulfilled jobs and new jobs, and any change in the distance table. As we mentioned (see Section 7.3.2), a hole will be created in the Job Buffer when a job is fulfilled. After the Job Generator generates a job, it puts the job into a hole of the buffer. The software marks the nodes and arcs associated with the fulfilled and new jobs. The most important events that affect the spanning tree are the fulfilled and new jobs. The fulfilled jobs are removed from the list of vehicles and model whereas the new jobs are appended to remaining jobs and inserted into the model. Note that any change in the problem, without any fulfilled or new job, doesn't affect the spanning tree. In this case, only body of the algorithm is executed and finds out the optimal solution.

The software processes the recorded events and updates the MCF-AGV model. After removing the nodes and arcs (associated with the fulfilled jobs) from the model and omitting the jobs from the vehicle's lists, a new spanning tree is made. Next, the nodes and arcs associated with the new jobs are put into the new model and then the spanning tree is repaired. These jobs are assigned to one or more vehicles, randomly. These two tasks are made by 'Reconstruct New BFS'. After repairing the spanning tree, the main body the algorithm is executed and it finds out the optimal solution. Note that these tasks are non-preemptive, i.e. when a task starts execution on the processor, it finishes to its completion.

7.5 A comparison between DNSA+ and NSA+

To test and compare the performance of the algorithms, many jobs in dynamic fashion have been generated. Their sources, destinations and the distance between every two points in the port have been chosen randomly. During three hours simulation, about 90 problems with a condition of generating 5 jobs for any idle crane have been solved by DNSA+ and NSA+H. In these samples we assumed that there were 50 AGVs and 7 cranes in the port (see Table 5-1). It was very difficult to isolate the CPU-Times required to tackle the problems by the algorithms and the

CPU-Time required for memory management. Hence, we considered the number of iterations as an indicator to compare the algorithms. The numbers of iterations required to solving the problems have been drawn by Figure 7-11. A sample was collected every time when there were changes in the problem and the algorithms had to solve the new problem.

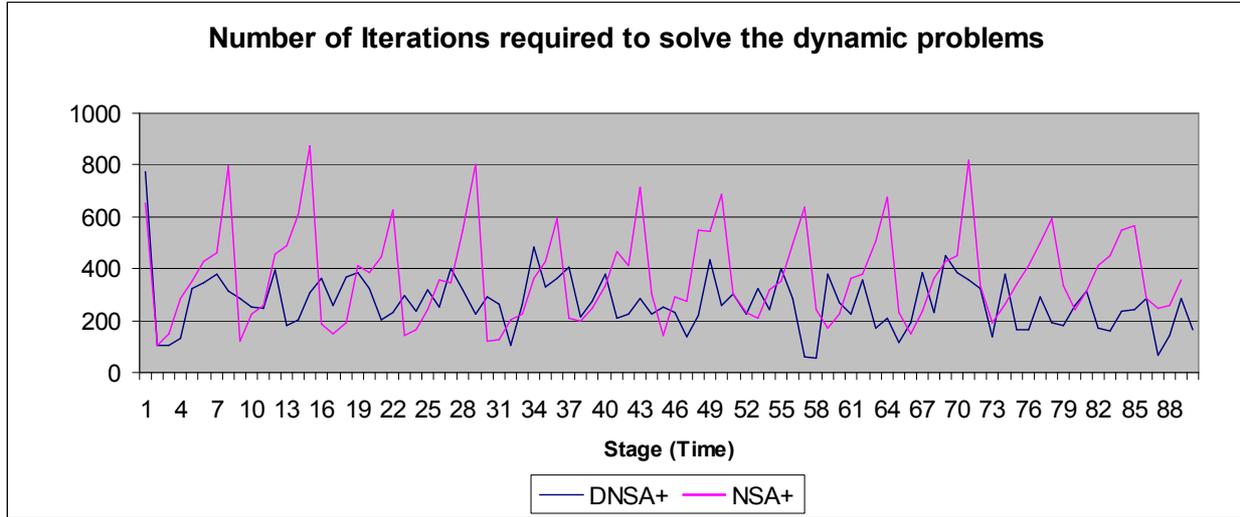


Figure 7-11: A comparison of the number of iterations in DNSA+ and NSA+

From Figure 7-11, we can observe that:

Observation 7-1: As we can see in the figure, the number of iterations in DNSA+ greatly has been decreased compared with NSA+. Therefore, the average number of iterations in DNSA+ is less than NSA+ for the dynamic problem. Since the major process of the algorithms is performed in the body and the operations of the body are identical (see Figures 5-2 and 7-2), the CPU-time required to solve the problems is also decreased practically.

In these results for 90 problems, there was about 40 percent reduction in the number of iterations by DNSA+ compared with NSA+. The percentage of improvement, in reduction of the number of iterations, is calculated by the following terms and equation:

- NSA_i⁺ : The number of iterations in NSA⁺ for the dynamic problem at stage i.
- DNSA_i⁺ : The number of iterations in DNSA⁺ for the dynamic problem at stage i.
- TPR : The Total Percentages of Reduction in the number of iterations in the experiment.

$$TPR = \frac{\sum_{i=1}^{90} (DNSA_i^+ - NSA_i^+)}{\sum_{i=1}^{90} DNSA_i^+} * 100 = -38.79\%$$

7.6 Statistical test for the comparison

The number of iterations of running the two algorithms, DNSA+ and NSA+ (Figure 7-11), has been analysed statistically. We tested the null hypothesis that the means produced by the two algorithms were statistically indifferent ($\alpha=5\%$). Table 7-2 provides the test's result along with the values of T-distribution for a particular degree of freedom. Since we cared the change (the difference between the two means) was positive or negative, 'One-tail' test was chosen. The Paired T-test determines the two means are significantly different at 95% degree of confidence since the test's result is in the reject region (see Figure 6-3 for the acceptance and reject regions).

Table 7-2: The result of T-Test for the two algorithms, DNSA+ and NSA+

Statistical Parameters	Values
Observations	90
T-Test (Paired Two Sample For Means)	-5.0936
Degree of Freedom	89
Critical T-Value	-1.662

7.7 Complexity of the algorithm

The complexity of Network Simplex plus Algorithm was calculated in Section 6.5. In this section, it is shown that Network Simplex plus Algorithm and Dynamic Network Simplex plus Algorithm have the same complexity. Both the algorithms run the 'BFS' procedure, which finds a Basic Feasible Solution at the beginning. The Dynamic Network Simplex plus Algorithm then calls the 'Reconstruct New BFS' procedure to repair the spanning tree and current solution when 's' (the input of the algorithm) becomes greater than zero. Given n as the number of nodes in the graph, it is easy to understand that the complexity of both BFS and 'Reconstruct new BFS' are n and $3n^2$, respectively. Based on the number of iterations and the complexity of each pivot (see Section 6.5), the total complexity of this algorithm is determined as follows:

$$O(3n^2 + (m + n)mn^2C^2KLogK)$$

Note that m is the number of arcs in the graph model. In Section 6.5, we had the following equations:

$$m = O(N^2); n = O(N) \text{ (N is the number of jobs)}$$

Therefore, the total complexity of the algorithm for the problem is:

$$O(N^6)$$

7.8 Summary and conclusion

In this chapter, the dynamic extensions of NSA and NSA+ were presented. These extensions are Dynamic Network Simplex Algorithm (DNSA) and Dynamic Network Simplex plus Algorithm (DNSA+). To evaluate the performance of the algorithms, we considered the dynamic scheduling problem of AGVs in the container terminal (the problem defined in Chapter 4). Many random problems have been solved by both DNSA+ and NSA+. The comparison showed that the number of iterations significantly are improved.

To conclude Network Simplex Algorithm and its three extensions (NSA+, DNSA and DNSA+), we made a summary. Table 7-3 shows this summary, including the important features of these algorithms as well as their advantages and disadvantages. In dynamic problems, NSA and NSA+ start from the scratch and reconsider the pre-established schedules. Memory management in these two algorithms is easy task since a block of memory is allocated for the whole of the graph. Also there is no partitioning in the graph and its spanning tree to solve the problem by those algorithms. The disadvantage of these algorithms is to take a time to rebuild the graph and put it into memory. DNSA and DNSA+ repair the solution rather than starting from scratch. The main advantage of these dynamic algorithms over NSA and NSA+ is the performance. On the other hand, DNSA and DNSA+ deal with memory management, partitioning of the graph and its spanning tree. However, they are disadvantages and have to be paid for the performance.

Table 7-3: A comparison between NSA and its extensions

Algorithms	Data Structure	Features	Memory Management	Advantages	Disadvantages
NSA	Graph and operations on the graph	The standard version of the algorithm	Easy: One block of memory is allocated for the whole graph	Faster than equivalently size Linear Program	Time-consuming to rebuild the graph in dynamic problem
NSA+		NSA with enhanced features		Faster than NSA	
DNSA		Dynamic version of NSA; Repairs the solution and spanning tree	Difficult: Partitioning of the graph and its spanning tree	Faster than NSA and NSA+ in dynamic problems	Needs memory management; adding, removing and updating nodes and arcs
DNSA+		Dynamic version of NSA+; Repairs the solution and spanning tree		Faster than DNSA in dynamic problems	

Chapter 8: Greedy Vehicle Search and Dynamic Scheduling of AGVs

In this chapter, an incomplete algorithm to the scheduling problem of AGVs is presented. We called it Greedy Vehicle Search (GVS). To evaluate the relative strength and weakness of Network Simplex plus Algorithm (NSA+) and GVS, results of the two algorithms are compared.

8.1 Motivation

In the previous three chapters, the scheduling problem of Automated Guided Vehicles, the problem in Chapter 4, was solved by NSA and its extensions. Although these complete solutions are efficient, they can only work on problems with certain limits in size (see Section 5.6). When size of the problem goes beyond the limits or the time available for computation is too short, incomplete search methods are used. To complement the solutions, Greedy Vehicle Search (GVS) method is designed and implemented in this chapter. This incomplete search method can be applied to both the static and dynamic problems.

8.2 Problem formalization

The problem here is the problem defined in Chapter 4, but we model it as an incomplete case of the MCF-AGV model (see Definition 4-12). The MCF formulation requires this incomplete model. Given M AGVs and N jobs in the problem, there are M vehicle nodes, N job nodes and one sink node in the model. The graph model is illustrated by Figure 8-1 for two AGVs and four container jobs.

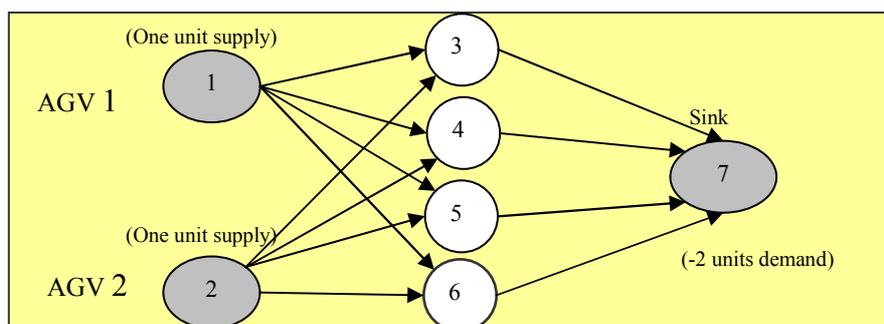


Figure 8-1: An example of the incomplete case of the MCF-AGV model with two AGVs and four jobs

The MCF-AGV model and its incomplete case can be compared. There were four different types of arcs in the MCF-AGV model; the Inward arcs, Intermediate arcs, Auxiliary arcs and Outward arcs. In the incomplete case of the MCF-AGV model, there are only Inward and a partial of Outward arcs. Moreover, in the MCF-AGV model we considered two nodes for each job. In the incomplete case of the MCF-AGV model, there is only one node for each job (see Figures 8-1 and 4-5 for the differences). We formalize this incomplete case with two definitions.

