

**Submission to Annals of Operations Research
(post-MISTA 2005 proceedings)**

**Applying the Extended Network Simplex Algorithm and a Greedy
Search Method to Automated Guided Vehicle Scheduling**

Hassan Rashidi
Department of Computer Science
University of Essex, Colchester CO4 3SQ,
U.K.
Email: hrashi@essex.ac.uk
Tel: +44 1206 872315

Edward P. K. Tsang
Department of Computer Science
University of Essex, Colchester CO4 3SQ,
U.K.
Email: edward@essex.ac.uk
Tel: +44 1206 872774

Abstract: In this paper, a scheduling problem for Automated Guided Vehicles in container terminals is defined and formulated as a Minimum Cost Flow model. 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. To complement NSA+, an incomplete algorithm Greedy Vehicle Search (GVS) is designed and implemented. The NSA+ and GVS are compared and contrasted to evaluate their relative strength and weakness. With polynomial time complexity, NSA+ can be used to solve very large problems, as verified in our experiments. Should the problem be too large for NSA+, or the time available for computation is too short (as it would be in dynamic scheduling), GVS complements NSA+.

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

Introduction

The Minimum Cost Flow (MCF) problem is the problem of flowing 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). This problem arises in a large number of industries, including agriculture, communications, defence, education, energy, health care, manufacturing, medicine, retailing, and transportation [1]. This paper has been motivated by a need to schedule Automated Guided Vehicles (AGVs) in 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. 5, 6, 10, 16, 17, 22]. A transportation requirement in a port is described by a set of jobs, each of which being characterized by the source location of a container, the destination location and its pick up or drop-off times 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 structure of this paper is as follows. Section 1 establishes a new definition for Minimum Cost Flow (MCF) model. Then the scheduling problem of Automated Guided Vehicles (AGV) is defined and formulated as a special case of MCF model. We present our problem with MCF-AGV model in Section 2. Section 3 presents two algorithms to tackle the MCF-AGV problem. Experimental results from applying the two algorithms to tackle the problem are compared in Section 4. Section 5 is considered to summary and conclusion.

1 Minimum Cost Flow (MCF) model

In this section, we systematically introduce a formal definition for the MCF model:

Definition 1 [1]: In an informal description of the MCF model, 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 is greater (less) than zero, node i is a supply (demand) node; 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 model is as follows:

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

These constraints state that flows must be feasible and conserve each node. 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. We

now define a special graph for the MCF model as follows:

Definition 2: A MCF 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 NP: $N \rightarrow \mathbb{R}$ (Real numbers; possibly negative) gives the amount of supply/demand of the nodes. This function for each node is defined as follows:

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

Each arc in A has three properties: a lower bound, an upper bound and a cost. The arc property function AP maps each arc to these properties, AP: $A \rightarrow \mathbb{R} \times \mathbb{R} \times \mathbb{R}$ (Real numbers; nonnegative). For each arc $\in A$, we denote the mapping by AP(i,j), or AP $_{ij}$ in short. We denote the lower bound, upper bound and cost by m_{ij} , M_{ij} and c_{ij} . Based on Definitions 1 and 2, we define the standard Minimum Cost Flow (MCF) problem, formally as follows:

Definition 3: a MCF model is defined as $\text{MCF} = (G_{\text{MCF}}, f, D, \text{CS}, \text{FC})$ where $G_{\text{MCF}} = ((N,A), \text{NP}, \text{AP})$ is a graph with nodes and arcs specific to the MCF model (Definition 2); f is a finite set of decision variables on A (f stands for flow), $f = \{ f_{ij} \mid (i, j) \in A \}$; D is 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} (D stands for Domain); CS is a finite set of Constraint on NP and f ; FC is 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 of FC. In the standard form of the MCF model we have:

(a) For each element in 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 $\sum_{j:(j,i) \in A} c_{ij} \cdot f_{ij}$

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

In this section, a 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. 5, 6, 10, 19, 22]. This problem is formulated as a special case of MCF model.

2.1 Assumptions

Assumption 1: The layout of a port container terminal is given [22]. 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 [19] 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 [6], i.e., the AGVs will not suffer delays in the storage yard location or waiting for the yard cranes.

Assumption 5: 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 6: There are N jobs and M AGVs in the problem. The source and destination of jobs as well as their appointment time on the quay side are given. For the dynamic problem, it is assumed that M is fixed, but the number of jobs and the distance between every two points in the port may be changed.

