Proceedings, 2nd Multidisciplinary Conference on Scheduling (MISTA), New York, USA, 18-21 July 2005, 677-692

Applying the Extended Network Simplex Algorithm to Dynamic Automated Guided Vehicles Scheduling

Hassan Rashidi
Department of Computer Science
University of Essex, Colchester CO4 3SQ, U.K.
Email: hrashi@essex.ac.uk

Edward P. K. Tsang
Department of Computer Science
University of Essex, Colchester CO4 3SQ, U.K.
Email: edward@essex.ac.uk

Abstract: Scheduling of vehicles in the container terminal is often studied as a static problem in the literature, where all information, including the number of jobs, their arrival time, etc., is known beforehand. The objective is generally minimizing the total travelling and/or waiting times of the vehicles. When the situation changes, for example new jobs arrive or a section of the terminal is blocked, new solutions are generated from scratch. In this paper, the problem of dynamically scheduling automated guided vehicles in the container terminals is formulated as a minimum cost flow problem. This problem is then solved by a novel algorithm, NSA+, which extended the standard Network Simplex 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. Our software, DSAGV, employs NSA+ and can find the global optimal solutions for 3,000 jobs and 10 millions arcs in the graph model within 2 minutes on a 2.4 GHz Pentium PC. To complement NSA+, an incomplete algorithm Greedy Heuristic Search (GHS) is designed and implemented. To evaluate the relative strength and weakness of NSA+ and GHS, the two algorithms are compared for the dynamic automated vehicle scheduling problem. Both NSA+ and GHS are practical algorithms for dynamic automatic vehicle scheduling.

Keywords: Search Methods, Dynamic Scheduling, Network Simplex Algorithm, Optimization, Container Terminals.

1. Introduction

The Minimum Cost Flow (MCF) Problem is to flow resources from a set of supply nodes, through the arcs of a network, to a set of demand nodes at minimum total cost, without violating the lower and upper bounds on flows through the arcs (which represent the capacities of the arcs). The MCF problem has numerous applications in scheduling, transportation,

logistics, and telecommunication. This paper has been motivated by a need to schedule a number of Automated Guided Vehicles (AGVs) in the container terminals. The container terminals components that are relevant to our problem include Quay Cranes (QC), container storage areas, Rubber Tyred Gantry Crane (RTGC) or yard crane, and a road network [see e.g. 1, 2, 4, 5, 7, 10]. A transportation requirement in a port is described by a set of jobs, where each job is characterized by the source location of a container, the target location and the time of its picking up or dropping-off on the quay side by the quay crane. Given a number of AGVs and their availability, the task is to schedule the AGVs to meet the transportation requirements.

The Network Simplex Algorithm (NSA) is an adaptation of the bounded variable of traditional primal simplex algorithm in Linear Programming. Generally, NSA is employed to solve MCF problems. The basis of NSA is represented as a rooted spanning tree of the underlying network, in which variables are represented by arcs. The method iterates towards an optimal solution by exchanging basic and non-basic arcs. At each iteration, an entering arc is selected by some pricing strategy, and the arc to leave the basis is ascertained. The construction of a new basis tree is called pivoting. There are many strategies for selecting the entering arc. The performance of algorithm is depended on those strategies. This paper extends NSA with some modification to the NSA. The novel algorithm, NSA+, then is applied to dynamic automated guided vehicles scheduling.

The structure of this paper is as follows. Section 2 is a description of MCF problem and a literature review over NSA. After that features of NSA+ are presented. In section 3, an application of NSA+ for dynamic scheduling of automated guided vehicles in the container terminal is presented. Then, the problem is formulated and solved by NSA+. Also some outputs of running NSA+ to solve static problems are given in this section. Section 4 presents a Greedy Heuristic Search (GHS) and compares results of NSA+ and GHS in dynamic aspect. Section 5 is considered to summary and conclusion.

2. Minimum cost flow problem and algorithms (NSA, NSA+)

In this section some preliminary definitions of graph and minimum cost flow problem are briefly explained. After that a literature review over NSA and main features of NSA+ are presented.

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

Definition 2: In an *undirected graph* the arcs are unordered pairs of nodes {i, j} Є A, i, j Є N. In a *directed graph* or network the arcs are ordered pairs of nodes (i, j).

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

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

Definition 5: 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' ⊂ N and A' ⊂ A. A sub-network (N', A') is a spanning tree if it is a tree and N' = N (figure 1).

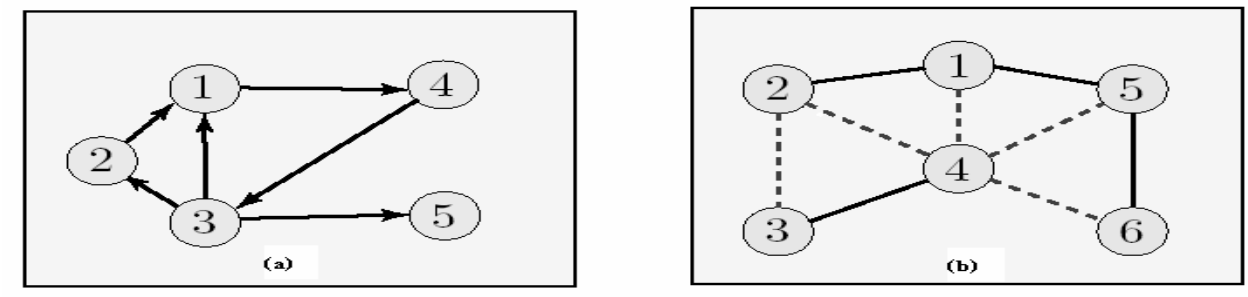

Figure 1: A connected Graph (a) and a spanning Tree (b) as dotted

2.1 Minimum cost flow problem

Given a directed network graph, let f_{ij} denote the amount of flow of some material on arc (i, j) Є A and b_i, i Є N, denote the amount of flow that enters the network at node i. If $b_i > 0$, the node is a source that supplies b_i units of flow. If $b_i < 0$, the node is a sink with a demand of $|b_i|$ units of flow. Let there be a cost of c_{ij} per unit flow on arc (i, j) ЄA. The *minimum cost flow* (MCF) *problem* is:

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

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

These constraints show that the flows must be feasible and conserve each node. For feasible flows to exist, the MCF problem must also have $\sum_{i \in N} b_i = 0$.

