INTERNATIONAL JOURNAL OF MARITIME TECHNOLOGY
IJMT Vol.11/ Winter 2019 (1-12)

Available online at: http://ijmt.ir/browse.php?a_code=A-10-961-1&sid=1&slc_lang=en

Simulation and Evaluation of Network Simplex Algorithm and its Extensions for Vehicle Scheduling Problems in Ports

Hassan Rashidi

Associate Professor, Faculty of Mathematical Sciences & Computer, Allameh Tabataba'i University; hrashi@gmail.com

ARTICLE INFO

Article History:
Received: 10 Oct. 2018
Accepted: 24 Feb. 2019

Keywords:
Network Simplex Algorithm
Dynamic Network Simplex Algorithm
Optimization Methods
Dynamic Scheduling
Container Terminals

ABSTRACT

The Minimum Cost Flow (MCF) problem is a well-known problem in the area of network optimization. To tackle this problem, Network Simplex Algorithm (NSA) is the fastest solution method. NSA has three extensions, namely Network Simplex plus Algorithm (NSA$^+$), Dynamic Network Simplex Algorithm (DNSA) and Dynamic Network Simplex plus Algorithm (DNSA$^+$). The objectives of the research reported in this paper are to simulate and investigate the advantages and disadvantages of NSA compared with those of the three extensions in practical situations. To perform the evaluation, an application of these algorithms to scheduling problem of automated guided vehicles in container terminal is used. In the experiments, the number of iterations, CPU-time required to solve problems, overheads and complexity are considered.

1. Introduction

One of the most well-known problem in the area of network optimization is the Minimum Cost Flow (MCF) problem. The problem is to send flow from a set of supply nodes, through the arcs of a network, to a set of demand nodes, at minimum total cost, and without violating the lower and upper bounds on flows through the arcs (see [1, 2, 3]). The MCF problem has numerous applications in scheduling, transportation, logistics, and telecommunication.

One of the fastest algorithms to solve the MCF problem is Network Simplex Algorithm (NSA). This algorithm is an adaptation of the bounded variable of traditional primal simplex algorithm in Linear Programming [2], specifically for the MCF problem. In NSA, the basis is represented as a rooted spanning tree of the network graph, in which the arcs represent variables. The algorithm iterates towards an optimal solution by exchanging basic and non-basic arcs in the graph. NSA has three extensions, namely network simplex plus algorithm (see [4]), dynamic network simplex algorithm and dynamic network simplex plus algorithm (see [5]).

To compare the advantages and disadvantages of those algorithms, we choose one of the challenging problems in transportation area. The problem is to schedule a number of Automated Guided Vehicles (AGVs) to transport container jobs inside the terminal statically and dynamically. In the static problem, where there is no change in the situation whereas in dynamic ones, the problem changes over time. The components that are relevant to the problem include quay cranes, container storage areas, and a road network [6]. The 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.

In order to determine to what extent NSA and its extensions can be applied in practice, this paper followed the research done in [5]. The structure of the remaining parts is as follows: Next section is a brief description of the related works over algorithms and problems. Section 3 is a description of the MCF problem in container terminals. Section 4 presents the experimental results in this research. The final section is considered for the summary and conclusion.

2. Related Works

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. Figure 1 shows the pseudo code of Network Simplex Algorithm and its extensions. At the beginning of the algorithm when the software made a MCF model, an initial feasible solution is generated by the procedure Generate-Initial BFS in the *Step 01'*. The operation of this procedure

was described in [2]. In fact, in this step an initial feasible spanning tree solution (T_0, L_0, U_0) is created. In dynamic problems, the *Reconstruct New BFS,' Step 02'*, is executed. When *S* (as the dynamic stage) is zero, the procedure *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 (See [5] for more detail). The main body of the algorithms, NSA and DNSA, are the same. Sis Stage for the dynamic problem and is increased by the dynamic algorithms for each problem. *SODN* is a set of nodes that have to be removed from the model. *SOIN* is a set of nodes that have to be put into the new model. The 'Step 1' in the algorithm selects an entering arc, which is appended to the spanning tree. The 'Step 2' determines the leaving arc, which must be removed from the spanning tree. The 'Step 3' makes pivoting and exchanges the entering and leaving arc. The operation of the main body was described in [4] and [2].