Assumption 7: 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. Cheng et al. (2003) minimized the impact of delays and waiting times of the AGVs at the quay side [6].

2.2 Formulation of the problem

Here, we present a special case of the MCF model for the scheduling problem of automated guided vehicles in 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 values to the nodes. Additionally, the property function of arcs may assign integer values to the lower bound, the upper bound and the cost of each arc. Here, 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 AGVs (MCF-AGV).

Based on Definition 2, we introduce the following definition for the G_{MCF} in a special case:

A MCF Graph for AGV, $G_{MCF-AGV} = (GS, NPS, APS)$, is a special case of $G_{MCF} = (G, NP, AP)$ (Definition 2). The graph $GS = (NS, AS)$ will be defined in the subsections below; 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). We formally describe the components of $G_{MCF-AGV}$ in the two following sub-sections:

2.2.1 Nodes and their properties in the special graph

Let N be the number of jobs and M be the number of AGVs in the problem. The nodes of the MCF Graph for the AGV scheduling problem are defined as follows:

a) Supply Nodes: For each vehicle m , a supply node $AGVN_m$ with one unit supply is considered.

Therefore, the set of supply nodes in the graph are $SAGVN = \{AGVN_m \mid m=1,2,\dots,M; NPS(m)=1\}$

b) Transshipment Nodes: for each job j , a couple of nodes, Job-Input and Job-Output, are considered.

Hence, the sets of transshipment nodes in the graph are $SJIN \cup SJOUT$ where:

$SJIN = \{JIN_i \mid i=1,2,\dots,N; NPS(i)=0\}$ where JIN_i is a node through which an AGV enters job i .

$SJOUT = \{JOUT_i \mid i=1,2,\dots,N; NPS(i)=0\}$ where $JOUT_i$ is a node from which an AGV leaves job

i .

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

Therefore, there are $M+2*N+1$ nodes in the $G_{MCF-AGV}$: $NS = SAGVN \cup SJIN \cup SJOUT \cup SINK$.

2.2.2 Arcs and their properties in the special graph

Below we describe the four types of arcs that join the nodes in $G_{MCF-AGV}$, together with their properties:

1) **Intermediate Arcs:** These arcs are directed arcs from every Job-Output node i to other Job-Input node j .

These arcs with their properties are $ARC_{intermediate} = \{ (i, j) \mid i \in SJOUT, j \in SJIN, j \neq JIN_i, APS(m, j) = [0,1, C_{ij}] \}$

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

In the cost, w_1 and w_2 are the weight of waiting and travelling times of the AGVs, respectively; t_i and t_j is the appointment time of job i and j on the quay side (to be unloaded or dropped-off); DT_{ij} is Travelling time from location of job i to location of job j ; (calculation of the DT_{ij} is illustrated by figure 1 in different cases). If an AGV can serve job j after serving job i ($t_j \geq t_i + DT_{ij}$), the waiting and travelling times of the AGV are calculated without any lateness time. Otherwise, only the lateness time of serving job j with a penalty (P) is considered for the cost (see Assumption 7).

2) Inward Arcs: a set of arcs from SAGVN to SJIN. These arcs along with their properties are:

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

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

In the cost, w_1 and w_2 are the weight of waiting and travelling times of the AGVs, respectively; t_j is the appointment time of job j at the quay side (to be unloaded or dropped-off); RTA_m is Ready time of AGV m at start location, where may be either the quay-side or yard; TTA_{mj} is the travel time of AGV m from the start location to the location of job j on the quay side; (The TTA_{mi} should be calculated in the similar manner like the calculation of DT_{ij} ; see Intermediate arcs). If AGV m could arrive on the quay side in the appointment time of job j ($t_j \geq RTA_m + TTA_{mj}$), the waiting and travelling times of AGV m to serve job j are calculated as the cost. Otherwise, the lateness time to serving job j with a penalty (P) is considered.

3) Outward Arcs: These are directed arcs from every Job-Output node i and AGV node m to SINK.

These arcs along with their properties are $ARC_{outward} = \{ (i, j) \mid i \in SAGVN \cup SJOUT, j = SINK; APS(m, j) = [0, 1, 0] \}$. 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. These arcs along with their properties are $ARC_{auxiliary} = \{ (i, j) \mid i \in SJIN, j \in SJOUT; APS(i, j) = [1, 1, 0] \}$. These arcs guarantee that every Job-Input and Job-Output node is visited once only so that each job is served.