Note that the MCF problem is a special form of Linear Program. The constraints of a Linear Program are of the form Ax = b, where

$$(A)_{ik} = \begin{cases} +1 \ node \ \ i \ is \ \ start \quad of \ k^{th} \ arc; \\ -1 \ node \ \ i \ is \ \ end \quad of \ k^{th} \ arc; \\ 0 \quad otherwise \end{cases}$$

2.2 Spanning tree solutions and optimality conditions

In the network simplex algorithm, it is assumed that the network is connected. Every connected network has a spanning tree. A spanning tree solution, T, is a tree that can be constructed as follows:

- Given n as the number of nodes in the graph, pick a set $T \subset A$ of n-1 arcs forming a spanning tree. The remaining arcs can be divided into the two sets L and U.
- Set $f_{ij} = m_{ij}$ for each arc (i, j) $\in$ L and $f_{ij} = M_{ij}$ for each arc (i, j) $\in$ U.
- Use the flow conservation constraints to determine the flows f_{ij} for arcs (i, j) $\in$ T. The algorithm begins by determining the flows on arcs incident to leaves of the tree T. Subsequently it determines the flows on other arcs of T. A spanning tree solution with $m_{ij} \leq f_{ij} \leq M_{ij}$ is a feasible spanning tree solution.

Theorem 1 [13]: Every basic feasible solution corresponds to a feasible spanning tree. Flow on non-basic arcs must be either m_{ij} or M_{ij}..

Theorem 2 [13]: A spanning tree structure (T, L, U) is optimal for the minimum cost flow problem if it is feasible and for some choice of node potential π the arc reduced costs satisfy the following conditions:

$$\bar{C}_{ij} = C_{ij} - \pi(i) + \pi(j) = 0 \quad for \ \ all \ (i, j) \in T$$
$$\bar{C}_{ij} = C_{ij} - \pi(i) + \pi(j) \geq 0 \quad for \ \ all \ (i, j) \in L$$
$$\bar{C}_{ij} = C_{ij} - \pi(i) + \pi(j) \leq 0 \quad for \ \ all \ (i, j) \in T$$

3 The Algorithm NSA

In network simplex algorithm, the linear algebra of the original simplex algorithm is replaced by simple network operations. With simple network operations, the minimum cost flow problem can be solved more than 100 times faster than equivalently sized Linear Programs. Ahuja, Magnanti, and Orlin [13] described the network simplex algorithm and gave pseudo-codes and hints. Helgason and Kennington [17] provided detailed information on an implementation. Here, a short description of the algorithm is presented.

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 2 specifies steps of the method [13,19] .

To create the initial or Basic Feasible Solution (BFS), an artificial node 0 is appended to the graph. It provides one way of obtaining an initial spanning tree structure for a connected graph. It can be assumed that for every node j Є N-{0}, the network graph contains arc (0,j) and (j,0), with sufficiently large costs and capacities. The initial tree T is constructed as follows: each node j, other than node 0, is examined one at a time. If b(j) ≥ 0, the algorithm includes (0,j) in T with a flow value of b(j). If b(j) ≤ 0, it includes arc (j,0) in T with a flow value of –b(j). The set L consists of the remaining arcs, and the set U is empty [13].

Algorithm ***Network Simplex Method***
Begin
Generate Initial BFS; (T, L, U)
(k, l) ← ***Entering Arc Є { L + U }***
While (k, l) ≠ *NULL* **Do**
Find Cycle W Є { T + **(k, l)** }
θ ← ***Flow Change***
(*p, q*) ← ***Leaving Arc Є W***
Update Flow in W by **θ**
Update BFS; Tree T
Update node potentials
(k, *l*) ← ***Entering Arc Є { L+ U}***
End while
End Algorithm

Figure 2: The Network Simplex algorithm (NSA)

Appending the entering arc, (k, l), to the spanning tree forms a unique cycle, *W*, with the arcs of the basis. In order to eliminate this cycle, one of its arcs must leave the basis. The cycle is eliminated when we have augmented flow by a sufficient amount to force the flow in one or more arcs of the cycle to their upper or lower bounds. By augmenting flow in a negative cost augmenting cycle, the objective value of the solution is improved. The first task in determining the leaving arc is the identification of all arcs of the cycle. The flow change is determined by the following equation:

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

The leaving arc is selected based on cycle W. The substitution of entering for the leaving arc and the reconstruction of new tree is called a pivot. After pivoting to change the basis**,** the reduced costs $\bar{c}_{ij} = c_{ij} - (\pi_i - \pi_j)$ for each arc $(i, j) \notin T$ is calculated. Note that $\bar{c}_{ij} = 0$ for all arcs $(i, j) \in T$ by construction. If $\bar{c}_{ij} \geq 0$ for all $(i, j) \in L$ and $\bar{c}_{ij} \leq 0$ for all $(i, j) \in U$ then the current basic feasible solution is optimal. Otherwise, an arc (i, j) where there is a violation should be chosen and operations of the algorithm should be repeated.

Different strategies are available for finding an entering arc for the basic solution. These strategies are called pricing rules. The performance of the algorithm is affected by these strategies. A literature review over these strategies is given below:

The standard textbook [13] provided a detailed account of the literature on those strategies. We now briefly review this literature. Goldfarb and Reid (1977) proposed a steepest edge pricing criterion. Instead of selecting an entering arc on the basis of the reduced cost, it is selected by taking into account the changes in all arcs. Another candidate list strategy has been described by Mulvey (1978). 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 strategy 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. Andrew [18] studied practical implementation of minimum cost flow algorithms and claimed that his/her implementations worked very well over a wide range of problems.

In recent years, several researches have been devoted on network simplex algorithm. Muramatsu [15] 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. Eppstein [22] presented a clustering technique for partitioning trees and forests into smaller sub-trees or clusters. 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 [12] developed some pricing rules to select an entering arc, a mixture of arc block and hot-list. In that pricing rule, a basket of violated arcs are collected in each iteration. A general pricing scheme for the simplex method has been proposed by Istvan [14]. He/she claimed that it creates a large flexibility in pricing and applicable to general and network simplex algorithms. Ahuja et al [16] developed a network simplex algorithm with O(n) consecutive degenerate pivot. He presented an anti-stalling pivot rule, based on concept of *strong feasible* spanning tree. The basis structure (T, L, U) is *strongly feasible* if we can send a positive amount of flow from any node to 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.