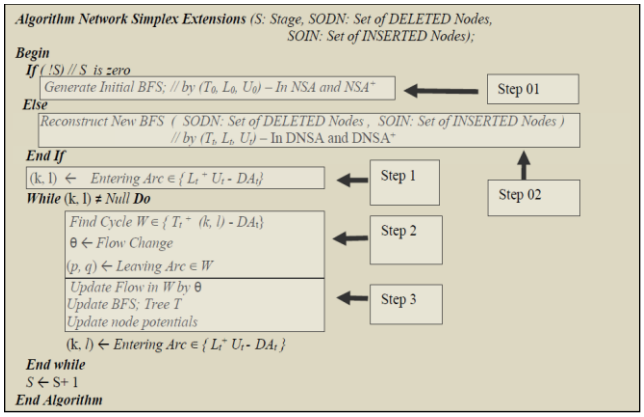

Figure 1. The pseudo code of the Network Simplex Algorithm and its extensions

2.1. The Algorithms NSA and NSA$^+$

The 'Step 1'of the algorithms (see Figure 1) is certainly an important step in the Network Simplex Algorithm (NSA) since the total computational effort to solve a problem heavily depends on its choice. This step is called pricing scheme which does two things. It checks whether the optimality conditions for the non-basic arcs are satisfied, and if not it selects a violated arc to enter the spanning tree structure. The selected arc has a potential of improving the current solution. According to the theory [7], NSA terminates in a finite number of iterations regardless of which profitable candidate is chosen if degeneracy is treated properly. The most well-known schemes in NSA are the *steepest edge scheme* [8], the *Mulvey's list* [9], the *block pricing scheme* [1], the *BBG Queue pricing scheme* [10], *the clustering technique* [11], *the multiple pricing schemes* [12], the *general pricing scheme* [13]. In this paper we present a new pricing scheme, which significantly reduces the CPU-time required to tackle MCF model.

Rashidi and Tsang (2012) develop an extension for network simplex algorithm, namely NSA$^+$ [4]. Compared with the standard version of NSA by Grigoriadis's blocking scheme [1] and maintaining the strongly feasible spanning tree [14], NSA$^+$ has three new features. These features are concerned with the starting point/block for scanning violated arcs, the memory technique and the scanning method. The pricing scheme of NSA$^+$ is designed based on these features. There are two options to choose the first block to be scanned; Randomly and Heuristically. Hence, NSA$^+$ has two extensions: (a) NSA^{+R}: The entering arc function chooses the first block by Random selection; (b) NSA^{+H}: The entering arc function chooses the first block by a Heuristic method and the location of the largest cost in the graph model.

2.2. The Algorithms DNSA and DNSA$^+$

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 edges or vertices. 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 have been discovered. In a typical dynamic graph problem one would like to response to the changes in the graph that are under-going a sequence of updates, for instance, insertions and deletions of edges and vertices. Given their powerful versatility, it is not surprising that dynamic algorithms and dynamic data structures are often more difficult to design and analyze than their static counterparts. The goal of DNSA is to update efficiently the solution of a problem after dynamic changes, rather than having to resolve it from scratch-line each time. The Dynamic Network Simplex Algorithm is based on the Network Simplex Algorithm. DNSA and DNSA$^+$ are the dynamic version of NSA and NSA$^+$, respectively.

The dynamic flows networks over time and their variations are very challenging problems. These types of problems are arising in various real applications such as communication networks, air/road traffic control, and production systems. Some major examples and further applications of the problems are found in the references (see [15,16,17,18,19]). Below we survey the results most closely related to the dynamic network flows and problems.

Rauch (1992) classified dynamic graph problems according to the types of updates allowed [20]. 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 detrimental. DNSA and DNSA$^+$ are fully dynamic.

Afshari and Taghizadeh (2013) present a dynamic version of the maximum flow network in the simplest

kinds of interdiction problem [21]. In the problem, they assume that a positive number is assigned to each arc in the graph model, which indicates the traversal time of the flow through the arcs. Moreover, they assume that an intruder uses a single resource with limited supply to interrupt the flow of a single commodity through the arcs in the network graph within a given limited time period. So the arcs in the graph model is either vital or non-vital. To formulate the problem, a mixed integer mathematical programming model is presented, based on the concept of Temporally Repeated Flow (TRF). The model is then tackled by a couple of algorithms [22]: (a) an algorithm based on the Benders' decomposition and (b) another based on the algorithm of Ratliff et al. (1975) for the most vital arcs [22]. Although they consider a dynamic problem of the network flow model, the algorithms are not dynamic; i.e. without having any exploitation the current solution to respond to the dynamic changes.

Geranis et al. (2012) develop a new Dual Network Exterior-Point Simplex Algorithm (DNEPSA) for the Minimum Cost Network Flow Problem (MCNFP) [23]. The algorithm starts from an initial dual feasible tree-solution and, after a number of iterations, it reaches an optimal solution by producing a sequence of tree solutions that can be both dual and primal infeasible. In following the work, Geranis and Sifaleras (2013) utilize the dynamic trees data structure in the DNEPSA algorithm, in order to achieve an improvement of the amortized complexity per pivot [24]. In extensive computational studies, DNEPSA performed better than the classical dual network simplex algorithm. Although the authors consider a dynamic tree data structure, the problem does not change over time and the algorithm is not dynamic.