There are $M \cdot N + N \cdot (N-1) + M + 2 \cdot N$ arcs in the graph ($AS = ARC_{inward} \cup ARC_{intermediate} \cup ARC_{outward} \cup ARC_{auxiliary}$)

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

Now we present our model for the Automated Guided Vehicles Scheduling with the following definition:

Definition 4: A MCF-AGV model is a special case of the MCF (see Definition 3) for the Scheduling problem of Automated Guided Vehicles in the container terminals. A MCF-AGV model is defined as

MCF-AGV=($G_{\text{MCF-AGV}}$, f , D , CS , FC) where $G_{\text{MCF-AGV}} = (GS, \text{NPS}, \text{APS})$ is a graph for the MCF-AGV problem; f = a finite set of integer decision variables on AS , $f = \{f_{ij} \mid (i, j) \in AS\}$; 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 , we have
$$\begin{cases} 1) D_{f_{ij}} = [0,1]; \text{ for } \forall (i, j) \in (\text{ARC}_{\text{inward}} \cup \text{ARC}_{\text{intermediate}} \cup \text{ARC}_{\text{outward}}) \\ 2) D_{f_{ij}} = [1,1]; \text{ for } \forall (i, j) \in \text{ARC}_{\text{auxiliary}} \end{cases}$$

$$CS = \left\{ \begin{array}{l} 1) \sum_{j:(i,j) \in AS} f_{ij} = 1; \quad \forall i \in \text{SAGVN}; \text{ Sending one unit flow into the network from each node in SAGVN} \\ 2) \sum_{j:(j,i) \in AS} f_{ji} = M; \quad \text{for } i = \text{SINK}; \text{ Receiving } M \text{ units flow (the flows sent from the nodes in SAGVN set).} \\ 3) \sum_{j:(i,j) \in AS} f_{ij} - \sum_{j:(j,i) \in AS} f_{ji} = 0; \quad \forall i \in \text{SJIN} \cup \text{SJOUT}; \text{ Flow balance at every Job - Input and Job - Output node.} \end{array} \right\}$$

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

The MCF-AGV model can be illustrated by figure 2 for two AGVs and four jobs. 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. Solving the MCF-AGV 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. Suppose that for some values of arc costs, the paths given by a solution 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 jobs 1 and 4, and AGV 2 is assigned to serve jobs 2 and 3, respectively.

3 The Algorithms

In this section, a complete algorithm (NSA+) and an incomplete search method (GVS) to tackle the MCF-AGV model are presented. Since NSA+ is an extension of Network Simplex Algorithm (NSA), we describe NSA, first.

3.1 Network Simplex Algorithm (NSA)

Every connected network has a spanning tree [1]. The network simplex algorithm maintains a feasible spanning tree at each iteration and successfully goes toward the optimality conditions until it becomes optimal. At each iteration, the arcs in the graph are divided into three sets; the arcs belong to

the spanning tree (T); the arcs with flow at their lower bound (L); the arcs with flow at their upper bound (U). A spanning tree structure (T, L, U) is optimal if the reduced cost for every arc $(i,j) \in L$ is greater than zero and at the same time the reduced cost for every arc $(i,j) \in U$ is less than zero [1]. With those conditions, the current solution is optimal. Otherwise, there are arcs in the graph that violate the optimal conditions. An arc is a violated arc if it belongs to L (U) with negative (positive) reduced cost. The algorithm in figure 3 specifies steps of the method [1,12].

To create the initial or Basic Feasible Solution (BFS), an artificial node 0 and artificial arcs are appended to the graph. The node '0' will be the root of spanning tree (T) and the artificial arcs, with sufficiently large costs and capacities, connect the nodes to the root. The set L consists of the main arcs in the graph, and the set U is empty [1]. Appending the entering arc (k, l), which is a violated arc, 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 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 for each arc $(i, j) \notin T$ is calculated. If the reduced costs for all $(i, j) \in \{L + U\}$ satisfy the optimality condition 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. The standard textbook [1] provided a detailed account of the literature on those strategies. Goldfarb and Reid (1977) proposed a steepest edge pricing criterion. Mulvey (1978) suggests 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. Andrew (1997) studied practical implementation of minimum cost flow algorithms and claimed that his/her implementations worked very well over a wide range of problems [4]. Masakazu (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 [14]. Eppstein (1999) presented a clustering technique for partitioning trees and forests into smaller sub-trees or clusters [7]. 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 [13]. A general pricing scheme for the simplex method has been proposed by Istvan [11]. He/she claimed that it creates a large flexibility in pricing and applicable to general and network simplex algorithms. Ahuja et al. (2002) developed a network simplex algorithm with $O(n)$ consecutive degenerate pivot [2]. 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.