Based on Definitions 4-11, we introduce the following definition:

Definition 8-1: A graph $G_{MCF-AGV-I} = (GSI, NPSI, APSI)$ is an Incomplete graph of $G_{MCF-AGV} = (GS, NPS, APS)$. The elements of $G_{MCF-AGV-I}$, nodes and arcs in the $GSI = (NSI, ASI)$, are formally defined in the two following sub-sections:

8.2.1 Nodes and their properties in the incomplete graph

There are three types of nodes in the $G_{MCF-AGV-I}$. The elements in each set and the sets themselves with their properties are defined as follows:

- a) **AGVN_m**: a supply node corresponding to vehicle m . Each node has one unit supply. Hence, there are M supply nodes in the model. We define the following set for these supply nodes:

SAGVN: a set of M supply nodes in the $G_{MCF-AGV-I}$.

$$SAGVN = \{AGVN_m \mid m=1,2,\dots,M; NPS(m)=1\}$$

- b) **JN_j**: a node for job j . There is neither supply nor demand in this node, i.e. it is a transshipment node. We define the following set for these transshipment nodes:

SJN: a set of N job nodes in the $G_{MCF-AGV-I}$.

$$SJN = \{JN_j \mid j=1,2,\dots,N; NPS(j)=0\}$$

- c) **SINK**: This is a demand node in the $G_{MCF-AGV-I}$ with M units demand. This node corresponds to the end state of the process. For the property of this node, we have:

$$NPS(SINK)=-M$$

Therefore, there are $M+N+1$ nodes in the $G_{MCF-AGV-I}$:

$$NSI=SAGVN \cup SJN \cup SINK$$

8.2.2 Arcs and their properties in the incomplete graph

The following two types of arcs connect the nodes in the $G_{MCF-AGV-I}$:

- 1) Inward Arcs:** There is a directed arc from every vehicle node, to the node of job i . We use the following notation for these arcs:

ARC_{inward} : a set of arcs from SAGVN to SJN.

$$ARC_{inward} = \{ (m, j) \mid \forall m \in SAGVN, \forall j \in SJN, APS(m, j) = [0, 1, C_{mj}] \}$$

The number of these arcs is $M \times N$. Each arc has the lower bound zero, and the upper bound one, i.e., only one AGV goes through each of these arcs. Given the appointment time of container job j , t_j , the ready time of AGV m to get the next location, RTA_m , and the travel time of the AGV from its next location to the source/destination of job j on the quay side, TTA_{mj} , the cost of arc (m, j) is calculated as:

$$C_{mj} = \begin{cases} w_1 \times (t_j - (RTA_m + TTA_{mj})) + w_2 \times (RTA_m + TTA_{mj}) & \text{if } (t_j \geq RTA_m + TTA_{mj}) \\ P \times (RTA_m + TTA_{mj} - t_j) & \text{otherwise} \end{cases}$$

Note that this cost is exactly the same as what we calculated in Chapter 4 (see Section 4.5.2 and Assumption 4-10); w_1 and w_2 are the weight of waiting and travelling times of the vehicles, and P is a penalty.

- 2) Outward Arcs:** There is a directed arc from every job node i to SINK. We use the following notation for these arcs:

$ARCI_{outward}$: a set of arcs from SJN to SINK.

$$ARCI_{outward} = \{ (i, j) \mid \forall i \in SJN, j = SINK, APS(i, j) = [0, 1, 0] \}$$

The number of these arcs is N . Each arc has the lower bound zero; the upper bound one and the cost zero, i.e., the AGV (that visited a job node in SJN) goes through each of these arcs.

Therefore, there are $M \times N + N$ arcs in the $G_{MCF-AGV-I}$:

$$ASI = ARC_{inward} \cup ARCI_{outward}$$

8.2.3 The special case of the MCF-AGV model for Automated Guided Vehicles Scheduling

Now we present an incomplete version of the MCF-AGV model for the Automated Guided Vehicles Scheduling with the following definition.

Definition 8-2: A MCF-AGV-I model is defined on graph of $G_{\text{MCF-AGV-I}}$ as an Incomplete case of the MCF-AGV (Definition 4-12) for the Scheduling problem of Automated Guided Vehicles. The elements of D , CS and FC in the $\text{MCF-AGV-I} = (G_{\text{MCF-AGV-I}}, f, D, CS, FC)$ are introduced as follows:

a) For each element in D , we have:

$$D_{f_{ij}} = [0, 1] \text{ for } (i, j) \in \text{ARC}_{\text{inward}} \cup \text{ARCI}_{\text{outward}}$$

b) The constraints of CS in the MCF-AGV-I are:

$$\left. \begin{array}{l} 1) \sum_{j:(i,j) \in \text{ASI}} f_{ij} = 1, \forall i \in \text{SAGVN} \\ 2) \sum_{j:(j,i) \in \text{ASI}} f_{ji} = M, i = \text{SINK} \\ 3) f_{is} = \begin{cases} 1 & \text{if } \sum_{m:(m,i) \in \text{ARC}_{\text{inward}}} f_{mi} = 1 \\ 0 & \text{otherwise} \end{cases}, \forall i \in \text{SIN}, s = \text{SINK} \end{array} \right\}$$

The first constraint shows every node i ($i \in \text{SAGVN}$) sends one unit flow into the network. The second constraint ensures SINK node receives M units flow (the flows sent from nodes in SAGVN set). The third constraint shows that one unit flow can be sent from every node in SIN to SINK provided that it received one unit flow.

c) The objective function is:

$$FC = \underset{\substack{m=1,2,\dots,M \\ j=1,2,\dots,N}}{\text{Min}} C_{mj} \times f_{mj}$$

Solving the MCF-AGV-I model generates M paths, each of which commences from a node in SAGVN and terminates at SINK. Each path determines a job for every AGV. The decision variable f_{ij} for every $(i,j) \in \text{ASI}$ (the flow between nodes i and j in the $G_{\text{MCF-AGV-I}}$) is either 1 or 0. $f_{ij} = 1$ means that an AGV goes from node i to node j . Otherwise, moving the AGV from node i to node j is not possible.

8.3 Algorithm formalization

The block diagram of Greedy Vehicle Search method is demonstrated by Figure 8-2.

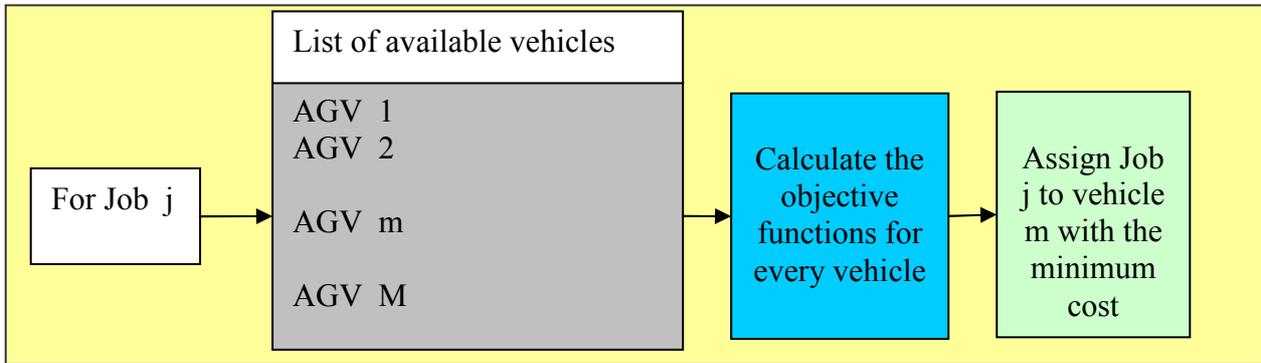


Figure 8-2: The block diagram of Greedy Vehicle Search.

There are N container jobs and M vehicles in the problem (the same as Chapter 4). In this simple search method, every time a job needs to be served, as what a Taxi Service System (TSS) does. In fact, for any unassigned job and the list of idles AGVs, a job is assigned to a vehicle with minimum cost, including waiting and travelling times of the vehicles as well as lateness of the jobs.

The pseudo code of GVS in dynamic aspect is demonstrated by Figure 8-3. This pseudo code is divided into two parts. In the first part, the cost for any combination between the remaining jobs and the idle vehicles is calculated. In the second part, one vehicle is assigned to a job, based on the minimum cost.

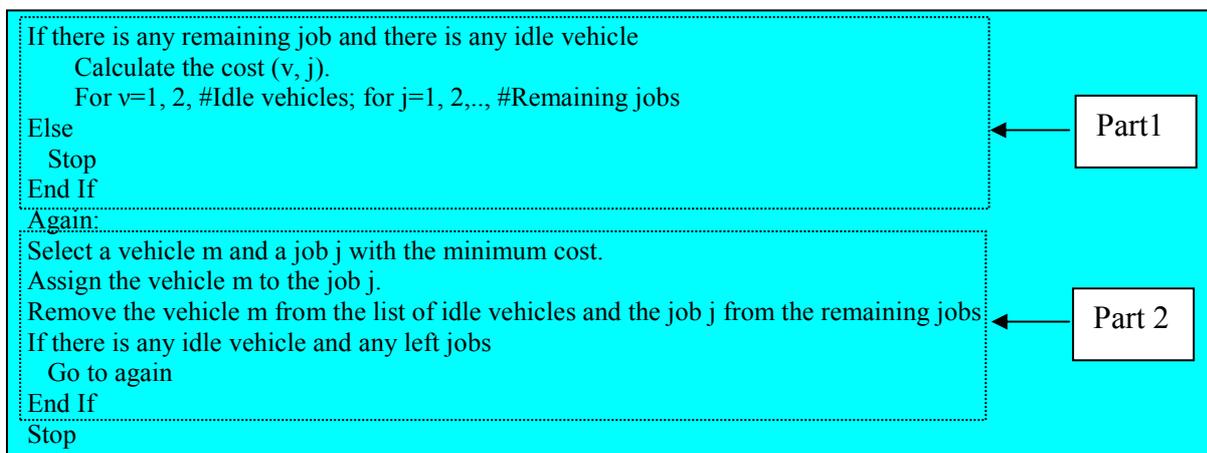


Figure 8-3: The pseudo code of Greedy Vehicle Search in dynamic aspect

8.4 Software architecture for dynamic aspect

The architecture of main part of the software is demonstrated by Figure 8-4. At the start of the process, the Job Generator generates a few jobs for the cranes. These jobs will be appended to the remaining jobs, which is empty at the beginning. The remaining jobs are used by Greedy Vehicle Search and the output of this method is an individual job for every vehicle.

The software does two tasks in the real time processing and dynamic fashion. The first task is related to updating the vehicle's status and assigning a job to any available vehicle whereas the second one takes influence from any idle crane. While the time is running, the amount of time travelled and waited for every vehicle is updated. At the same time, if a vehicle picks up a job from the quay side, the assigned job will be deleted from the list of jobs for the vehicle and will be removed from the list of remaining jobs. If the job has to be delivered to the crane on the quay side, it could not be removed until the meeting time between the crane and the vehicle (note that, the appointment place is on the quay side, not the yard side). The second task refers to any change in the crane's status. The Job Generator has to generate a few new jobs, when it finds out any idle crane.

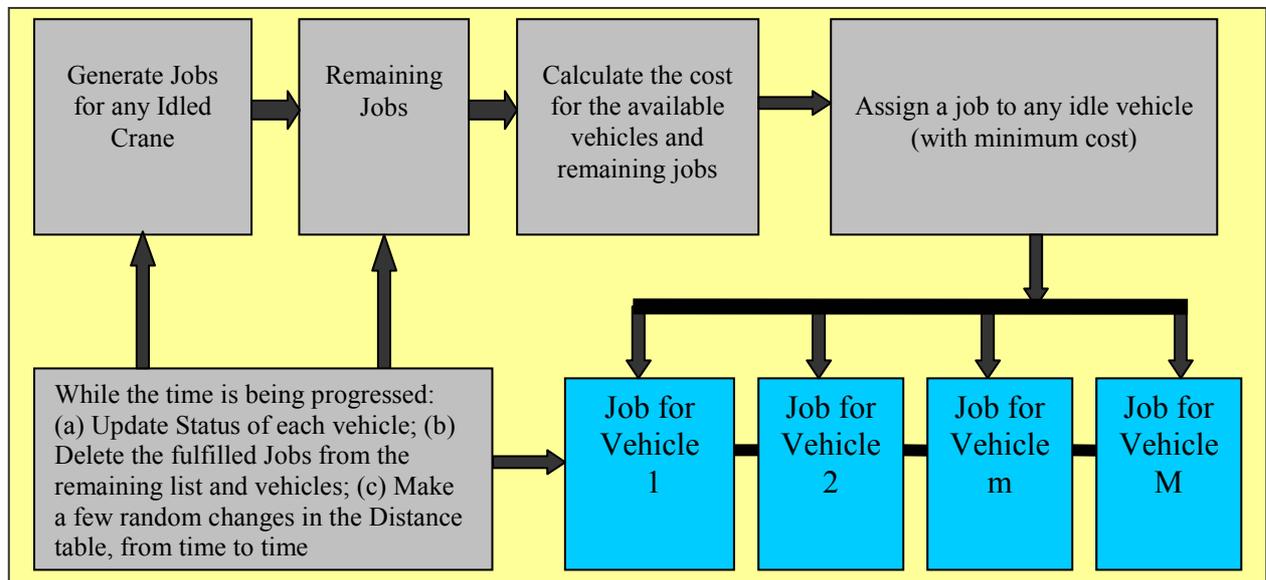


Figure 8-4: The block diagram of the software and algorithm (GVS) in dynamic aspect

From time to time, the software makes a few random changes in the distance table (see Table 4-1) in order to produce dynamic problems. These changes are applied to the problem directly. Since the algorithm is reactive, it finds out a solution for the new problem in each run.