2.4 The 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. Secondly, a mixture of heuristic approach and memory technique are used in NSA+. The first one is not a new, but a combination of the two features provides a higher speed. These features are briefly explained below.

The first feature is related to maintaining the strongly feasible basis at each iteration. At the beginning, NSA+ chooses a strongly feasible solution. In each pivot, the leaving arc is selected appropriately so that the next basis is also strongly feasible.

The second feature is concerned with the entering arc. The arcs in 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 block's size depends on how many arcs are in the graph, while the number of most violated arcs may be a percentage of block's size. In our software, DSAGV, 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 biggest cost of arcs, for example). Scanning of the arcs for violation among different blocks is chosen circularly. At the beginning of the entering arc procedure, reduced costs of the most violated arcs in previous stage are recalculated. If they violate the optimality conditions again, they are kept in the packet. Otherwise they are replaced

by new violated arcs. Then, some new violated arcs, based on scanning of arcs from the blocks, are put into the packet as long as it has empty place. At the end of procedure, if the packet is empty, the current solution is optimal. Otherwise the packet will be sorted decreasingly, based on absolute value of the reduced costs, and the most violated arc will be chosen as the entering arc.

3. Scheduling problem of Automated Guided Vehicles

In this section, scheduling problem of Automated Guided Vehicles (AGVs) in the container terminals is introduced. The problem is to deploy several AGVs in a port to carry many containers from the quay-side to yard-side or vice versa. The main reason to choosing this problem is that the efficiency of a container terminal is directly related to use the AGVs with full efficiency [see e.g. 1, 2, 4, 8, 10]. This problem is formulated as a Minimum Cost Flow (MCF) model and then solved by primal version of NSA+.

3.1 Assumptions.

Following are the assumptions made in this paper.

Assumption 1: The layout of a port container terminal is given [10]. According to a specific layout the travel time between every combination of Pickup (P) /Drop-off (D) points is provided.

Assumption 2: It is assumed the vehicles move with an average speed so that there are no Collisions, Congestion, Live-locks, Deadlocks [8] and breakdown problem while they are carrying the containers.

Assumption 3: Every AGV can transport only one container. Also it is assumed that the start location of each AGV at the beginning of process is given.

Assumption 4: The yard is divided to several blocks and RTGCs or yard crane resources are always available [4], i.e., the AGVs will not suffer delays in the storage yard location or waiting for the yard cranes.

Assumption 5: Appointment time of every job at its source (destination) on the quay side is given.

Assumption 6: The quay side consists of several Quay Cranes (QCs). For each QC, there is a predetermined job sequence, consisting of loading or unloading jobs, or a combination of both.

For each loading (unloading) job, there is a predetermined pickup (drop-off) point in the yard, which is the origin (destination) of the job.

Assumption 7: There are N containers jobs in the container terminal. The source and destination of them in the port are given.

Assumption 8: 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.

3.2 Minimum Cost Flow Model of the Problem

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

- t_i : Appointment time of job i at the quay side (to be unloaded or dropped-off).
- RTA_m: Ready time of AGV m at start location, where may be either the quay-side or yard.
- T_{ij}: Travelling time from location of job i (quay-side) to location of job j (quay-side).
- P: Penalty of delay if an AGV can not reach on the quay side at the appointment time.
- TTA_{mi}: The travel time of the AGV m from the start location to the location of job i on the quay side.

The calculation of travel time between the jobs i and j, T_{ij} , is illustrated by figure 3 in different cases [4]. The RTA_m should be calculated in the similar manner.

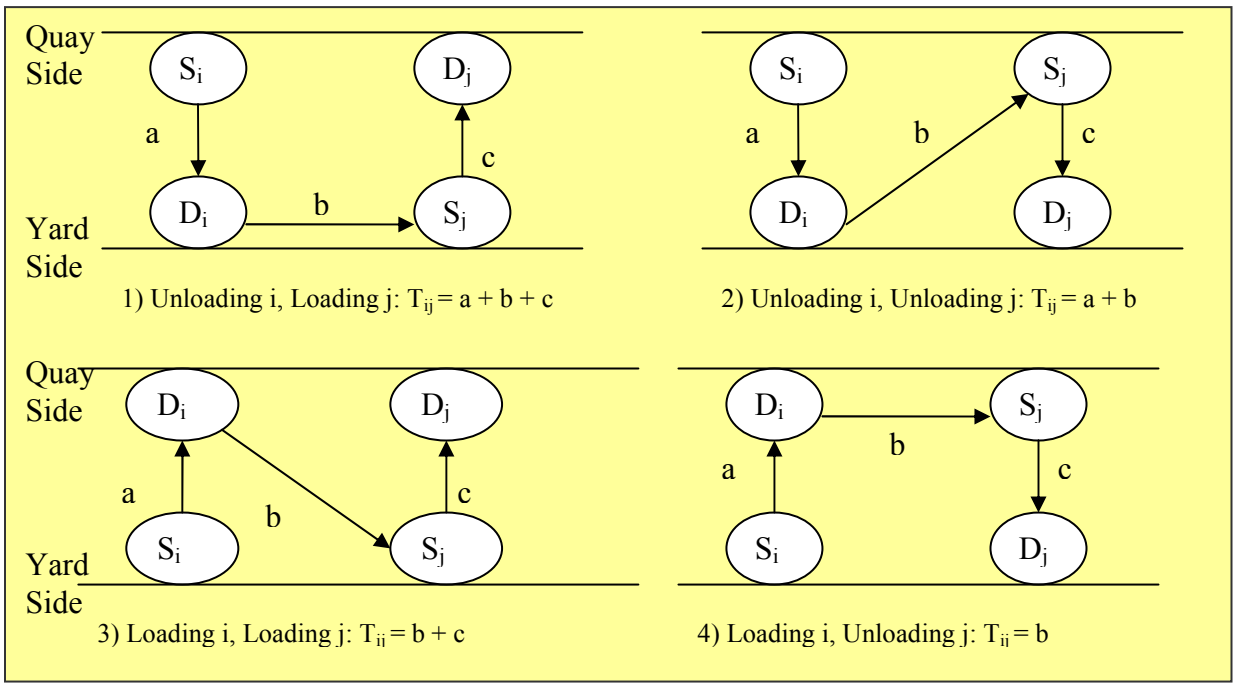

Figure 3: Travelling time computations between two jobs.