3.2 The Network Simplex plus Algorithm (NSA+)

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

The first feature is related to maintaining the strongly feasible basis at each iteration. At start, NSA+ chooses a strongly feasible solution. In order to do this, the initial basic solution, which was described in the previous section, is strongly feasible. In each pivot, the algorithm also selects the leaving arc in such a way that the next basis would be strongly feasible. By maintaining strongly

feasible bases due to Cunningham (1976, 1979), cycling can be prevented without restrictions on the entering arc [1].

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 violating arcs are collected. The capacity of the packet is more than the block's size and the most violating arcs are kept at the top of the packet. The number of most violating arcs may be a percentage of the block's size. We set the block's size and number of most violating 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, 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 each scan, one violating arc (at most) from each block is put into the packet. 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 could be replaced by new violating arcs. Then, some new violating arcs, through the 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 in descending order, based on the absolute value of the reduced costs, and the most violated arc will be chosen as the entering arc.

3.3 Greedy Vehicle Search Method

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 Vehicle Search (GVS) is designed and implemented. GVS will be useful for problems which sizes go beyond the limit of NSA+ or when the time available for computation is too short (as it would be in dynamic scheduling). This simple search method 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 and travelling times of the vehicles as well as lateness of the jobs. The pseudo code of GVS is demonstrated in figure 4.

4 Experimental Results from the implementation and running the algorithms

4.1 Static Problems

To test the model and make a comparison between NSA+ and GVS, a hypothetical port was designed. The parameters in table 1 were used to define the port, the objective function, the number of vehicles and generate the jobs. We implemented our software (DSAGV) in Borland C++. Then, DSAGV has been run to solve several random problems. The sources and destinations of jobs were chosen randomly. The time to solve the problems by the two algorithms has been drawn in figure 5, according to the number of jobs. 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 figure, NSA+ can find the global optimal solution for 3,000 jobs and 10 millions arcs in the MCF-AGV model within 2 minutes. GVS is fast and could find a local optimum for 8,000 jobs within 35 seconds. Given N jobs and M AGVs in the problem ($N \gg M$), the complexity of the two algorithms are calculated as follows:

- **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 $O((m + n) \cdot K \cdot \text{Log}K)$. Based on the complexity of the number of iterations and the complexity of each pivot, the total complexity of this algorithm is determined by the following equation:

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

Since $m=O(N^2)$; $n=O(N)$, the total complexity of NSA+ to tackle the MCF-AGV model is $O(N^6)$.

- **GVS:** 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 (among M AGVs). 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, the number of runs of GVS 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 significantly less than the complexity of NSA+.

4.2 Dynamic Problems

The majority part of the literature treat the vehicle scheduling problem in the container terminal as static problems, where all the jobs and travelling time are 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.

The main architecture of GVS is shown in figure 6. A similar architecture and operations for NSA+ are considered in our software. We shall use figure 6 to describe the architecture and operations briefly. 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 GVS and the output of this method is an individual job for every vehicle. When the time is running, 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 Job Generator has to generate a few new jobs, when it finds out any idle crane.

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 1. Figures 7 to 9 demonstrate some outputs of the software during the six hours. As we can see from figure 7, waiting times of the vehicles for GVS 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, being a complete algorithm, NSA+ finds the optimal solution for the Minimum Cost Flow problem whereas GVS is incomplete search method. The number of jobs carried out by the end of six-hour (21,600 seconds) for both algorithms is approximately the same (see figure 8). 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. Therefore, the average lateness 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 appointment time of job i , APT_i , a schedule's average lateness is calculated by the following equation:

$$\text{Average Lateness} = \frac{\sum_{i=1}^N (ACT_i - APT_i)}{N}$$

Figure 9 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 terms of the Average Lateness. As a greedy algorithm, GVS achieved this by sacrificing waiting and travelling times of the AGVs.