8.5 A comparison between GVS and NSA+ and quality of the solutions

To evaluate the relative strength and weakness of GVS and NSA+ in the dynamic scheduling problem, we used randomly generated problems. Distance between every two points in the port as well as the source and destination of jobs were chosen randomly. We did a simulation for 6 hours subject to generating 5 jobs for any idle crane. Other parameters for this simulation were the same as Table 5-1. We compared solutions of the both algorithms, NSA+H and GVS. The components in the objective function, the number of carried jobs and delay from the appointment time were compared in this experiment. Our observations were:

Observation 8-1: Figures 8-5 shows components in the objective function, the waiting and travelling times of vehicles for both the algorithms. As we can see from the figure, waiting times of the vehicles for Greedy Vehicle Search is significantly greater than waiting times of the vehicle in Network Simplex plus Algorithm, although travelling times of the vehicles for both algorithms are almost the same during the 6 hours. The main reason for the result is that Network Simplex plus Algorithm solves the MCF-AGV model (in Chapter 4) and produces the global optimum solution for the problem whereas GVS does a search in the search space and finds out a local optimum for the MCF-AGV-I model.

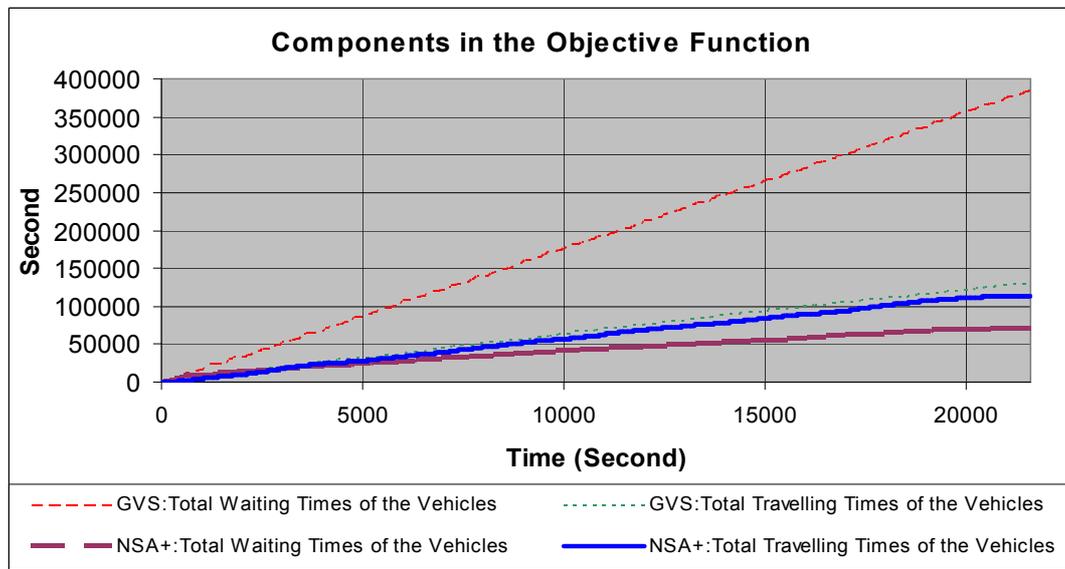


Figure 8-5: A comparison of NSA+ and GVS for Travelling and Waiting Times of the Vehicles

Observation 8-2: Figure 8-6 shows the number of carried jobs during the six hour simulation (21,600=6×3,600). As we can see in the figure, the number of carried jobs for both algorithms, NSA+ and GVS, approximately is the same. Generally, due to the tight schedules of the quay cranes, it is undesirable for containers to be served early or too late for the appointment.

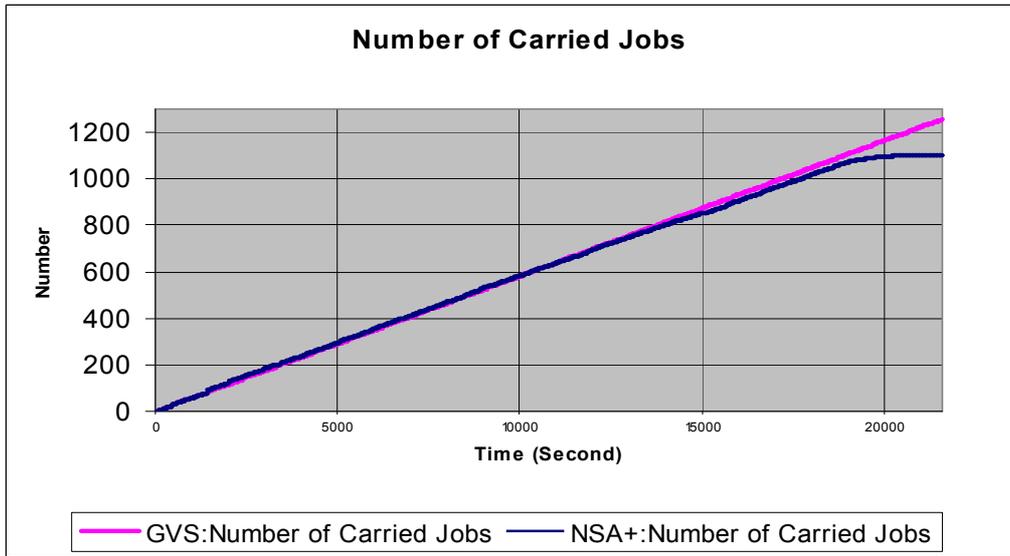


Figure 8-6: The number of carried jobs by NSA+ and GVS during 6 hour simulation

However, it may be argued that the average lateness from the appointment times is another indicator for goodness of the algorithms. Given the number of served jobs, N , the time at which the job i is served, ACT_i , and the time of Appointment, APT_i , the Average Lateness is calculated by the following equation:

$$Average\ Lateness = \frac{\sum_{i=1}^N (ACT_i - APT_i)}{N}$$

For this indicator, we got the following observation:

Observation 8-3: Figure 8-7 presents the Average Lateness indicator for both NSA+ and GVS during the six-hour simulation. The figure shows that both algorithms performed well, but GVS is superior to NSA+ in the Average Lateness. GVS sacrifices the waiting and travelling times of the vehicles to the Average Lateness.

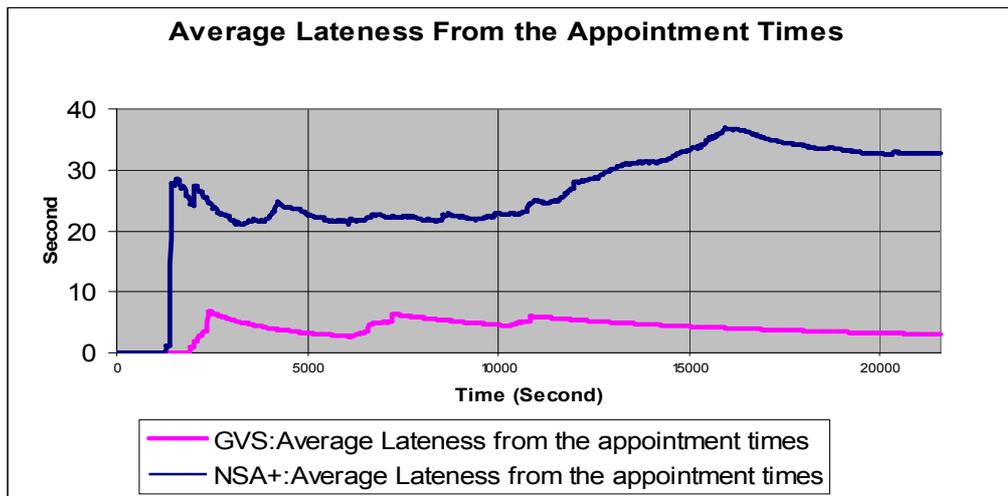


Figure 8-7: A comparison of NSA+ and GVS for the Average Lateness from the appointment time

8.6 Statistical test for the comparison

The waiting and travelling times of the vehicles as well as the average lateness of jobs, produced by NSA+ and GVS, were analysed statistically. During the simulation, the samples were collected at regular 30 second intervals. We tested the null hypothesis that the means produced by the two algorithms were statistically indifferent ($\alpha=5\%$). Table 8-1 provides the test's result along with the Critical t-value for a particular degree of freedom.

Table 8-1: The result of T-Test for the two algorithms, GVS and NSA+

Statistical Parameters	Total Waiting Times of the Vehicles	Total Travelling Times of the Vehicles	Average Lateness from the appointment times
Observations	720	720	720
T-Test (Paired Two Sample For Means)	-43.4054744	-43.5902651	73.6809406
Degree of Freedom	719	719	719
Critical T-Value	-1.646972	-1.646972	-1.646972

Since we cared the change (the difference between the two means) was positive or negative, 'One-tail' test was chosen. The Paired T-test confirms that NSA+ is significantly better than GVS in both travelling and waiting times of the vehicles with 95% level of confidence (see Figure 6-3 for the acceptance and reject regions). On the other hand, GVS is statistically better than NSA+ in the Average Lateness.

8.7 Complexity of Greedy Vehicle Search

As we mentioned, GVS can be applied to both static and dynamic problem. In this section, complexity of the algorithm is calculated.

8.7.1 Complexity of GVS for static problem

For static problems, we assume that every job has to be served by the vehicles. The algorithm operates as follows:

In the first run, it finds out one job with minimum cost (among N jobs) for a vehicle. In the second run, another job among $N-1$ jobs is assigned to a vehicle, which could be the selected vehicle in the first run or others. This process is continued until there is no remaining job. Hence, given N jobs and M vehicles in the problem, the complexity of the algorithm is calculated by the following equation:

$$M \times N + M \times (N - 1) + M \times (N - 2) + \dots + M \times 1 = \frac{M \times N \times (N + 1)}{2}$$

Therefore, the complexity of GVS is $O(M \cdot N^2)$. It is less than the complexity of NSA+ (see Section 6.5)

We got some samples to show the performance of GVS. The CPU-Time required to solve the problems by GVS is shown in Figure 8-8.

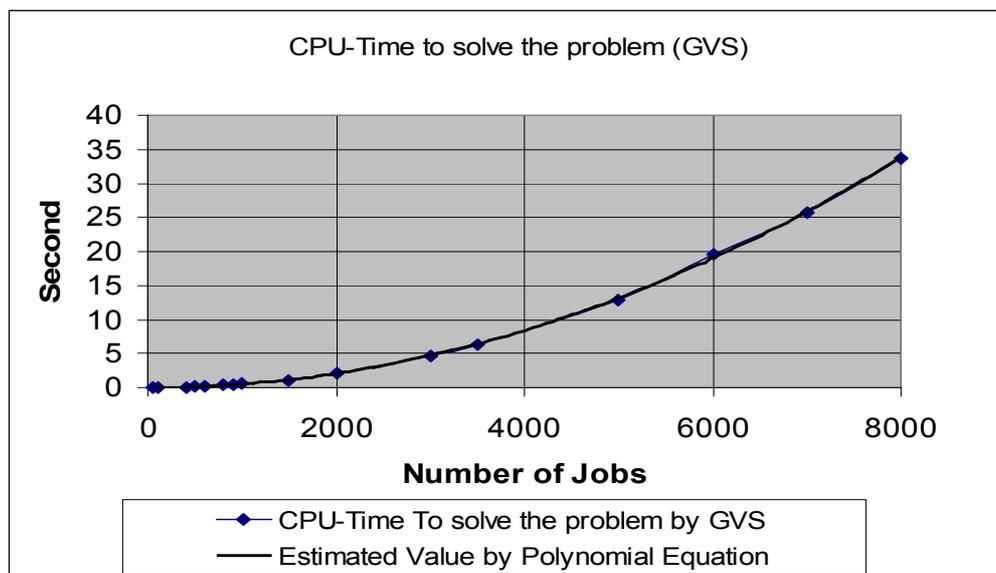


Figure 8-8: CPU-Time required to solve the static problems by GVS

The estimated values by a polynomial equation (with degree 2) have also been shown on the figure. As we can see when the number of jobs is 8,000, it takes 35 seconds CPU-time to solve this large problem ($M = 50$). Note that these samples have been collected by running GVS on Windows-XP computer with 2.2 GHz processor and 1GB RAM.

From a comparison between Figure 8-8 and Figure 5-11, we can observe that:

Observation 8-4: Greedy Vehicle Search (GVS) is faster than NSA and NSA+. Moreover, GVS could solve the larger problems, which are beyond of the limits of NSA and NSA+. GVS could find a local optimum for the problem of 8,000 jobs within 35 seconds whereas NSA and NSA+ solve the problem with 3,000 jobs within 2 minutes. The reason is that GVS is an incomplete algorithm while NSA and NSA+ are complete.

We made an estimate of time complexity of the algorithm by the experimental results in Figure 8-8. The time complexity can be expressed in CPU-Time required to find for a local optimum. The CPU-Time is estimated based on the number of jobs. We considered the following equation to estimate CPU-Time required to find a local optimum:

$$CPU - Time_{GVS} = f(\text{Number of Jobs}) = a \times \text{NumberofJobs} + b \times \text{NumberofJobs}^2$$

The estimation's results for the CPU-Time have been shown in Table 8-2. The coefficients of 'a' and 'b' have been calculated and put in the Coefficients section of the table.