The optimal assignment of the AGVs to container jobs can be determined by solving a minimum-cost network flow problem. Cheng et al. (2003) minimized the impact of delays and waiting times of the AGVs at the quay side [4]. In this paper, our objectives are to minimize (a) the total AGV waiting time at the quay side (b) the total AGV travelling time in the route of port (c) the total lateness times to serve the jobs.

A directed graph to represent the complete network flow is suggested. Given number of jobs, N, and number of vehicles, M, in the problem, there are N+M+1 node in the graph. Every container job (unloading/loading) is represented as a node in the graph. There is a node corresponding to each vehicle, capturing the state of the vehicle at the time of deployment. We also need to insert a sink node, corresponding to the end state of the vehicles, after all container jobs have been served.

To define the arcs in the graph model, the following notations are introduced:

- The job j is consistent with job i if a single vehicle can be used to serve job j after serving job i. Therefore, these jobs are consistent if:

$$t_i + T_{ij} \leq t_j \ .$$

- Similarly the vehicle m is consistent with job i if:

$$RTA_m + TTA_{mi} \leq t_i.$$

The following three types of arcs connect the various nodes in the graph:
1) There is a directed arc from a vehicle, node m, to a container job, node i. If the vehicle m is consistent with job i, cost of the arc (m, i) corresponds to the waiting and travelling time that the vehicle m incurs at the quay crane location of job i. However, in practical situation, it may not be possible for the vehicle to arrive at the quay side at the appointment time. A large penalty should be considered for this delay to avoid those inconsistent arcs.

Hence, the Waiting and Travelling (Distance) times between vehicle m and job i as well as the Lateness times to serve job i are:

$$\begin{cases} \text{if job } i \text{ can be carried by AGV } m \ (t_i \geq RTA_m + TTA_{mi}) \rightarrow \begin{cases} WT_{mi} = t_i - (RTA_m + TTA_{mi}) \\ DT_{mi} = RTA_m + TTA_{mi} \\ LT_{mi} = 0 \end{cases} \\ \text{if job } i \text{ can not be carried by AGV } m \ (t_i < RTA_m + TTA_{mi}) \rightarrow \begin{cases} LT_{mi} = (RTA_m + TTA_{mi} - t_i) \\ WT_{mi} = DT_{mi} = 0 \end{cases} \end{cases}$$

2) There is a directed arc from container job i to container job j. If the job j is consistent with job i, cost of the arc (i, j) is the waiting and travelling times that the vehicle incurs. However in the practical situation, it is impossible to serve the jobs within the specified time. Similar to the inconsistent arc between vehicle node and containers node, a large penalty should be considered to avoid serving these sequences. Thus, the Waiting and Travelling (Distance) times of vehicles and the Lateness times to serve job node j after serving job node i are:

$$\begin{cases} if\ \ job\ \ j\ \ can\ \ \ be\ \ served\ after\ job\ i\ \ (t_j \geq t_i + TT_{ij}) \rightarrow \begin{cases} WT_{ij} = t_j - (t_i + TT_{ij}) \\ DT_{ij} = TT_{ij} \\ LT_{ij} = 0 \end{cases} \\ if\ \ job\ \ j\ \ can not\ \ be served\ after\ \ job\ i (t_j < t_i + TT_{ij}) \rightarrow \begin{cases} LT_{ij} = (t_i + TT_{ij} - t_j) \\ WT_{ij} = DT_{ij} = 0 \end{cases} \end{cases}$$

3) There exists a directed arc from each of the vehicle nodes and the container nodes to the sink node, which denoted by s. These arcs show that a vehicle can remain idle after serving any number of jobs or without serving any job. Therefore, a cost of zero is assigned to these arcs:

$$\begin{cases} WT_{ms} = DT_{ms} = LT_{ms} = 0; Cost\ \ \ between\ \ \ vehicle\ m\ \ and\ \ \ Sink\ \ is\ \ \ zero. \\ WT_{is} = DT_{is} = LT_{is} = 0; Cost\ \ \ between\ \ \ job\ \ \ i\ \ and\ Sink\ \ is\ zero. \end{cases}$$

3.3 The Mathematical Model

The MCF model in the previous section can be written in mathematical form as follows:

$$Min \left[w1(\sum_i \sum_j WT_{ij} f_{ij}) + w2(\sum_i \sum_j DT_{ij} f_{ij}) + P \sum_i \sum_j LT_{ij} f_{ij} \right]$$

$$S.t \begin{cases} \sum_j f_{ij} - \sum_j f_{ji} = 0; \quad \forall\, i \in Container\ \ \ Nodes \\ \sum_j f_{ij} = 1; \quad \forall\, i \in AGV\ \ \ Nodes \\ \sum_j f_{ji} = M; \quad for\ i = Sink\ \ Node \\ f_{ij} = \begin{cases} 1 & if\ \ AGV\ \ goes\ \ from\ node_{\ i}\ to\ node_{\ j} \\ 0 & otherwise \end{cases} \end{cases}$$

In the model f_{ij}, WT_{ij} , DT_{ij} and LT_{ij} are the flow traversing, waiting, travelling and lateness costs across arc a_{ij}, respectively. In the objective function, w1 and w2 are weights for the waiting and travelling times of the vehicles. A large value, P, is considered to avoid the

inconsistent arcs. The first three constraints show the flow balance for the network, and the fourth constraint decrees that all the container nodes are visited exactly once. Solving this network model generates M paths, each of which commences from a vehicle node and terminates at the sink node. Each path determines a job sequence of every vehicle.

The MCF model of the problem can be illustrated by figure 4 for two AGVs and four container jobs. In order to prevent an AGV to go directly to the sink node, every container node should be replaced by a couple of nodes. The arc between this couple (nodes 3 and 4 for job_1, for example) has unit lower and upper bound. Moreover, the transition cost between these couples is zero. These conditions guarantee that every container node should be visited once only.

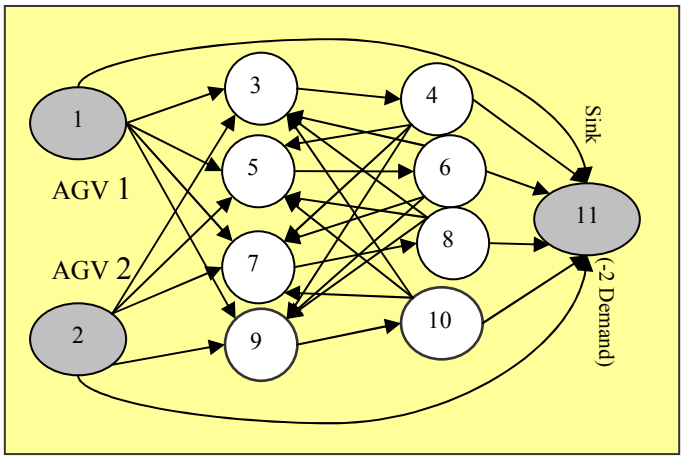

Figure 4: An example of network flow model for AGV scheduling.