5 Concluding Summary

In this paper, the automated guided vehicles scheduling problem in container terminals was formulated as a special case of the minimum cost flow problem. Then, two novel algorithms, namely NSA+ and GVS, for tackling the problem were presented. NSA+ is a complete algorithm. Our

experimental results suggested that it could find the global optimal solution for 3,000 jobs and 10 millions arcs in the graph model within 2 minutes by running on a 2.4 GHz Pentium PC. GVS is an incomplete algorithm. It is useful when the problem is too big for NSA+ to solve, or when time available for computation is too short, as it could be the case in dynamic scheduling. The two algorithms were compared on both static and dynamic problems. Our experimental results suggested that NSA+ is efficient and effective in both minimizing the waiting and travelling times of the vehicles, whereas GVS is more effective in minimizing the average lateness. The two algorithms complement each other and can be used according to the situation and the users' needs. We claim that NSA+ and GVS together are practical algorithms for automatic vehicle scheduling.

REFERENCES

1. Ahuja R.K., Magnanti T.L., Orlin J.B., (1993), "Network Flows: Theory, Algorithms and Applications". Prentice Hall.
2. Ahuja R.K., Orlin J. B., Sharma P., Sokkalingam P.T., (2002), "A network simplex algorithm with $O(n)$ consecutive degenerate pivots". *Operations Research*, Vol 30(3), pp 141-148..
3. Ahuja R.K., Orlin J.B., Giovanni M.S., Zuddas P, (1999), "Algorithms for the simple equal flow problem", *Management Science*, Vol 45(10), pp 1440-1455.
4. Andrew V.G., (1997) "An efficient implementation of a scaling minimum-cost flow algorithm". *Journal of Algorithms*, Vol 22(1), pp 1-29.
5. 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*, pp 1-10.
6. 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.
7. Eppstein D, (1999) "Clustering for faster network simplex pivots", In *Proc. 5th ACM-SIAM Symposium. Discrete Algorithms*, pp 160–166.
8. Goldberg A.V., Kennedy, R. (1993), "An Efficient Cost Scaling Algorithm for the Assignment Problem". Technical Report, Stanford University.
9. Helgason R., Kennington J., (1995), "Primal Simplex Algorithms for Minimum Cost Network Flows," *Handbook on Operations Research and Management Science Volume 7*, North-Holland, Amsterdam, pp 85-133.
10. 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.
11. Istvan M, (2001-3), "A General Pricing Scheme for the Simplex Method ", Technical Report, Department of Computing, Imperial College, London.
12. Kelly D.J., ONeill G.M., (1993), "The Minimum Cost Flow Problem and The Network Simplex Solution Method", Master Degree Dissertation, University College, Dublin.
13. Löbel A., (2000), "A Network Simplex Implementation", Technical Report, Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB).
14. Masakazu M., (1999), "On network simplex method using primal-dual symmetric pivoting rule", *Journal of Operations Research of Japan*, Vol 43, pp 149-161.

15. 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, Vol 39, pp 309-332.
16. Patrick J.M., Dekker R., (2003), “Operations research supports container handling”, Technical Report EI 2001-22, Erasmus University of Rotterdam, Econometric Institute.
17. 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.
18. 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.
19. 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 (3), pp 745-760
20. Sen H., (2001), “Dynamic AGV-Container Job Deployment”. Technical Report, HPCES Programme, Singapore-MIT Alliance.
21. Tsang E.P.K., (1995), “Scheduling techniques -- a comparative study”, British Telecom Technology Journal, Vol.13 (1), Martlesham Heath, Ipswich, UK.
22. 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 (2), pp 47-60.

Table and Figures

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
Travelling Time between every two points in the port (see Assumption 1)	Random between 1 and 100 seconds
Weight of waiting Times for the AGVs in the cost of the objective function	1
Weight of travelling Times for the AGVs in the cost of the Objective Function	5
P as the penalty in the costs of the objective function	10,000