Table 8-2: Regression result for CPU-Time required to finding a local optimum by GVS for static problem

Multiple R	R-Square	Adjusted-R-Square	Standard Error	Observations		
0.99988	0.99977	0.92832	0.1640	16		
	<i>DF</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance-F</i>	
Regression	2	1656.086	828.043	30780.79811	1.28E-24	
Residual	14	0.3766	0.0269			
Total	16	1656.46				
	<i>Coefficients</i>	<i>Standard-Error</i>	<i>t-Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>
X Variable 1	-7.488E-05	4.121E-05	-1.8171	0.0906578	-0.00016	1.35E-05
X Variable 2	5.33383E-07	6.332E-09	84.242	2.40409E-20	5.198E-07	5.46E-07

Based on the Coefficients in the table, we have the following equation for the CPU-Time to find a local optimum:

$$CPU - Time_{GVS} = -7.488 \times 10^{-5} \times \text{NumberofJobs} + 5.33 \times 10^{-7} \times \text{NumberofJobs}^2$$

More details about information in the table are the same as Section 5.5.

8.7.2 Complexity of GVS for dynamic problem

In dynamic problems, we assume that only one job is assigned to an idle vehicle. Here, there is no doubt that GVS is very fast. We got some samples to show its performance. The CPU-Time required to solve the problems by GVS is shown in Figure 8-9. As we can see when the number of jobs is 10,000, it doesn't get too much CPU-time to solve the large problems (less than 1 second). Note that these samples have been collected by running the software on Windows-XP computer with 2.2 GHz Pentium processor and 1GB RAM for 50 vehicles.

Given the number of jobs and vehicles in the problem, N and M , respectively for this algorithm, its complexity is $O(N \cdot M)$. It is easy to understand this complexity by Figure 8-3 (Pseudo code of the algorithm).

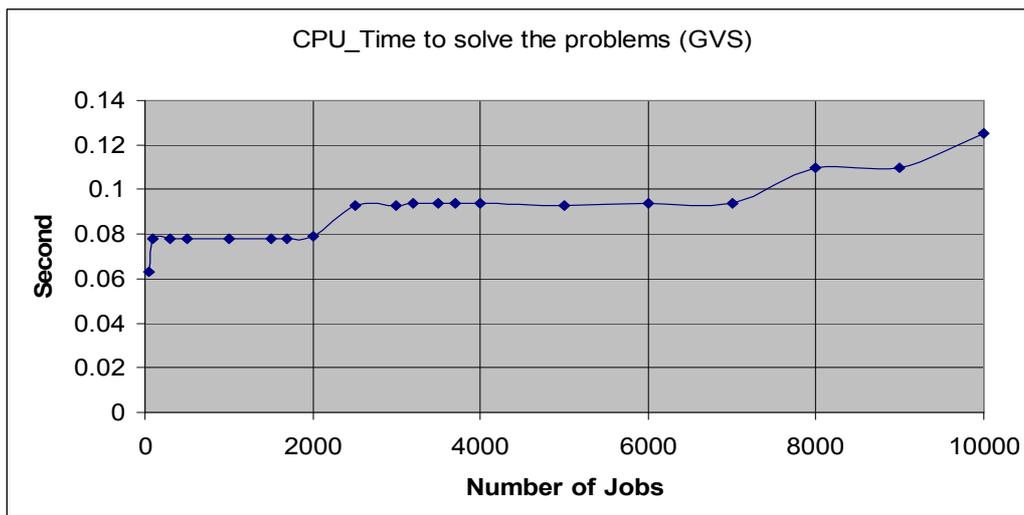


Figure 8-9: CPU-Time required to solve the dynamic problems by GVS

8.8 A discussion over GVS and meta-heuristic

A discussion could be arisen in using GVS compared with some well-known meta-heuristics (stochastic search methods) such as Genetic Algorithms, Tabu Search, Simulated Annealing method and others when the problem is too big or when the time available to tackle the problem is too short. In the literature, we reviewed these solutions methods, including general considerations and major specific considerations in them (see Section 3.7). In this section, we have a short discussion on the matter.

According to the literature, GVS could be considered as a heuristic. Voß (2000) [98] defines heuristic as follows: “A heuristic is a technique (consisting of a rule or a set of rules) which seeks (and hopefully finds) good solutions at a reasonable computational cost. A heuristic is approximate in the sense that it provides (hopefully) a good solution for relatively little effort, but it does not guarantee optimality”. We based GVS on two simple rules, the idle vehicles and jobs remained. Moreover, GVS doesn’t get too much CPU-Time to tackle the problem and for that reason it finds out a local optimum solution.

We had the problem with memory to put the MCF-AGV model into (see Section 5.6). One of the reasons to use GVS is that it has no memory technique. In the literature, we studied that “the meta-heuristics manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration” [98]. In order to do that, they require memory. Although, we didn’t provide any numerical comparison for the matter, our judge is that the memory usage of GVS is nil compared with the meta-heuristics.

Our work shows that GVS solves a huge problem in a short time. The problem of 10,000 jobs and 50 AGVs could be solved in a second (see the previous section). Additionally, GVS is effective in the average lateness to serve the jobs (see Section 8.5). The weakness of the meta-heuristics is that effectiveness could be sensitive to choice of parameters values and operators [94]. Basically, finding out a set of suitable parameters for the meta-heuristics and their training to tackle the problem will be beyond of the scope of this thesis.

8.9 Summary and conclusion

In this chapter a greedy method, Greedy Vehicle Search (GVS) for the scheduling problem of Automated Guided Vehicles was presented. Then, we compared some solutions of GVS and NSA+ for the Dynamic Automated Vehicle scheduling problem. Many random problems with the same distribution were generated and solved by both algorithms. Given the results of the six-hour simulation, we claim that NSA+ is efficient and effective in both waiting and travelling times of the vehicles. GVS is useful when the problem is too big for NSA+ to solve or when the time available to tackle the problem is too short. Being an incomplete algorithm, GVS sacrifices completeness.

Chapter 9: Conclusions and Future Research

This thesis was devoted to solution methods for Static and Dynamic Scheduling problem of Automated Guided Vehicles (SDSAGV) in the container terminals. A special case of Minimum Cost Flow (MCF) model was defined and presented for the problem. Then, we studied the effectiveness and efficiency of the Network Simplex Algorithm (NSA) in the literature. We proposed three new versions of the algorithm; Network Simplex plus Algorithm (NSA+), Dynamic Network Simplex Algorithm (DNSA) and Dynamic Network Simplex plus Algorithm (DNSA+). NSA, NSA+, DNSA and DNSA+ are complete algorithms. They were designed to find optimal solutions. To complement the solutions, Greedy Vehicle Search (GVS) method was designed and implemented. GVS is an incomplete algorithm which can be used for reactive scheduling or when the problem is too big for the complete algorithms. In this final chapter, we summarise the research conducted on NSA, NSA+, DNSA, DNSA+ and GVS and also discuss the prospects of future research on the subject.

9.1 Summary of work done

The research started with the study of problems in container terminals. We classified these problems into five scheduling decisions (Chapter 2). Then we systematically and thoroughly surveyed the literature over these decisions and formulated them as Constraint Satisfaction Optimization Problems (Chapter 3). The survey showed that vehicles are important equipment in the ports and their scheduling is one of the most challenging problems.

We then focused on scheduling problem of Automated Guided Vehicles (AGVs) in container terminals. Another reason to choosing this problem is that the efficiency of a container terminal is directly related to use the AGVs with full efficiency. The problem was to carry many container jobs by several AGVs in their appointment times. We formulated the problem as a Minimum Cost Flow (MCF) model, a directed graph with particular assumptions. The main motivation to formulate the problem as a MCF model is that MCF has a rich history and arises in almost all industries, including agriculture, communications, defence, education, energy, health care, manufacturing, medicine, retailing, and transportation. The MCF problem is to send flow from a

set of supply nodes, through the arcs of the network, to a set of demand nodes at minimum total cost, without violating the lower and upper bounds on flows through the arcs. We defined and presented a special case of Minimum Cost Flow (MCF) model for the Scheduling problem of Automated Guided Vehicles (Chapter 4). The MCF-AGV is an established name for our model. The cost of each arc in the MCF-AGV model was the waiting and travelling time of vehicles as well as the lateness times to serve the container jobs.

The main objectives of this thesis were to solve the Scheduling problem of Automated Guided Vehicles efficiently and effectively. The MCF-AGV model, formulated in Chapter 4, had a huge search space and its solution had to provide the optimal paths for each vehicle. Additionally, the problem was dynamic. From time to time a few new jobs arrived and the distance between the source and destination of the jobs could be changed.

We first tackled the Static problems (defined in Chapter 4). In order to do that, we used Network Simplex Algorithm (NSA), which is one of the solution methods for MCF model. In Chapter 5, we applied the standard version of the algorithm to the problem. We reviewed the literature over NSA and different schemes to select the next basic solution. Then, implementation of the algorithm and finding the optimal solution in static problems were considered. Many random data were generated and fed to the MCF-AGV model for 50 vehicles. Our software, implemented in Borland C++, by running on a 2.4 GHz Pentium PC, could find the global optimal solution for 3,000 jobs and ten millions arcs in the MCF-AGV model within two minutes. It has been found that, in practice, the NSA runs in polynomial time to solve the problems.

To tackle the Dynamic Scheduling problem of Automated Guided Vehicles (the problem in Chapter 4), we extended Network Simplex Algorithm (NSA). In Chapter 6, some enhanced features were added to NSA to have obtained a novel version of the algorithm, Network Simplex plus Algorithm (NSA+). The same MCF-AGV models were solved by both algorithms, NSA and NSA+, and CPU-Time required to solve the problems has been compared. Our experiments showed that NSA+ can solve the problems faster than NSA. Then, complexity of NSA+ was calculated. After that, the software for dynamic aspect of the problem has been executed for six hour simulation. The result of simulation showed the ‘Actual time’ of jobs, at which they have been handled by the vehicles and cranes, have a good fitting with their ‘Appointment times’.

Another goal of this thesis was to extend NSA in dynamic aspect. In Chapter 7, Dynamic Network Simplex Algorithm (DNSA) and Dynamic Network Simplex plus Algorithm (DNSA+) were presented. The objectives of these algorithms were to solve the new problem faster, to use some parts of the previous solution for the next problem and to respond to change in the situation. In order to confirm the validity of DNSA+, again we used the Dynamic Scheduling problem of Automated Guided Vehicles (the problem defined in Chapter 4). The same problems have been solved by NSA+ and DNSA+. Our experiment showed that the number of iterations is decreased if we repair the current solution for the next problem when any changes happen, compared with starting from the scratch by NSA+.

NSA and its extensions are complete algorithms. Although they are efficient, they can only work on problems with certain limits in size. To complement the algorithms, the Greedy Vehicle Search (GVS) method was designed and implemented (Chapter 8). GVS is useful for problems which sizes go beyond the limits, or in dynamic scheduling where reactive responses are called for, or when the time available to tackle the problem is too short.

To evaluate the relative strength and weakness of GVS and NSA+ in the Dynamic Scheduling problem (the problem defined in Chapter 4), we used randomly generated problems. The objective function of the problem had three terms, waiting times of the AGVs, travelling times of AGVs and the lateness time to serve the jobs. We did a simulation for 6 hours. By the end of the simulation, we claimed that (a) NSA+ is efficient and effective in both waiting and travelling times of the vehicles; (b) GVS is efficient in the average lateness to serve the container jobs.

Table 9-1 makes a summary of the solution methods for the problem in Chapter 4. NSA and its extensions, as complete algorithms, and GVS, as an incomplete algorithm, have been studied in this thesis. These algorithms have been applied to the defined scheduling problem of Automated Guided Vehicles. The main features, complexity, performance and effectiveness of the algorithms have been compared in the table. Additionally, we specified which algorithms were designed and convenient for the static/dynamic problem.

Table 9-1: A summary of the algorithms studied in this thesis for the MCF-AGV model

Algorithms (Reference)	Main Feature	Complete/ Incomplete algorithm	Static/Dynamic Problem	Performance	Complexity (Reference)	Effectiveness (Reference)
NSA (Chapter 5)	A graph algorithm to solve MCF model	Complete; Produce optimal solution.	Designed for static problems; when applied to dynamic problems, the changed problems are tackled from scratch.	Faster than equivalently size Linear Program. It has a lower complexity than Original Simplex Method.	O (N ⁶); N is the number of jobs in the problem (Sections 6.5 and 7.7). We assumed the number of jobs is greater than the number of vehicles.	Effective in minimizing both travelling and waiting times of the vehicles (Section 8.5).
NSA+ (Chapter 6)	A graph algorithm with enhanced feature to solve MCF model			Faster than NSA in both static and dynamic problems		
DNSA (Chapter 7)	Dynamic version of NSA to solve MCF model		Designed for dynamic problems; graph structure is changed incrementally.	Faster than NSA and NSA+ in dynamic problems		
DNSA+ (Chapter 7)	Dynamic version of NSA+ to solve MCF model			Faster than DNSA in dynamic problems		
GVS (Chapter 8)	Greedy Vehicle Search to solve MCF model in the special case	Incomplete; Produce a local optimum.	Designed for both static and dynamic problems, preferred when size of the problem is beyond the limits of NSA, NSA+, DNSA and DNSA+ or when the time available to tackle the problem is too short	Faster than NSA and its extensions (NSA+, DNSA, DNSA+).	Given N jobs and M vehicles in the problem 1) O (M·N ²): for static problems (Section 8.7.1) 2) O (M·N): for dynamic problems (Section 8.7.2).	Effective in minimizing the lateness time to serve the jobs (Section 8.5).