The problem has a huge search space and the solution should provide the optimal paths for each AGV from every vehicle node to the sink node. Given number of jobs and number of vehicles, N and M respectively, there are M+2*N+1 nodes and M+M*N + N*(N-1) + 2*N in the graph model. Suppose that for some values of arc costs, the paths given by NSA+ are 1-3-4-9-10-11 and 2-5-6-7-8-11. This states that AGV 1 is assigned to serve jobs 1 and 4, and AGV 2 is assigned to serve jobs 2 and 3, respectively.

3.4 Experimental results for Static Problems

To test the model and NSA+, a hypothetical port was designed. In each run, random jobs were generated. In each generated job, the sources and destinations were chosen randomly by our software (DSAGV). The following parameters were used to define the port, the objective function, the number of vehicles and generate the jobs (table 1).

Table 1: Value of Parameters for the simulation

Description of the Parameters	Values
Number of Vehicles 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	120 second
Weight of waiting Times for the Vehicles in the objective function	1
Weight of travelling Times for the Vehicles in the Objective Function	5
P as the penalty (See mathematical formulation)	10000

We implemented DSAGV in Borland C++. Then, the software has been run to solve several random problems. The time to solve the problems has been drawn, by figure 5, according to both number of jobs and number of arcs in the graph model. Also the power estimation for those two curves has been shown on the figures. All experiments were run on a Pentium-4 2.4 GHz PC with 1 GMB RAM. As it can be seen in the figures, NSA+ can find the global optimal solution for 3000 jobs and 10 millions arcs in the graph model within 2 minutes.

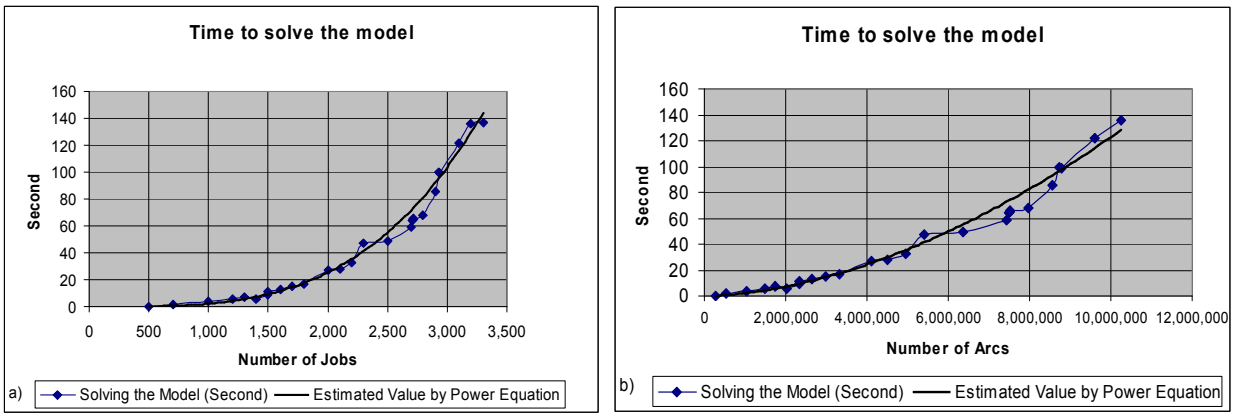

Figure 5:.Time to solve the graph model (a) based on number of Jobs, b) based on number of Arcs

4. Dynamic Scheduling of AGVs

Traditionally, scheduling problem of vehicles in the container terminal is static nature where everything is known beforehand. In reality, the problem is dynamic. New jobs arrive from time to time. Delays or breakdown of vehicles change the travelling time and availability of vehicles. NSA+ is a complete algorithm. Although it is efficient, it can only work on problems with certain limits in size. To complement NSA+, a Greedy Heuristic Search (GHS) is designed and implemented. GHS will be useful for problems which sizes go beyond the limit of NSA+, or in dynamic scheduling where reactive responses are called for.

4.1 The Greedy Heuristic Search Algorithm

In this simple search method, every time a job needs to be served. This behaves as a Taxi Service System. For any unassigned job and the list of idles AGVs, a job is assigned to a vehicle with minimum costs, including waiting time and travelling time. A block diagram describing GHS is presented in figure 6.

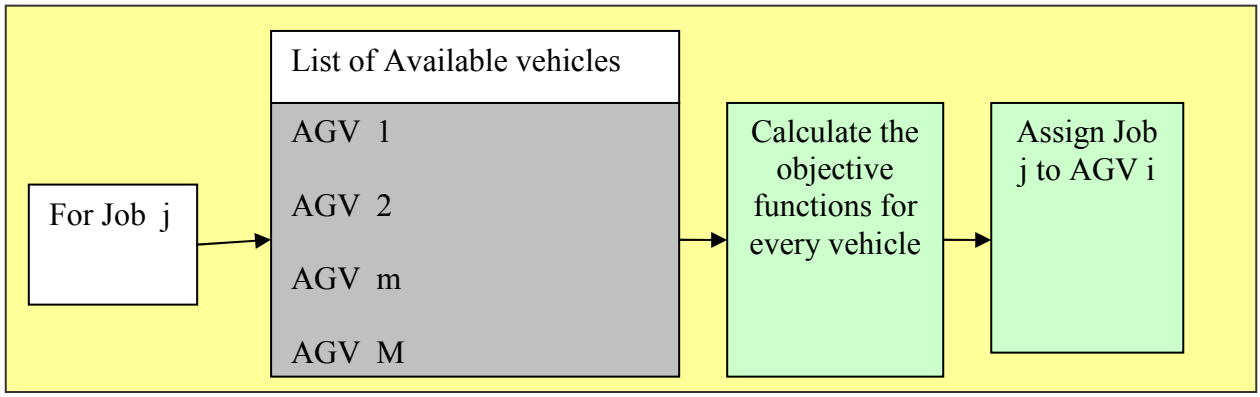

Figure 6: Block Diagram of Greedy Heuristic Search.