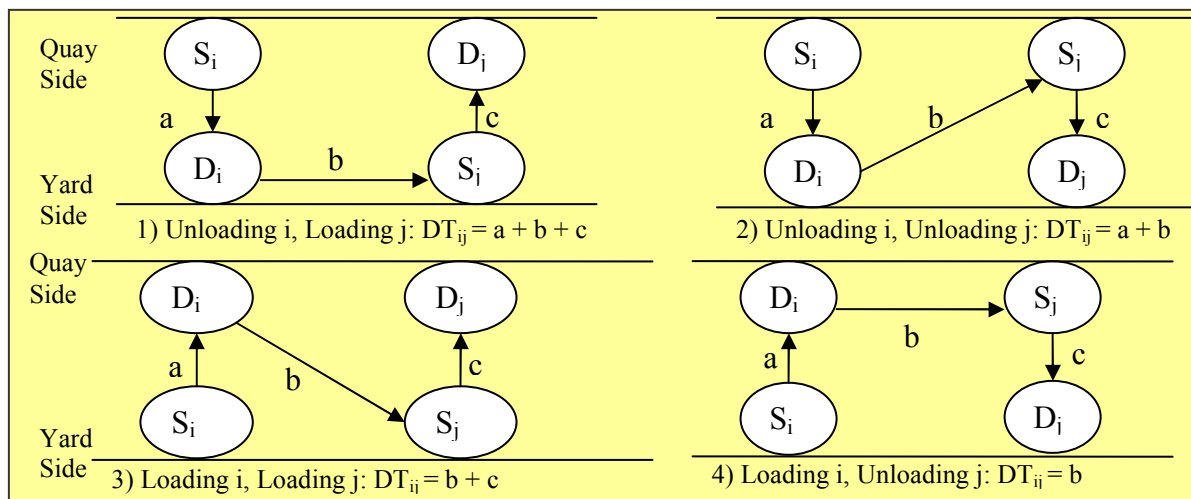


Figure 1: Travelling time computations between two jobs.

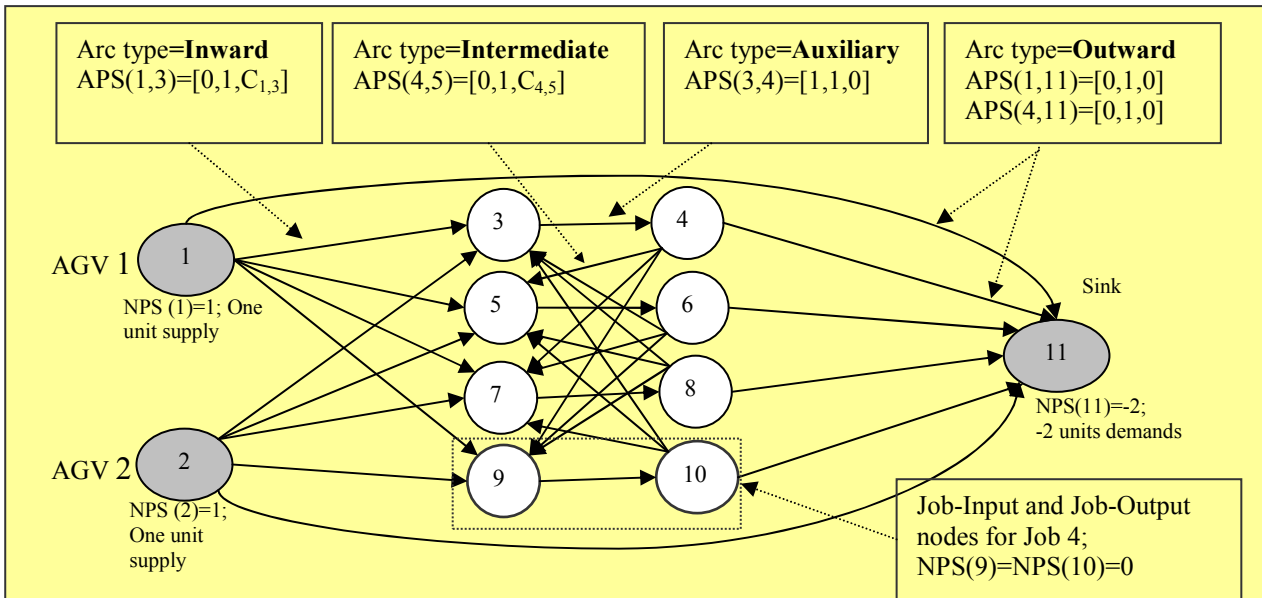


Figure 2: An example of the MCF-AGV model for 2 AGVs and 4 jobs

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

Figure 3: The Network Simplex Algorithm (NSA)