9.2 Observations and conclusions

Based on the experimental results by the algorithms, studied in this thesis for the problem defined in Chapter 4, we summarize the conclusions as follows:

- NSA, NSA+, DNSA and DNSA+ are complete algorithm whereas GVS is incomplete. The solutions of the complete algorithms are optimal while GVS provides a local optimum solution for the problem.
- NSA, NSA+, DNSA and DNSA+ solve the whole problem and assign every job to the vehicles. In GVS, each job is assigned to just one vehicle with minimum cost. In the normal situation the number of vehicles is less than the number of jobs in the port. In this case, if the problem is solved by GVS, then the number of remaining jobs after the first run is not zero. This shows the search is continued and the rate of execution to find out a job for the vehicles is significant (when the number of jobs is high) compared with other algorithms.
- NSA, NSA+, DNSA and DNSA+ are efficient and effective in both traveling and waiting times of the vehicles. GVS is more effective and efficient in the lateness time to serve the jobs.
- GVS is useful for both static and dynamic problems when the problem is too big. GVS has a lower complexity than the complete algorithms. It can be used when the size of problem is beyond of the limit of the complete algorithms or when the time available to solve the problem is too short.
- The performance of NSA+ is better than NSA in both static and dynamic problems.
- In dynamic aspect when there are changes in the problem, DNSA and DNSA+ have a better performance than NSA and NSA+, respectively. We therefore suggest DNSA and DNSA+ for dynamic problem and NSA and NSA+ for static one. If the percentage of changes is more than 60 percent, NSA+ is preferred in our experience.
- Given the results, we claim that NSA, NSA+, DNSA and DNSA+ as well as GVS are practical algorithms for Automatic Vehicle Scheduling.

9.3 Research contributions

The main contributions of this thesis are as follows:

- We formulated the five scheduling decisions, defined in Chapter 2, as Constraint Satisfaction Optimization Problems (Chapter 3).
- We presented a definition for the special Graph of the MCF model and a formal definition for the MCF model itself. We formulated the Scheduling problem of Automated Guided Vehicles in container terminals and modelled it under the MCF. We established a name for the model, the MCF-AGV (Section 4.5 in Chapter 4). The objective function of the MCF-AGV model is to minimize the travelling and waiting times of vehicles as well as the lateness time to serve container jobs, as a single objective optimization problem.
- We have applied the standard version of Network Simplex Algorithm to the static problem (Defined in Chapter 4). Our software can find the global optimal solution for 3,000 jobs and ten millions arcs in the MCF-AGV model within two minutes⁴ by running on 2.4 GHz Pentium PC (Chapter 5).
- We have developed a novel version of Network Simplex Algorithm (NSA), Network Simplex plus Algorithm (NSA+). We have demonstrated that NSA+ is faster than NSA (Chapter 6).
- We have extended NSA to dynamic problems. In this aspect two algorithms, Dynamic Network Simplex Algorithm (DNSA) and Dynamic Network Simplex plus Algorithm (DNSA+) were presented. The objectives of these algorithms are to respond to change in the problem and to use some parts of the previous solution for the next problem. In dynamic aspect, DNSA and DNSA+ are faster than NSA and NSA+, respectively (Chapter 7).
- We have developed Greedy Vehicle Search (GVS) algorithm for Scheduling Automated Guided Vehicles in the container terminals. It can be applied to both static and dynamic problems. GVS is an incomplete algorithm and useful when the problem is too big for the complete algorithms or when the time available to tackle the problem is too short (Chapter 8).

⁴ <http://privatewww.essex.ac.uk/~hrashi/Overview%20of%20this%20research.htm>

- We have produced a set of benchmark⁵ problems for Automated Guided Vehicles in the container terminals. These are published in our benchmark web pages, which enable other researchers to compare other algorithms to the one proposed in this thesis. Now there are four sizes of the problem (Small, Medium, Large and very Large) with their solutions.

9.4 Future research

The research reported in this thesis (discussed in Chapters 4 to 8) focused on certain topic of Scheduling problem of Automated Guided Vehicles in the container terminals. In this section, several topics for further research are presented.

9.4.1 Scheduling and routing of the vehicles

The first interesting extension to this research is to combine scheduling and routing of the vehicles together. In Chapter 4, this research assumed that there are no traffic problems such as breakdown, congestion, collision, live-lock and deadlock for the vehicles while they are carrying and handling the jobs. Therefore, a possible extension is to relax this assumption and develop a new algorithm for routing of vehicles according to different port layout with respect to those traffic problems.

A few different topologies for container terminal including linear path, single-circle and mesh-like path [79] may be considered. In linear path topology the scheme is to schedule and route a batch of AGVs concurrently. In the second topology, circle, including single-circles and multi-circles, few vehicles are running in same direction within the circle. In the last topology, mesh-like path, the the storage area are usually arranged into rectangular blocks, which leads to a mesh-like path topology for the vehicles.

In an automated container terminal, the traffic problems are critical. An AGV malfunction or breakdowns lead to an interruption in container handling. Collision occurs when more than one AGV attempt to occupy the same segment of the path at the same time. Congestion arises at a

⁵ <http://privatwww.essex.ac.uk/~hrashi/Current%20Research.htm#CurrentResearch>

location where there is insufficient resource so that for a period of time the number of arrivals is greater than that of serviced requests. A live-lock may arise at the junction where the horizontal stream of traffic is given higher priority to obtain the left-of-way such that the vertical one may keep waiting indefinitely. A deadlock will arise when multiple AGVs mutually wait for the release (which will never occur) of the resource held by the others. The problem, here, will be to find a suitable route for the AGVs from origin to destination based on current traffic situation, according to the port topology.

It should be clear that AGV systems are, intrinsically, parallel and distributed systems that require a high degree of concurrency. Our feeling is that the routing and scheduling of these systems are a fertile area where engineers and computer scientists can have significant contributions.

9.4.2 Economic and optimization model

Investments in container terminals are very substantial and scheduling of their equipment are very challenging problem. In order to obtain maximum benefits it is necessary to develop an economic model and combine it with an optimisation model. It should be pointed out that in the literature there is no significant model with links between economic indicators and the optimisation model. Further research on this topic is needed.

As we mentioned, the main functions of container terminals are delivering containers to consignees and receiving containers from shippers, loading containers onto and unloading containers from vessels and storing containers temporarily. A complete economic plan has to identify and represent the fundamental components in container terminal and transportation system. These components are demand, supply, cost, performance measures, and decision criteria. Their interactions may be considered. Developing a demand function to receiving containers from shippers, developing a supply model to delivering containers to consignees, estimating a cost function for the vehicles, quay cranes, yard cranes and even container terminal are in the list for the future. The research may estimate the weights of travelling and waiting times of the vehicles, the weights of holding cost of jobs on the quay-side or in the yard-side with particular assumptions. A performance function based on some economic indicators may be maximized.

Constraints of the function are the spatial allocation of containers in the terminal yard, the allocation of resources and the scheduling of operations.

Therefore development an integrated system for both aspects, economic and optimisation, is suggested for future research. Automatic adaptation and estimation methods in real time are necessary.

9.4.3 Other possible extension

Automated Guided Vehicles in the container terminals, as the most flexible equipment, affect other decisions in the port. Therefore, another possible extension is to integrate the scheduling of Automated Guided Vehicles with other decisions. Allocation of berth to arriving vessels, Quay Cranes to docked vessels, storing the incoming containers in the yard and deployment of the Yard Cranes may be in the candidate lists for this integration. These decisions have been formulated in Chapter 3 of this thesis.

Firstly, allocation problem of berth and quay cranes to arriving vessel may be integrated with Scheduling of Automated Guided Vehicles. An objective function of the integrated decision is to minimize the sum of handling costs of containers. A set of assumptions and constraints according to the berth, quay cranes and vehicles should be considered. New solution methods may need to be developed.

Secondly, storing incoming containers in the yard has an important role in global productivity of the terminal. It can be combined with Scheduling of Automated Guided Vehicles. An objective function of this decision is to minimize distribution of the total number of containers among blocks in the yard and the sum of container transportation costs. A set of assumptions and constraints according to layout of the yard and movement of the vehicles should be considered in the model. New solutions may be needed to be developed.

Thirdly, deployment of yard cranes is also highly interrelated to the movement and Scheduling of Automated Guided Vehicles. These two decisions can be combined together. The objective function of this decision is to minimize the remaining workload at each block, travelling and waiting times of the vehicles as well as travelling time of the RTGCs among blocks during the

planning horizon. Developing new algorithms and new deployment policy for RTGC are recommended.

Appendix: Information on Web

This research is focused on [Dynamic Scheduling of Automated Guided Vehicles](#)⁶ (AGV). The problem is to schedule several AGVs in a port to carry many containers from the quay-side to yard-side or vice versa. This problem is formulated as a minimum cost flow problem and then solved by Network Simplex Algorithm (NSA), Network Simplex plus Algorithm (NSA+), Dynamic Network simplex Algorithm (DNSA) and Dynamic Network simplex Plus Algorithm (DNSA+). In this research my contributions are NSA+, DNSA and DNSA+. NSA+ is faster than NSA. DNSA and DNSA+ repair the previous solution when any changes happen.

<p>Instances for Static Problems and their Solutions</p> <hr/> <p>Distance Table Problem1 (Small Size) Problem2 (Medium Size) Problem3 (Large Size) Problem4 (Extra Large Size)</p>	<p>Instances for Dynamic Problems and their Solutions</p> <hr/> <p>Problems Solutions Performances</p>	<p>Links</p> <hr/>  <p><i>Institute for Operations Research and the Management Sciences</i></p> <p><u>The VRP Web</u></p>
---	--	--

To test the model and performance of the algorithms in our implementation, many jobs have been generated. Their sources, destinations and the distance between every two points in the port have been chosen randomly. As it can be seen in Figure web-1, our software, which has been implemented by C++, running on 2.4 GHz Pentium PC, can find the global optimal solution for 3,000 jobs within two minutes.

⁶ <http://privatewww.essex.ac.uk/~hrashi>

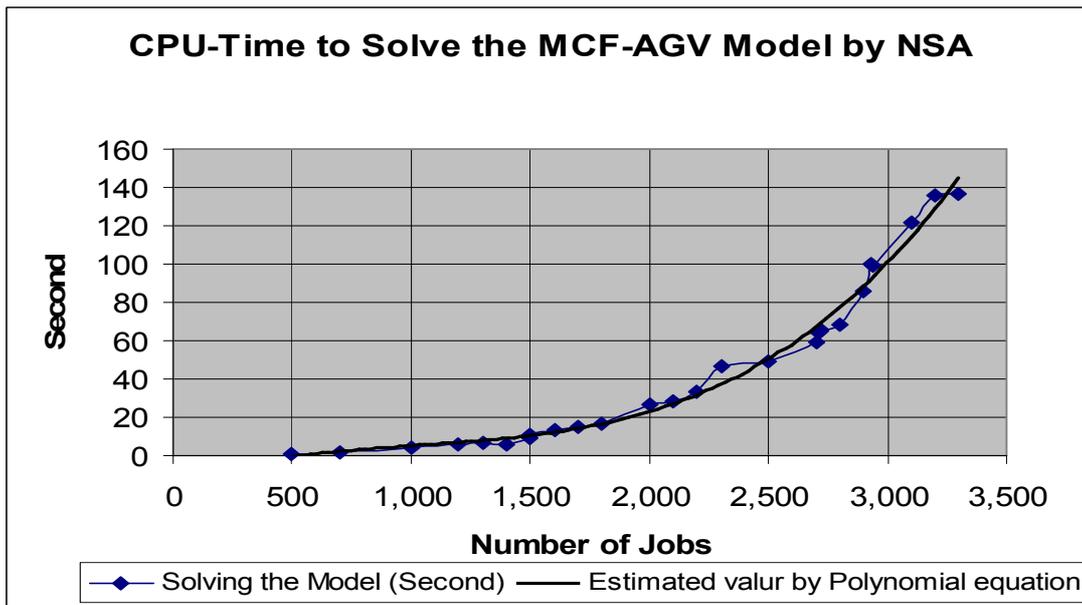


Figure web-1: CPU-Time required to solve the graph model

Overview of this research:

This research concerns itself with the scheduling of Autonomous Guided Vehicles (AGV's) in a port. Port components that are relevant to our problem include berths, Quay Cranes (QC), container storage areas, and a road network. A transportation requirement in a port is described by a set of jobs. Each job is described by (a) the source location of a container; (b) the target location, where the container is to be delivered to; and (c) the time at which it is available for pick-up or drop-off on the quay-side. Any delay will incur heavy penalties. Given a number of AGVs and their availability, the task is to schedule the AGVs to meet the transportation requirements.