Given the appointment time of container job j, t_i, the ready time of AGV *m* to get the start location, RTA_m, and the travel time of the AGV from its start location to the source/destination of job j on the quay side, TTA_{mj}, the objective function is calculated as:

$$ObjectiveFunction(m, j) = w1(t_j - RTA_m - TTA_{mj}) + w2(RTA_m + TTA_{mj})$$

Note that w1 and w2 are the weight of waiting time and travelling time of the vehicles, as section 3.

We can compare this method with the graph model in section 3. This is a special case of the minimum cost flow problem with M vehicle node, N container nodes and one sink node, but every time an idle vehicle is assigned to just one of the remaining jobs. The graph model of the problem can be illustrated by figure 7 for two AGVs and four container jobs.

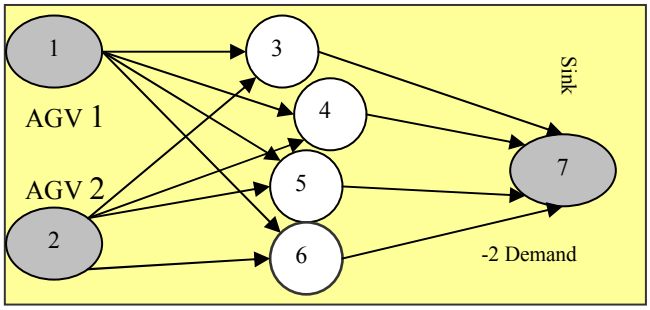

Figure 7: An example of network flow model for AGV scheduling.

The constraints for this simple cost flow problem are as follows:

$$S.t\begin{cases}\sum_j f_{ij}=1; \quad \forall i \in AGV\ \ Nodes \\ \sum_j f_{ji}=M; \quad for\, i = Sink\ Node \\ f_{ij}=\begin{cases}1 \ \ if\ AGV\ goes\ from\, node_i\, to\, node_j \\ 0\ otherwise\end{cases} \\ f_{is}=\begin{cases}1 \ \ if\ \sum_m f_{mi}=1 \\ 0\ \ Otherwise\end{cases}\ \forall i \in Container\ \ Nodes; s = Sink\ Node\end{cases}$$

The pseudo code for GHS is shown in figure 8.

```
If there is any remaining job and there is any idle vehicle
      Calculate the objective function (v, j): the travelling times and waiting times.
      For v=1, 2, #Idle vehicles; for j=1, 2,.., #Remaining jobs
Else
   Stop
End If
Again:
Select a vehicle m and a job j with the minimum cost.
Assign the vehicle m to the job j.
Remove the vehicle m from the list of idle vehicles and the job j from the remaining job.
If there is any idle vehicle and any left jobs
   Go to again
End If
Stop
```

Figure 8: Pseudo code of Simple Heuristic Search

4.2 Software Architecture for dynamic aspect

The architecture of the main part of the software that implements GHS is shown in figure 9. In this section, this architecture and operation are explained briefly. A similar architecture and operation for NSA+ are considered in our software.

At the start of 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 the Greedy Heuristic Search and the output of this method is an individual job for every vehicle.

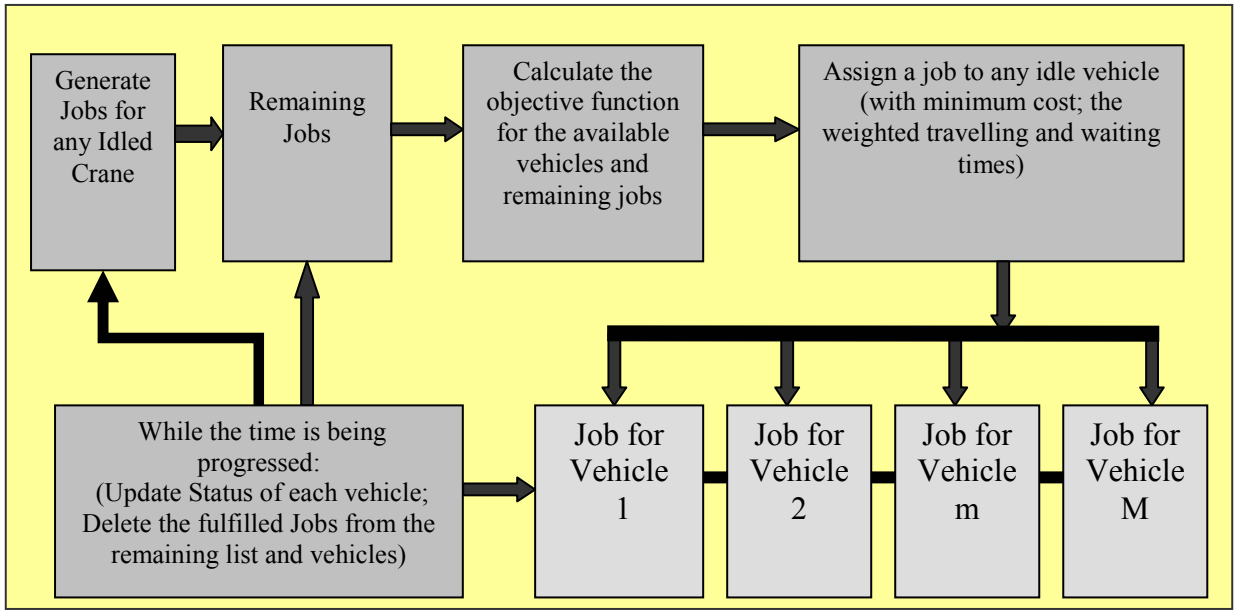

Figure 9: Block Diagram of the Greedy Heuristic Search in dynamic aspect

The software does two tasks in 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. In real time processing, the travelled and waited times of every vehicle should be updated. At the same time, if the vehicle picks up the job from the quay side, the assigned job for the vehicle will be deleted and removed from the list of remaining jobs. If the job should be delivered to the crane on the quay side, it could not be removed until meeting time between the crane and the vehicle (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.

4.3 A comparison between GHS and NSA+

To evaluate the relative strength and weakness of GHS 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 1. Figures 10 to 12 demonstrate some outputs of the software during the six hours.