If there is any remaining job and there is any idle vehicle
 Calculate the objective function (v, j) : the travelling times and waiting times of the AGVs as well as the lateness time of jobs.
 For $v=1, 2, \#$ Idle vehicles; for $j=1, 2, \dots, \#$ 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 4: Pseudo code of Simple Heuristic Search

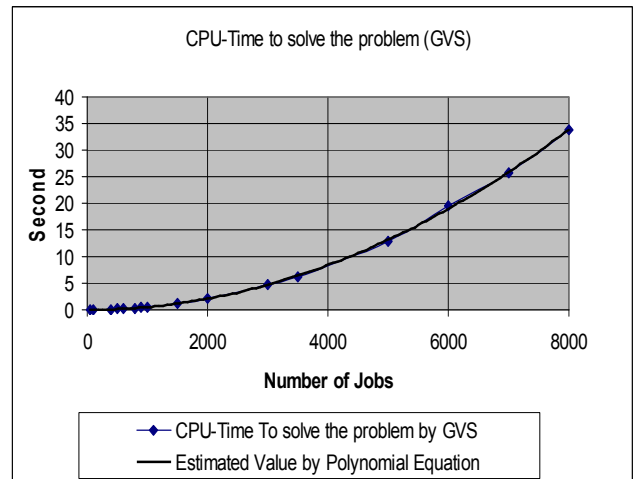
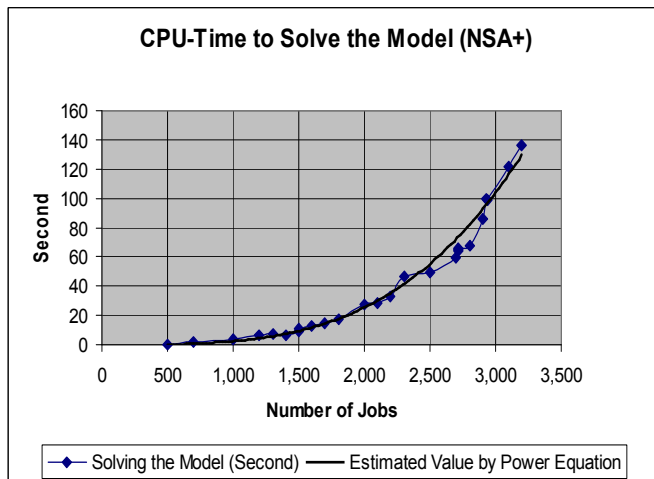


Figure 5: CPU-Time to solve the static problem by NSA+ (left) and GVS (right), based on the number of jobs