Assumptions:

- The layout of a port container terminal is given. Also it is assumed the vehicles move with an average speed so that there are no *Collisions, Congestion, Live-locks and Deadlocks problem*.
- The travel time between every combination of pick-up /drop-off points is provided according to our layout.
- Every AGV can transport only one container. Also it is assumed that the start location of each AGV at the beginning of the process is given.

- Rubber Tyred Gantry Cranes or yard crane resources are always available, i.e., the AGVs will not suffer delays in the storage yard location or waiting for the yard cranes.
- The source and destination of container jobs over the port are given.
- For each QC, there is a predetermined crane job sequence, consisting of loading jobs, or unloading/discharging jobs, or a combination of both. For each loading (discharging) job, there is a predetermined pickup (drop-off) point in the yard, which is the origin (destination) of the job.
- Appointment time of every container job at its source (destination) on the quay side is given.
- For the dynamic aspect of the problem, it is assumed that the number of vehicles is fixed, but the number of jobs and the distance between every two points in the port may be changed.

Development:

Our software consists of the optimisation, scheduling and a simulation program. The software can find the global optimal solution for 3,000 jobs and ten millions arcs in the graph model within 2 minutes by 2.4 GHZ Pentium processor on PC. Figure web-2 shows the main form of the software.

Some important features of our program are described briefly in the following sections:

- The user can define a few ports, the number of blocks in the yard, the number of working positions or crane and the number of Automated Guided Vehicles in each port.
- A facility to generate the distance between different points in the yard or in the berth has been considered. At the first step, this distance is generated randomly, but it is modifiable by the user.
- For static and dynamic fashion, a few container jobs might be generated, which have to be transported from their sources to their destinations. Either the source or the destination of them is the quay side, which is chosen randomly by the Job-Generator. There are three options for quay cranes: single crane and multiple cranes randomly and circular. In the first option, crane number 1 is selected to handle the job whereas in the second option one crane, among several cranes in the berth, is determined to handle the job. In the latter option, choosing the crane number is circular; the first job for the first crane, second job for the

second crane and so on. After the next job is assigned to the last crane, the turn goes to the first crane.

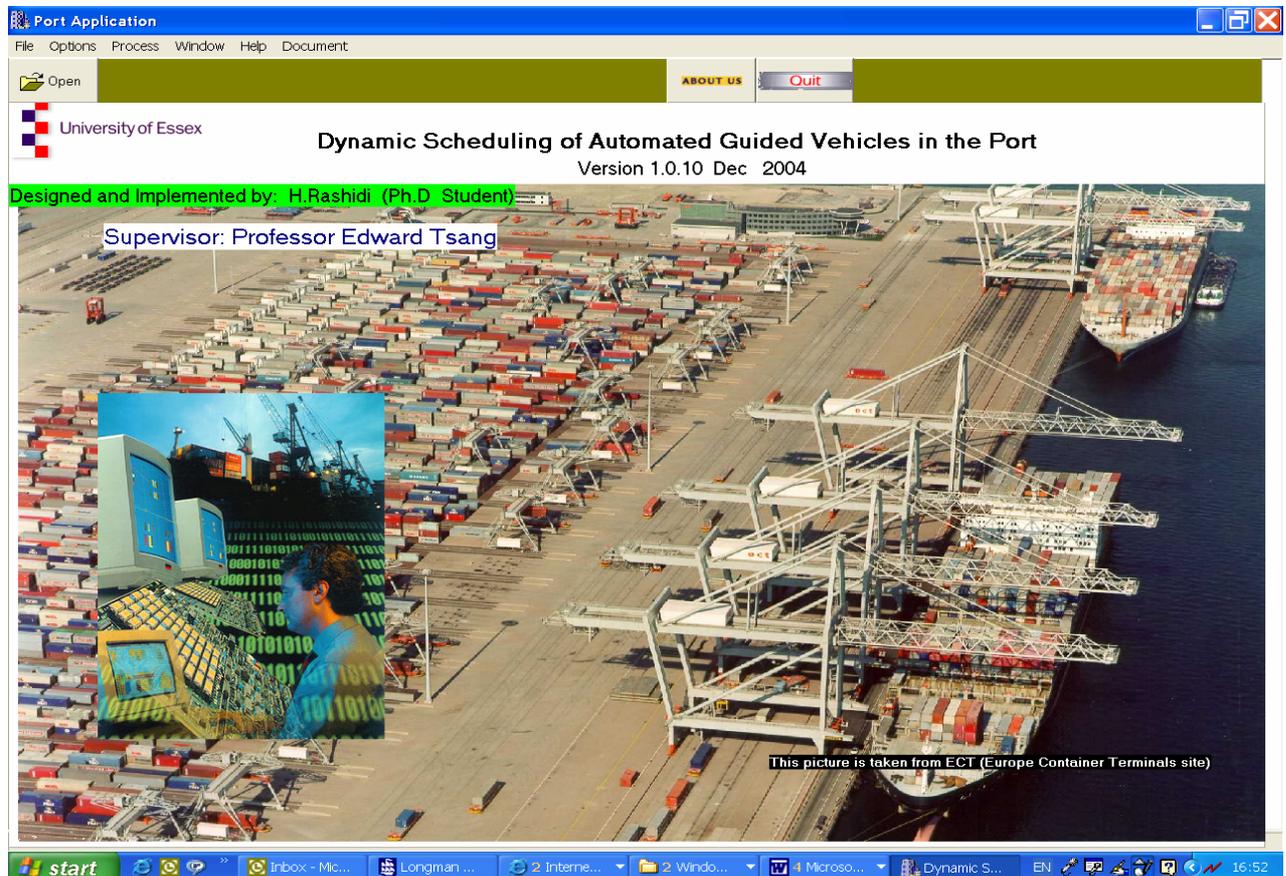


Figure web-2: The main form of the software

- At the start of the process, the start location of each vehicle may be any point in the port. The user can define or change the ready time of the vehicles at the start location and the location as well. But at the first stage, we generate them randomly.
- The initial time for the operation, the time window of the cranes and vehicles should be defined by the user. The first parameter plays a role as the ship-arrival time; the second one means how long it takes for every job to be picked-up or dropped-off by the crane. The last one is the time for the vehicle to pick-up or drop-off a job from/to the crane. We assume some defaults values for these parameters.
- The use can monitor some indices to measure the efficiency of the terminal. The waiting or delay time for every job, the number of jobs and the travelling and waiting times for every vehicle are calculated in the static and dynamic fashion.

Some interfaces of our Software:

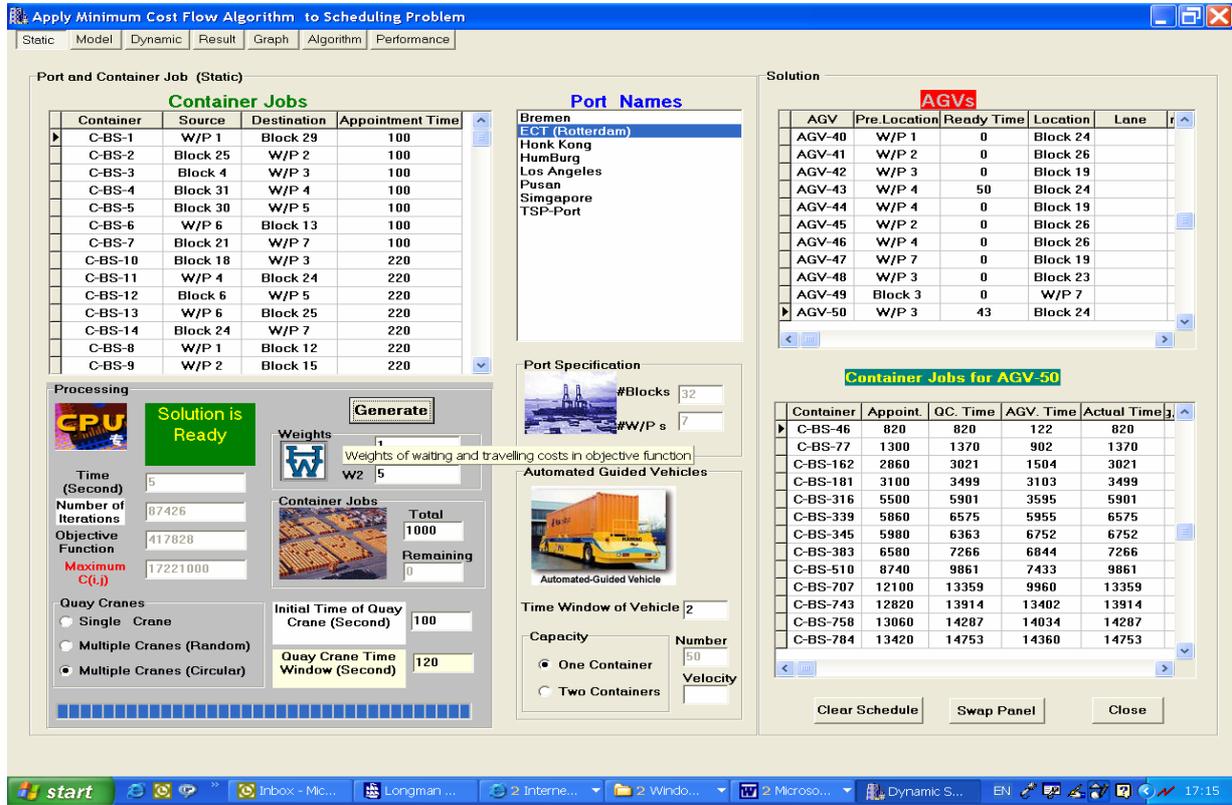


Figure web-3: The output of Static fashion.