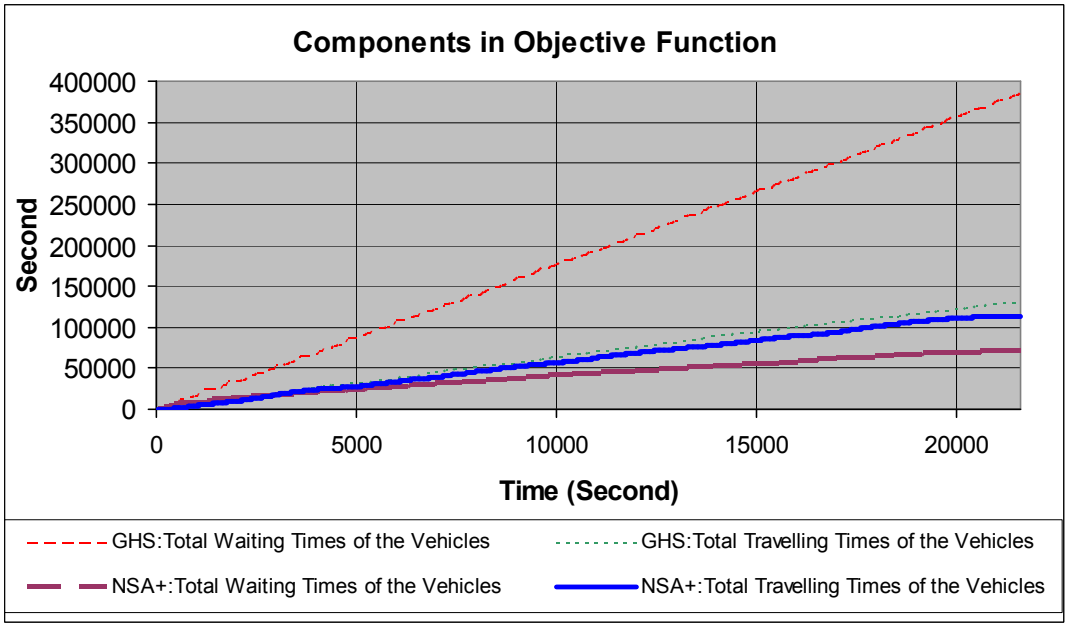

Figure 10: A comparison of Travelling and Waiting Times of the Vehicles

As we can see from figure 10, waiting times of the vehicles for Greedy Heuristic Search is significantly greater than waiting times of the vehicle in NSA+, although travelling times of the vehicles for both algorithms are almost the same during the 6 hours. The main reason for this result is that NSA+ finds out a complete solution for the Minimum Cost Flow problem whereas GHS is incomplete search method.

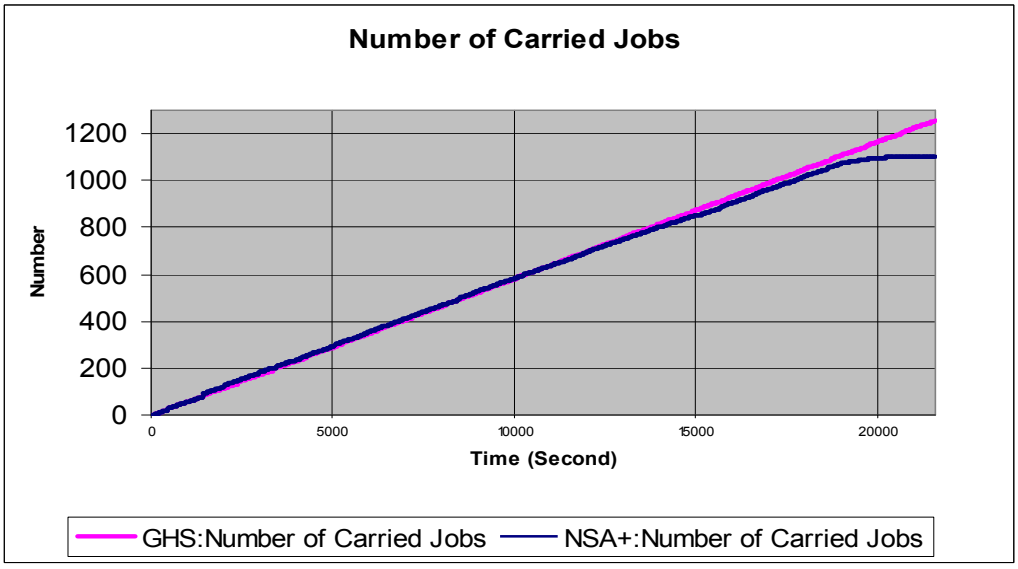

Figure 11: The number of carried jobs by two algorithms during 6 hour simulation

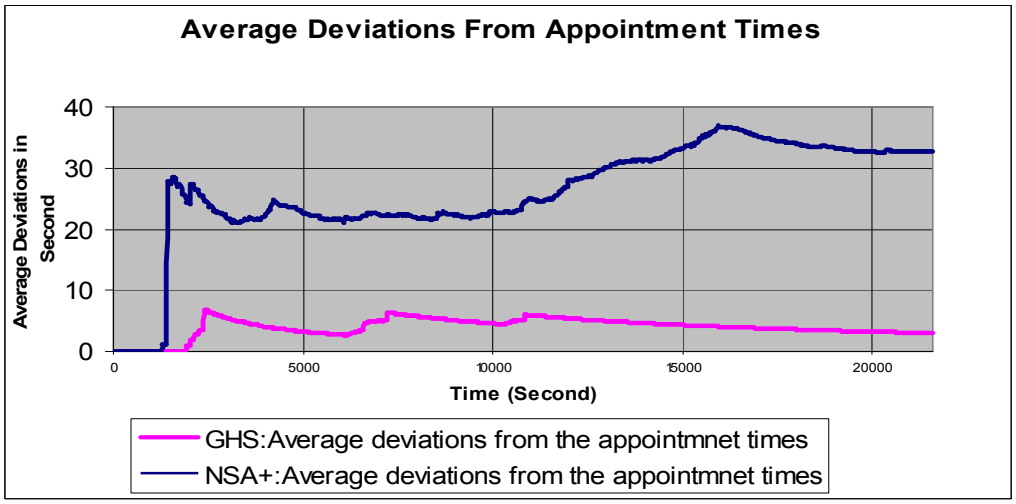

Figure 12: Average deviations from the appointment time

The number of jobs carried out by the end of six-hour (21,600 seconds) for both algorithms is approximately the same (figure 11). Generally, due to the tight schedules of the quay cranes, it is undesirable for containers to be served too early or too late for the appointment. Therefore, the average deviation from the appointment times is another indicator for evaluating the algorithms. Given the number of served jobs, N, the time at which the job i is served, ACT_i, and the time of job i's appointment, APT_i, a schedule's average deviations is calculated by the following equation:

$$Average\ Deviation = \frac{\sum_{i=1}^{N} \left| ACT_i - APT_i \right|}{N}$$

Figure 12 presents the Average Deviation indicator for both NSA+ and GHS during the six-hour simulation. The figure shows that both algorithms performed well, but GHS is superior to NSA+ in average deviation. This is hardly surprising as NSA+ was not designed to minimize the average deviation measure.