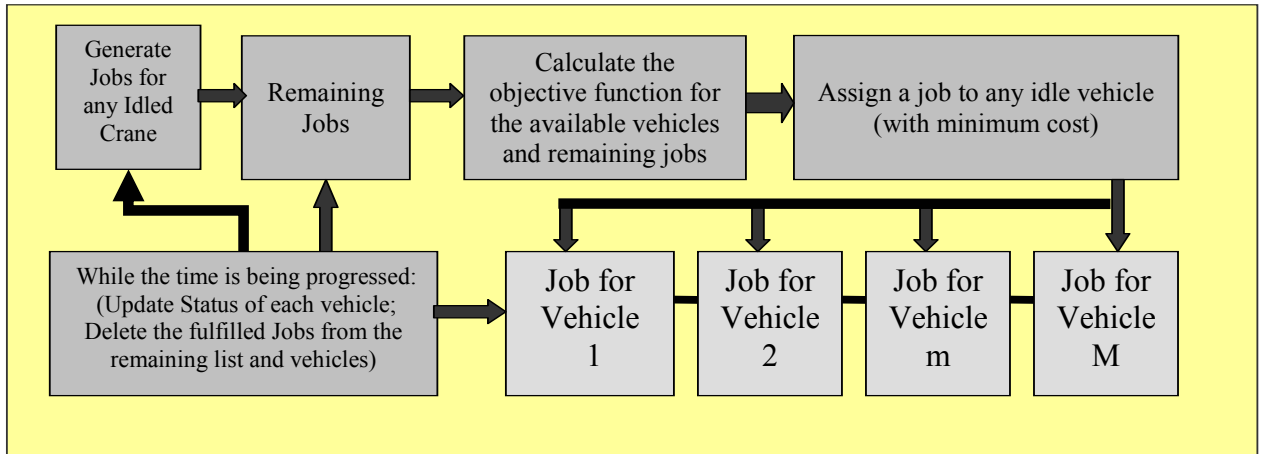


Figure 6: Block Diagram Greedy Vehicle Search in Dynamic Aspect

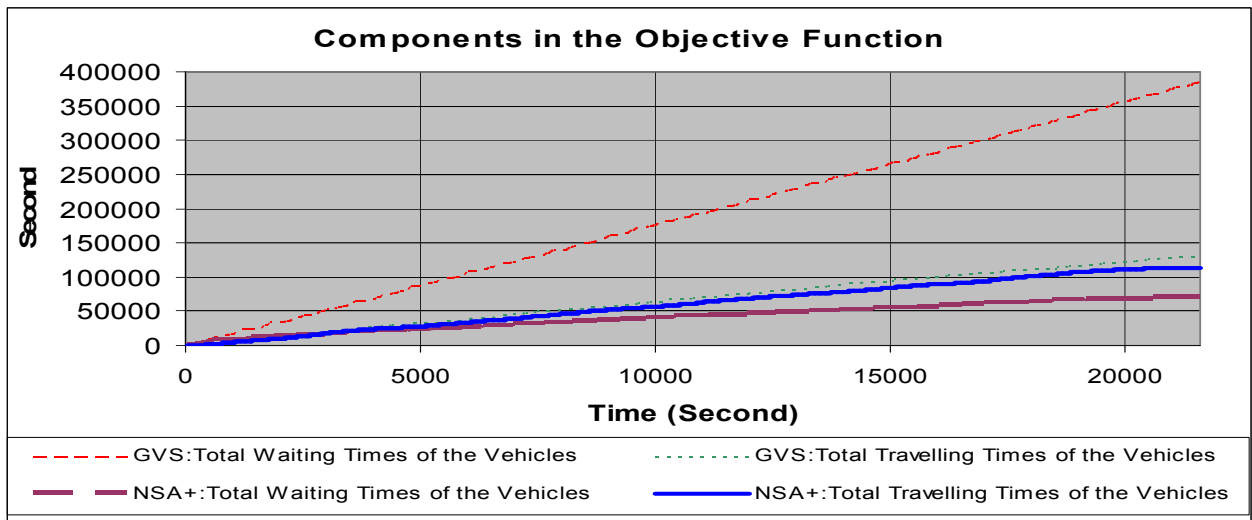


Figure 7: A comparison of Travelling and Waiting Times in dynamic problems

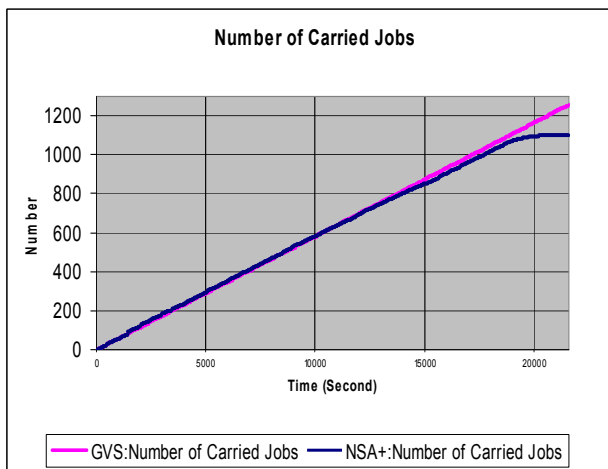


Figure 8: The number of carried jobs by the two algorithms

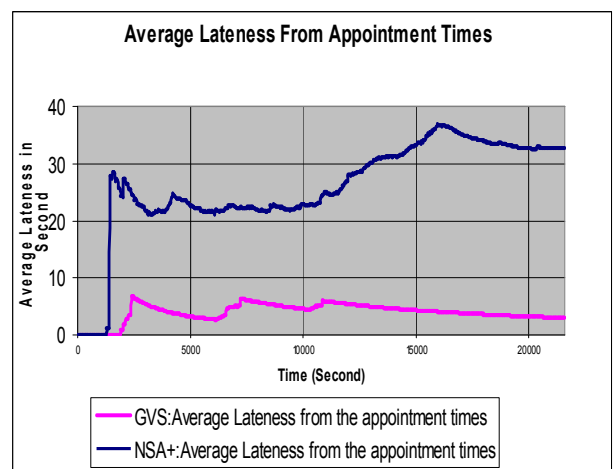


Figure 9: Average Lateness from the appointment time