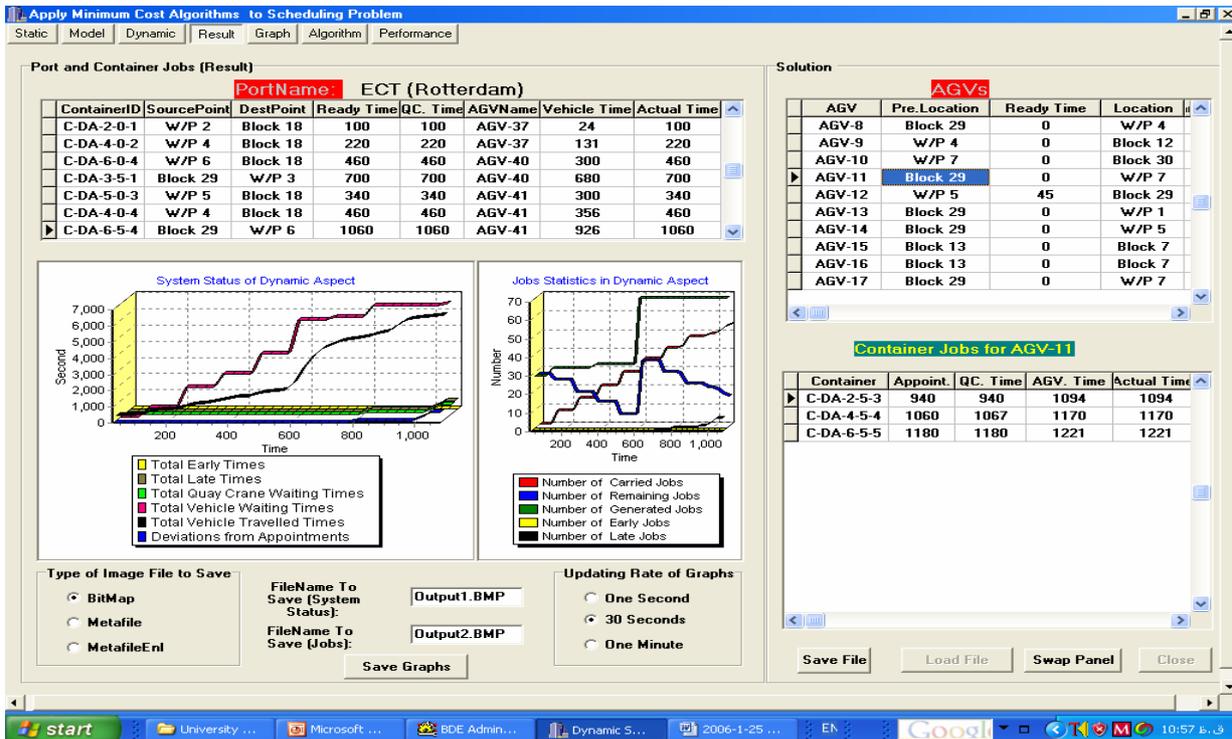


Figure web-4: Monitoring some indicators of the output in Dynamic aspect

References

1. Aggarwal C.C, Kaplan H, Tarjan R.E., “A Faster Primal Network Simplex Algorithm”, Massachusetts Institute of Technology, Operations Research Centre, Working Paper; OR 315-96, 1996.
2. Ahuja R.K, Magnanti T.L, Orlin J. B, “Network Flows: Theory, Algorithms and Applications”. Prentice Hall. 1993.
3. Ahuja R.K, Orlin J.B, Sharma P, Sokkalingam P.T, “A network simplex algorithm with $O(n)$ consecutive degenerate pivots”. Operations Research Letters, Volume 30(3), pp 141-148, 2002.
4. Akturk M.S, Yilmaz M, "Scheduling of Automated Guided Vehicles in a Decision Making Hierarchy," International Journal of Production Research, Volume 34, pp 577-591, 1996.
5. Ambrosino D, Marina M.E, Sciomachen A, “Decision rules for the yard storage management”, University of Genova, Technical Report, Italy, 2002
6. Andrew V.G, “An efficient implementation of a scaling minimum-cost flow algorithm”. Journal of Algorithms, Volume 22(1), pp 1-29, January 1997.
7. Blazewicz J, Machowiak M, Edwin Cheng T.C, Qğuz C, “On a certain Berth Scheduling Problem”, Proceedings of the 2nd Multidisciplinary International conference on Scheduling, Theory and Applications (MISTA), Volume 2, pp 694-697, 2005.
8. Böse J, Reiners T, Steenken D, Voß S, “Vehicle Dispatching at Seaport Container Terminals Using Evolutionary Algorithms”. Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, IEEE, Piscataway, pp 1-10, 2000.
9. Burke E., Hart E., Kendall G., Newall J., Ross P. and Schulenburg S. “Hyper-Heuristics: An Emerging Direction in Modern Search Technology”, Handbook of Meta-Heuristics, pp 457 – 474, 2003.
10. Burke E.K., Kendall G. and Soubeiga E. “A Tabu-Search Hyper-Heuristic for Timetabling and Rostering”, Journal of Heuristics, Volume 9(6), pp 451-470, 2003.
11. Carre B., “Graphs and Networks”. Oxford University Press, Oxford, UK, 1979.

12. Cave A, Nahavandi S, Kouzani A, "Simulation optimization for process scheduling through simulated annealing". Proceedings of the 2002 Winter Simulation Conference, pp 1909–1913, 2002.
13. Chan S.H, "Dynamic AGV-Container Job Deployment", Master of Science, University of Singapore, 2001.
14. Cheng Y, Sen H, Natarajan K, Teo C, Tan K, "Dispatching automated guided vehicles in a container terminal", Technical Report, National University of Singapore, 2003.
15. Chiang W.C, Russell R.A, "Simulated Annealing Metaheuristics for the Vehicle Routing Problem with Time Windows". Annals of Operations Research, Volume 63, pp 3-27, 1996.
16. Chowdhury M.S, Chein S.I, "Dynamic Vehicle Dispatching at inter-modal Transfer station", Transportation research board, 80th Annual meeting, Washington, 2001.
17. Christiansen M, Fagerholt K, Ronen D, "Ship Routing and Scheduling - Status and Trends", Norwegian University of Science and Technology, Trondheim, Norway. University of Missouri, USA. Accepted for publication in Transportation Science, 2003.
18. Chuanyu C, "Simulation and optimization of container yard operation: A survey", Technical Report, Nanyang Technological University, Singapore, 2003.
19. Cowling P I, Ouelhadj D, Petrovic S. "A Multi-agent Architecture for Dynamic Scheduling of Steel Hot Rolling", Journal of Intelligent Manufacturing, Volume 14(5), pp 457-470, 2003.
20. Cowling P, Ouelhadj D. and Petrovic S, "Dynamic scheduling of steel casting and milling using multi-agents", Production Planning and Control, Volume 15, pp 1-11, 2004
21. Czech Z.J, Czarnas P, "Parallel simulated annealing for the vehicle routing problem with time windows", 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, Canary Islands - Spain, pp 376-383, 2002.
22. Dondo R, Mndez C.A, Cerd J, "An optimal approach to the multiple-depot heterogeneous vehicle routing problem with time window and capacity constraints", Latin American Applied Research, Volume 33, pp 129-134, 2003.

23. Duinkerken M.B, Ottjes J.A, "A simulation model for automated container terminals". Proceedings of the Business and Industry Simulation Symposium (ASTC2000). April 2000.
24. Eppstein D, "Clustering for faster network simplex pivots", Proceeding of 5th ACM-SIAM Symposium. Discrete Algorithms, pp 160–166, 1994.
25. Eppstein D, Galil Z, Italiano G.F, "Dynamic graph algorithms", In CRC Handbook of Algorithms and Theory, chapter 22. CRC Press, 1997.
26. Fink A, Voß S, "HOTFRAME: A Heuristic Optimisation Framework", Optimisation Software Class Libraries, Kluwer, Boston, pp 81-154, 2002.
27. Gambardella L.M, Rizzoli A.E. Zaffalon M, "Simulation and Planning of an Intermodal Container Terminal", Simulation, Volume 71(2), pp 107-116, 1998.
28. Gebraeel N.Z, Lawley M.A, "Deadlock Detection, Prevention, and Avoidance for Automated Tool Sharing Systems", IEEE Transactions on Robotics and Automation, Volume 17 (3), pp 342-356, June 2001.
29. Goldberg A.V, Kennedy R, "An Efficient Cost Scaling Algorithm for the Assignment Problem". Technical Report, Stanford University, 1993.
30. Gribkovskaia I, Halskau O, Bugge M, Kim N, "Models for Pick-Up and Deliveries from Depots with Lasso Solutions". Working Paper, Molde University College, Norway, 2002.
31. Grunow M, Günther H.O, Lehmann M: "Dispatching multi-load AGVs in highly automated seaport container terminals", OR Spectrum, Volume 26 (2), pp 211-235, 2004.
32. Gunadi W.N, Rose A.A, Shamsuddin S.M, Mohd N.M. "Vehicle Routing Problem For Public Transport: A Case Study". Proceeding Of International Technical Conference on Circuits/Systems, Computers and Communications, Volume 2, pp. 1180-1183, 2002.
33. Hansen P, Oguz C, "A Note on formulation of the Static and Dynamic Berth Allocation Problems", Technical Report, Department of Management, Hong Kong Polytechnic University, 2003.
34. Hartmann S, "Generating scenarios for simulation and optimisation of container terminal logistics". Working paper 564, University of Kiel, Germany. 2002.

35. Hasama T, Kokubugata H, Kawashima H, "A Heuristic Approach Based on the String Model to Solve Vehicle Routing Problem with Backhauls", Proceeding of the 5th World Congress on Intelligent Transport Systems (ITS), Seoul, 1998.
36. Helgason R, Kennington J, "Primal Simplex Algorithms for Minimum Cost Network Flows," Handbook on Operations Research and Management Science, Volume 7, Amsterdam, pp 85-133, 1995.
37. Henesey L, Wernstedt F, Davidsson P, "Market-Driven Control in Container Terminal Management ", 2nd International Conference on Computer Applications and Information Technology in the Maritime Industries, 2003.
38. Henry Y. K. L, Ying Z, Chuanyou P, "Integrated Scheduling of different Types of Handling Equipment at Automated Container Terminals", Proceedings of the 2nd Multidisciplinary International conference on Scheduling, Theory and Applications (MISTA), Abstract paper, Volume 2, pp 536-537, 2005.
39. Hollingworth J, Gustavson P, Swart B, Cashman M., "Borland C++Builder 6 Developer's guide", Sams Publishing, 2003.
40. ILOG optimisation suite- White papers. Available via <http://www.ilog.com>. Last check of the address: 3 July 2005.
41. Indra-Payoong N, Kwan R.S.K, Proll L.G. "Constraint-Based Local Search for Rail Container Service Planning", Proceeding of the MISTA Conference, Nottingham, August 2003.
42. Ioannou P, Chassiakos A, Jula H, Unglaub R, "Dynamic optimization of cargo movement by trucks in metropolitan areas with adjacent ports", Metrans Technical Report, Center for Advanced Transportation Technologies, University of Southern California, June 2002.
43. Ioannou P.A, Jula H, Liu C.I, Vukadinovic K, Pourmohammadi H. "Advanced Material Handling: Automated Guided Vehicles in Agile Ports", CCDoTT Technical Report, Center for Advanced Transportation Technologies, University of Southern California, Jan. 2001.
44. Ioannou P.A, Kosmatopoulos E.B, Vukadinovic K, Liu C.I, Pourmohammadi H, Dougherty E, "Real time testing and verification of loading and unloading algorithms using Grid Rail (GR)", Center for Advanced Transportation Technologies, University of Southern California, Los Angeles, Technical Report, Oct. 2000.
45. Iris F.A. Vis, "<http://www.ikj.nl/container/>", Last check of the address: 3 July 2005.

46. István M, "A General Pricing Scheme for the Simplex Method", Department of Computing, Imperial College, Technical Report, London, 2001-3
47. Jianyang Z, Wen-Jing H, Yee V.V, "An AGV-Routing Algorithm in the Mesh Topology with Random Partial Permutation", Centre for Advanced Information Systems, School of Computer Engineering, Nanyang Technological University, Singapore, Technical Report, Singapore, 2003.
48. Kelly D.J, O'Neill G.M, "The Minimum Cost Flow Problem and The Network Simplex Solution Method", Master Degree Dissertation, University College, Dublin, 1993.
49. Kendall G. and Mohd Hussin N. "An Investigation of a Tabu-Search-Based Hyper-heuristic for Examination Timetabling, Multidisciplinary Scheduling; Theory and Applications", Springer, pp 309-328, 2005.
50. Kilby P, Prosser P, Shaw P, "Guided local search for the vehicle routing problem", Proceeding of 2nd International Conference on Metaheuristics - MIC97, Sophia-Antipolis, France, July 1997.
51. Kim K.H, Won S.H, Lim J.K, Takahashi T, "A simulation-based test-bed for a control software in automated container terminals", Department of Industrial Engineering, Pusan National University, Technical report, Pusan, 2000.
52. Kozan E, Wong A, "An Optimisation model for export and import container process in seaport terminals", 25th Australasian Transport Research Forum, Canberra, CD-ROM, 2002.
53. Lau H.C, Liang Z, "Pickup and Delivery with Time Windows : Algorithms and Test Case Generation", 13th IEEE International Conference on Tools with Artificial Intelligence, ICTAI-2001, Dallas, USA, pp 333-340. 2001.
54. Lau T.L, Tsang E.P.K, "Guided genetic algorithm and its application to radio link frequency assignment problems", Journal of Constraints, Volume 6, pp 373-398, 2001.
55. Lim A, Rodrigues B, Zhu Y, "Crane scheduling using squeaky wheel optimization with local search". Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning, Singapore, 2002.
56. Lin W, "On Dynamic Crane Deployment in Container Terminals", Master of Philosophy in industrial engineering and engineering management, University of Science & Technology, Hong Kong, Jan. 2001.

57. Liu C.I, Jula H, Ioannou P.A, “Design, simulation, and evaluation of automated container terminals,” *IEEE Trans. on Intelligent Transportation Systems*, Volume 3(1), pp 12–26, 2002.
58. Löbel A., “MCF: A Network Simplex Implementation”, Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), Technical Report, 2000.
59. Marco E. L Ubbecke, “Combinatorial Simple Pickup and Delivery Paths”, *Central European Journal of Operations Research*, 2003.
60. Meersmans P.J.M, Dekker R, “Operations research supports container handling”, Technical Report EI 2001-22, Erasmus University of Rotterdam, Econometric Institute, 2003.
61. Meersmans P.J.M, Wagelmans A.P.M, “Dynamic scheduling of handling equipment at automated container terminals”, Technical Report EI 2001-33, Erasmus University of Rotterdam, Econometric Institute, 2001.
62. Meersmans P.J.M, Wagelmans A.P.M, “Effective algorithms for integrated scheduling of handling equipment at automated container terminals”. Technical Report EI 2001-19, Erasmus University of Rotterdam, Econometric Institute, 2001.
63. Mitrovic-Minic S, “Pickup and delivery problem with time window: A survey. Technical Report 1998-12, School of Computing Science, Simon Fraser University, Burnaby, BC, Canada, May 1998.
64. Moin N.H, “Hybrid Genetic Algorithms for Vehicle Routing Problems with Time Windows”, submitted to *Computers & Operations Research*, 2002.
65. Moon K.C, “A Mathematical Model and a Heuristic Algorithm for Berth Planning”, *Industrial Engineering / Pusan National University, Telecommunication Grooming*, Volume 2 (3), May/June 2001.
66. Moorthy R.L, Hock-Guan W, Wig-Cheong N, Chung-Piaw T, “Cyclic deadlock prediction and avoidance for zone controlled AGV system”. *International Journal of Production Economics*, Volume 83, pp 309-324, 2003.
67. Muramatsu M, "On network simplex method using primal-dual symmetric pivoting rule", *Journal of Operations Research of Japan*, Volume 43, pp 149-161, 2000.
68. Murty K.G, Liu J, Wan Y.W, Linn R.J, “A Decision Support System for operations in a container terminal”. *Decision Support System*, Volume 39, pp 309-332, 2005.
69. Nuhut Ö, “Scheduling of Automated Guided Vehicles”, Technical Report, Department of Industrial Engineering, Bilkent University, 1999.