4.4 Statistical test for the comparison

The waiting and travelling times of the vehicles, produced by NSA+ and GHS, were analysed statistically. We tested the null hypothesis that the means produced by the two algorithms were statistically indifferent. Table 2 shows the test results along with the critical values of t-distribution. The one-tail t-test suggests that NSA+ is significantly better than GHS in both travelling and waiting times of the vehicles with 95% level of confidence. On the other hand, GHS is statistically better than NSA+ in average deviation.

Table 2: The result of T-Test for the difference between NSA+ and GHS

Statistical Parameters	Total Waiting Times of the Vehicles	Total Travelling Times of the Vehicles	Average deviation from the appointment times
Observations	721	721	721
t-test (Paired Two Sample For Means)	-43.40547441	-43.5902651	73.68094065
Degree of Freedom	720	720	720
Critical t-value	1.646972	1.646972	1.646972

5. Concluding summary

In this paper, the automated guided vehicles scheduling problem in the container terminals was formulated as a minimum cost flow and then solved by a new version of Network Simplex Algorithm (NSA+). Our software, implemented in Borland C++, running on a 2.4 GHz Pentium PC, could find the global optimal solution for 3000 jobs and 10 millions arcs in the graph model within 2 minutes. To complement NSA+, the Greedy Heuristic Search (GHS) was designed. The two algorithms were compared on both static and dynamic problems. Given the results so far, we claim that NSA+ is efficient and effective in both waiting and travelling times of the vehicles. GHS is useful when the problem is too big for NSA+ to solve. Being an incomplete algorithm, GHS sacrifices completeness. This establishes NSA+ plus GHS as complementary algorithms for practical dynamic automatic vehicle scheduling.

References:

1. Böse J., Reiners T., Steenken D., Voß S., (2000), "Vehicle Dispatching at Seaport Container Terminals Using Evolutionary Algorithms". Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, IEEE, Piscataway, DTM-IT: 1-10.
2. Huang Y., Hsu W.J., (2002), "Two Equivalent Integer Programming Models for Dispatching Vehicles at a Container Terminal". CAIS, Technical Report, School of Computer Engineering, Nan yang Technological University, Singapore 639798.
3. Murty K.G., Jiyin L., Yat-Wah W, Zhang C, Maria C.L. Tsang, Richard J. Linn. (2002), "DSS (Decision Support System) for operations in a container terminal". Decision Support System.
4. Cheng Y., Sen H., Natarajan K., Teo C., Tan K., (2003),"Dispatching automated guided vehicles in a container terminal", Technical Report, National University of Singapore.
5. Patrick J.M., Wagelmans P.M., (2001), "Dynamic scheduling of handling equipment at automated container terminals", Technical Report EI 2001-33, Erasmus University of Rotterdam, Econometric Institute.
6. Patrick J.M., Wagelmans P.M., (2001), "Effective algorithms for integrated scheduling of handling equipment at automated container terminals", Technical Report EI 2001-19, Erasmus University of Rotterdam, Econometric Institute.
7. Patrick J.M., Dekker R., (2003), "Operations research supports container handling", Technical Report EI 2001-22, Erasmus University of Rotterdam, Econometric Institute.
8. Qiu L., Hsu W.-J., Huang S.-Y and Wang H. (2002), "Scheduling and Routing Algorithms for AGVs: a Survey". International Journal of Production Research, Taylor & Francis Ltd, vol. 40, no. 3, pp (745-760).
9. Sen H., (2001), "Dynamic AGV-Container Job Deployment". Technical Report, HPCES Programme, Singapore-MIT Alliance.
10. Wook B.J., Hwan K.K., (2000),"A pooled dispatching strategy for automated guided vehicles in port container terminals", International Journal of management science, Vol 6, No 2.
11. Tsang E.P.K., (1995), "Scheduling techniques -- a comparative study", British Telecom Technology Journal, Vol.13, No.1., Martlesham Heath, Ipswich, UK.
12. Löbel A., (2000), "A Network Simplex Implementation", Technical Report, Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB).
13. Ahuja R.K., Magnanti T.L., Orlin J.B., (1993), "Network Flows: Theory, Algorithms and Applications". Prentice Hall.
14. Istvan M, (2001-3), "A General Pricing Scheme for the Simplex Method", Technical Report, Department of Computing, Imperial College, London.
15. Masakazu M., (1999), "On network simplex method using primal-dual symmetric pivoting rule", Journal of Operations Research of Japan.

16. Ahuja R.K., Orlin J. B., Sharma P., Sokkalingam P.T., (2002), “A network simplex algorithm with O(n) consecutive degenerate pivots”. Operations Research Letters 30.
17. Helgason R., Kennington J., (1995), "Primal Simplex Algorithms for Minimum Cost Network Flows," Handbook on Operations Research and Management Science Volume 7, Editors M. Ball, T. Magnanti, C. Monma, and G. Nemhauser, North-Holland, Amsterdam, 85-133.
18. Goldberg A.V., Kennedy, R**.** (1993), **“**An Efficient Cost Scaling Algorithm for the Assignment Problem**”.** Technical Report, Stanford University**.**
19. Kelly D.J., ONeill G.M., (1993), **"**The Minimum Cost Flow Problem and The Network Simplex Solution Method**",** Master Degree Dissertation, University College, Dublin.
20. Ahuja R.K., Orlin J.B., Giovanni M.S., Zuddas P, (1999), “Algorithms for the simple equal flow problem”, Management Science, 45(10):1440--1455.
21. Andrew V.G., (1997) “An efficient implementation of a scaling minimum-cost flow algorithm”. Journal of Algorithms, 22(1):1-29.
22. Eppstein D, (1999) “Clustering for faster network simplex pivots”, In Proc. 5th ACM-SIAM Symposium. Discrete Algorithms, pp 160–166.