70. Orlin J.B, "A Polynomial Time Primal Network Simplex Algorithm for Minimum Cost Flows (An Extended Abstract)", *Mathematical Programming 78 Series B*, pp 109-129, 1996.
71. Ouelhadj, D., Cowling, P., Petrovic, S., "Contract Net Protocol for Cooperative Optimisation and dynamic Scheduling of Steel Production", published in the Book of *Intelligent Systems Design and applications*, Springer-Verlag, pp 457-470, 2003.
72. Ouelhadj, D., Petrovic, S., Cowling, P. and Meisels, A., "Inter-agent cooperation and communication for agent-based robust dynamic scheduling in steel production", Accepted for publication in *Advanced Engineering and Informatics (Artificial Intelligence in Engineering)*, 2005.
73. Park Y.M, Kim K.H, "A scheduling method for Berth and Quay cranes", *OR Spectrum*, Volume 25, pp 1–23, 2003.
74. Petrovic S., Fayad C. and Petrovic D., "Job Shop Scheduling with Lot-Sizing and Batching in an Uncertain Real-World Environment", *The 2nd Multidisciplinary Conference on Scheduling: Theory and Applications*, pp 363-379, 2005.
75. Petrovic, S., Fayad, C., "A Fuzzy Shifting Bottleneck Hybridised with Genetic Algorithm for Real-world Job Shop Scheduling", *Proceedings of Mini-EURO Conference, Managing Uncertainty in Decision Support Models*, Coimbra, Portugal, pp 1-6, 2004.
76. Qiu L, Hsu W.J, "A bi-directional path layout for conflict-free routing of AGVs". *International Journal of Production Research*, Volume 39 (10), pp 2177-2195, 2001.
77. Qiu L, Hsu W.J, "Conflict-free AGV routing in a bi-directional path layout", *Proceedings of the 5th International Conference on Computer Integrated Manufacturing*, Volume 1, pp 392-403, Singapore, 2000.
78. Qiu L, Hsu W.J, "Scheduling of AGVs in a mesh-like path topology". *Technical Report CAIS-TR-01-34*, Centre for Advanced Information Systems, School of Computer Engineering, Nanyang Technological University, Singapore, July 2001.
79. Qiu L, Hsu W.J, Huang S.Y, Wang H. "Scheduling and Routing Algorithms for AGVs: a Survey". *International Journal of Production Research*, Taylor & Francis Ltd, Volume 40 (3), pp 745-760, 2002.
80. Qiu L, Hsu W.J, "Algorithms for routing AGVs on a mesh topology", *Proceedings of the 6th European Conference on Parallel Computing (Euro-par 2000)*, pp 595-599, Munich, Germany, 2000.

81. Qiu L, Hsu W.J. "Routing AGVs by sorting". Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000), Volume 3, pp 1465-1470, Las Vegas, Nevada, USA, 2000.
82. Rauch M, "Fully dynamic graph algorithms and their data structures". PhD thesis, Department of computer science, Princeton University, 1992.
83. Ravindra K.A, Thomas L.M, James B.O, Giovanni M.S, Zuddas P, "Algorithms for the simple equal flow problem", Management Science, Volume 45(10), pp 1440-1455, 1999.
84. Rebollo M, Julián V, Carrascosa C, Botti V, "A Multi-Agent System for the Automation of a Port Container Terminal", Workshop in Agents in Industry. Barcelona, 2000.
85. Seifert R.W, Kay M.G., Wilson J.R., "Evaluation of AGV routing strategies using hierarchical simulation", International Journal of Production Research, Volume 36 (7), pp 1961–1976, 1998.
86. Shih L.H, Chang H.C, "A routing and scheduling system for infectious waste collection". Environmental Modelling & Assessment, Volume 6, pp 261-69, 2001.
87. Steenken D, Vob S, Stahlbock R, "Container Terminal Operation and Operations Research- a classification and literature review", OR Spectrum, Volume 26, pp 3-49, 2004.
88. Steenken D, Winter T, Zimmermann U.T, "Stowage and transport optimisation in ship planning", Springer, Berlin, pp 731-745, 2001.
89. Tan K.C, Lee L.H, Zhu Q.L, Ou K, "Heuristic Methods for Vehicle Routing Problem with Time Windows", Artificial Intelligent in Engineering, pp 281-295, 2000.
90. Thurston T, Hu H, "Distributed Agent Architecture for Port Automation", Proceedings of the 26th Annual International Computer Software and Applications Conference, Oxford, England, August 2002.
91. Toth P, "The Vehicle Routing Problem Discrete Math", SIAM (Society for Industrial and Applied Mathematics) Press, 2003.
92. Tsang E.P.K, Wang C, Davenport A, Voudouris C, Lau T, "A Family of Stochastic Methods for Constraint Satisfaction and Optimisation", Proceedings of the First International Conference on the Practical Application of Constraint Technologies and Logic Programming (PACLP'99), London, pp 359-383, April 1999.

93. Tsang E.P.K., "Foundations of constraint satisfaction", Academic Press, London, 1993.
94. Tsang E.P.K., "Scheduling techniques -- a comparative study", British Telecom Technology Journal, Volume 13 (1), pp 16-28, Martlesham Heath, Ipswich, UK, 1995.
95. Tsang E.P.K.: "Spatio-Temporal Conflict Detection and Resolution", Constraints, Volume 3(4), pp 343-361, 1998.
96. Voudouris C, Tsang E.P.K, "Guided local search and its application to the travelling salesman problem", European Journal of Operational Research, Volume 113 (2), pp 469-499, March 1999.
97. Voudouris C, Tsang E.P.K., "Guided local search joins the elite in discrete optimisation", Proceedings of DIMACS Workshop on Constraint Programming and Large Scale Discrete Optimisation, Rutgers, New Jersey, USA, September 1998.
98. Voß S, "Meta-heuristic: the state of the art", Local Search for Planning and Scheduling: ECAI 2000 Workshop, Berlin, Germany, August 2000.
99. VRP Web, "<http://neo.lcc.uma.es/RADI-AEB/WebVRP/links.html>", Last check of the address: 3 July 2005.
100. Wayne K.D, "A polynomial combinatorial algorithm for generalized minimum cost flow". Proceedings of the 31st Annual ACM Symposium on Theory of Computing, pp 11-18, 1999.
101. Weber R, "Mathematics for Operational Research", Lecture Notes, Department of Pure Mathematics and Mathematical Statistics, University of Cambridge, 2003.
102. Westbrook J, "Algorithms and data structures for dynamic graph problems", PhD Thesis, Princeton University, 1989.
103. Wook B.J, Hwan K.K, "A pooled dispatching strategy for automated guided vehicles in port container terminals", International Journal of management science, Volume 6(2), pp 47-67, 2000.
104. Yokoo M, Durfee E.H, Ishida T, Kuwabara K, "The distributed constraint satisfaction problem: Formalization and algorithms". IEEE, Transaction on Knowledge and Data Engineering, Volume 10(5), pp 673-685, 1998.
105. Yokoo M, Hirayama K, "Algorithms for Distributed Constraint Satisfaction: A Survey", Autonomous Agents and Multi-Agent Systems, Volume 3(2), pp 198-212, 2000.

References

106. Zhang C, Liu J, Wan Y.W, Murty K.G, Linn R.J, “Storage space allocation in container terminals”. *Transportation Research B* 37, 2001.
107. Zhang C, Wan Y, Liu J, Linn R.J, “Dynamic Crane Deployment in Container Storage yard”. *Transportation Research B* 36. 2002.
108. Zhang L.W, Ye R, Huang S.Y, Hsu W.J: “Two Equivalent Integer Programming Models for Dispatching Vehicles at a Container Terminal”. School of Computer Engineering, Nan yang Technological University, Technical Report, Singapore, 2002.

Index

- AGV**, 8, 49, 51, 52, 53, 54, 55, 57, 62, 65, 66, 83, 86, 112, 120, 127, 139, 144, 155, 158, 160
- Automated Guided Vehicle, 8, 31, 32, 36, 48, 49, 51, 52, 53, 54, 55, 57, 61, 62, 63, 64, 65, 66, 83, 85, 86, 112, 120, 126, 127, 128, 139, 144, 145, 148, 149, 155, 158, 159, 160, 161
- BDE**
 - Borland Database Engine, 77, 79
- Branch and Bound**, 39, 41, 43
- Complexity**, 99, 123, 134, 136, 141
- CSOPs**
 - Constraint Satisfaction Optimisation Problems, 3, 12, 14, 49, 143
- Dijkstra**, 43
- DNSA**
 - Dynamic Network Simplex Algorithm**, 4, 106, 112, 120, 121, 122, 123, 124, 138, 140, 141, 142, 143, 148
- DNSA+**
 - Dynamic Network Simplex plus Algorithm**, 4, 106, 112, 121, 122, 123, 124, 138, 140, 141, 142, 143, 148
- DSAGV**
 - Dynamic Scheduling of Automated Guided Vehicles, 138
- DSSAGV**
 - Dynamic Scheduling Software for Automated Guided Vehicles, 77, 78, 94
- ERD**
 - Entity Relationship Diagram, 79
- Expert Systems**, 41
- Genetic Algorithms**, 39, 42, 44, 159
- Graph**
 - Node Potential**, 58, 61, 79, 124, 143
 - Optimality Conditions**, 58, 61, 79, 124, 143
 - Reduced Cost**, 58, 61, 79, 124, 143
 - Spanning Tree**, 58, 61, 79, 124, 143
 - Strongly Feasible Basic**, 58, 61, 79, 124, 143
- GVS**
 - Greedy Vehicle Search**, 4, 105, 125, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 140, 141, 142, 143
- Hill Climbing**, 41
- HOTFRAME**, 41, 156
- ILOG**, 41, 157
- Integer Programming**, 39, 163
- IT**
 - Internal Trucks., 8, 10, 36, 49, 154
- MCF**
 - Minimum Cost Flow, 50, 58, 59, 60, 61, 63, 64, 65, 66, 67, 68, 69, 81, 85, 87, 89, 90, 101, 102, 111, 125, 126, 127, 128, 138, 139, 141, 143, 159
- MCF-AGV**
 - Minimum Cost Flow model for Scheduling problem of AGVs, 61, 64, 65, 66, 87, 89, 90, 101, 102, 111, 125, 126, 127, 128, 139, 141, 143
- MILP**
 - Mixed Integer Linear Program, 13, 26, 31, 39

NSA

Network Simplex Algorithm, 3, 4, 67, 70, 71, 73, 75, 81, 84, 85, 87, 89, 90, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 103, 105, 106, 112, 121, 122, 123, 124, 125, 131, 132, 133, 134, 135, 137, 138, 139, 140, 141, 142, 143, 148

NSA+

Network Simplex Plus Algorithm., 3, 4, 93, 94, 95, 96, 97, 98, 99, 100, 101, 103, 105, 106, 112, 121, 122, 123, 124, 125, 131, 132, 133, 134, 135, 137, 138, 139, 140, 141, 142, 143, 148

OSA

Original Simplex Algorithm, 73

PSCDS

Primary Storage Containers Discharge, 20, 22, 23, 24, 25

PSCPI

Primary Storage Containers Pickup, 20, 22, 23, 24

PSCSS

Primary Storage Containers to Secondary Storage, 19, 22, 23, 24, 25

QC

Quay Cranes., 7, 9, 10, 31, 32, 36, 48, 52, 54, 55, 149, 150

RTGC

Rubber Tyred Gantry Cranes, 6, 7, 10, 11, 26, 27, 28, 29, 30, 32, 36, 39, 49, 146

SAM

Simulated Annealing Method, 41, 42, 44, 136

SC

Straddle Carrier., 8

SDSAGV

Static and Dynamic Scheduling of Automated Guided Vehicles, 138

Simulation, 39, 77, 155, 156**SSCGD**

Secondary Storage Container Grounding, 19, 20, 22, 23, 24, 25, 37, 38

SSCPI

Secondary Storage Container Pickup, 19, 20, 22, 23, 24, 25, 37, 38

SSCPS

Secondary Storage Containers to Primary Storage, 19, 22, 23, 24, 25, 38

Tabu Search, 41, 42, 43, 44**TG**

Terminal Gate, 36

TSS

Taxi Service System, 129

VRP

2-Opt Exchange, 30, 33, 43, 148, 162

Vehicle Routing Problem, 30, 33, 43, 148, 162

VRPTW

Vehicle Routing Problem with Time Window, 30, 31

XT

eXternal Truck., 8, 11, 36