Shen et al. (2007) [24] and Zheng and Chiu (2011) [25] worked on a dynamic problem and made simplified System Optimal Dynamic Traffic Assignment (SO-DTA) model. The model is based on the concept of Cell-Transmission Model (CTM), which requires the links in the graph model to be decomposed into cells in space and time. Both works gave definitions on traffic holding in CTM-based on single commodity and single destination problem. Shen et al. (2007) utilized a network flow structure and solved a simplified SO-DTA, thus losing the ability to capture wave propagation and queue spillback effects. They suggested a post-processing algorithm to remove traffic holding from a solution generated by the Linear Programming, but this algorithm depends on the fact that the traffic holding does not improve the objective function value. Zheng and Chiu observed that the definition on diverge node may lead to a suboptimal solution [25] and for the diverge links, it may be better to hold instead of discharge all flow early. So they only applied the definition of holding-free solution to merge and ordinary links. Then, they proved that an augmenting path algorithm produces holding-free solutions at non-diverge links. Therefore, the definitions of holding-free in [24] and [25] are too strict for diverge nodes, the algorithms may lead to suboptimal and are not appropriate for most dynamic problems.

Parpalea (2011) presents an approach for solving bi-criteria minimum cost dynamic flow problem with continuous flow variables [26]. The approach is to transform a bi-criteria problem into a parametric one by making a single parametric linear cost out of the two initial cost functions. The approach iteratively finds efficient extreme points in the decision space by solving a series of minimum parametric cost flow problems with different objective functions. On each of the iterations, the flow is augmented along a minimum path from the supply node to the demand node in the time-space network avoiding the explicit time expansion of the network.

Based on the previous research, Parpalea and Ciurea (2011) represent a generalization of the maximum flow of minimum cost problem for the case of minimizing the travelling cost (minimum cost flow) and travelling time (quickest flow) [27]. On this generalization, the research states a multi-criteria maximum flow problem in discrete dynamic networks with two objective functions. Then a solution method is based on generating efficient extreme points in the search space by iteratively solving a series of maximum flow problems with different single objective functions. Each time, the dynamic flow is augmented along a minimum cost path from the supply nodes to the demand nodes in the time-space network while avoiding the explicit time expansion of the network. Parpalea and Ciurea (2011) also study the generalization of the maximum flow of minimum cost problem for the case of maximum discrete dynamic flow of minimum travelling cost and time [27]. Their approach is very similar to the one used in [26].

Hosseini (2011) introduces a class of dynamic network flows in which the flow commodity is dynamically generated at supply nodes and dynamically consumed at demand nodes [28]. As a basic assumption in this research, the supply nodes produce the flow according to time generative functions and the demand nodes absorb the flow according to time consumption functions. In the general form and some special cases, the dynamic problems arise when the capacities and costs are time varying. This research formulates the problem as the minimum cost dynamic flow problem for a pre-specified time horizon. To solve the problems, some simple and efficient approaches based on the minimum cost static flow models are developed.

Nasrabadi and Hashemi (2007) present a general minimum cost dynamic flow problem in a discrete time model with time-varying transit times, transit costs, transit capacities, storage costs, and storage capacities [29]. For this problem, the authors develop an algorithm, which is a discrete-time version of the

successive shortest path. The time complexity of the algorithm is $O(V\,nT(n+T))$ where V is an upper bound on the total supply, n is the number of nodes, and T denotes the given time horizon of the dynamic flow problem.

Ciurea and Parpalea (2010) present a dynamic solution method for dynamic minimum flow networks [30]. The solution method solves the problem for a special parametric bipartite network [30]. Instead directly work on the original network, the method uses the parametric residual network and finds a particular state of the residual network from which the minimum flow and the maximum cut for any of the parameter values are obtained. The research implements a round-robin algorithm looping over a list of nodes until an entire pass ends without any change of the flow.

Fonoberova (2010) presents other class of dynamic flow networks with the cases of nonlinear cost functions on arcs, multi-commodity flows, and time- and flow-dependent transactions on arcs of the network [31]. All parameters of the networks are assumed to be dependent on time. To formulate the problems, the classical optimal flow problems on networks are extended and generalized. The algorithms for solving such kind of problems are developed by using special dynamic programming techniques based on the time-expanded network method together with classical optimization methods. To solve the problem, the author proposes an approach based on the reduction of the dynamic problem to a static problem. This approach is employed for solving some power systems problems by using optimal dynamic flow problems.

Sherbenym (2012) propose a new version of the minimum cost flow problem on a time varying and time windows [32]. For each vertex in the network, three integer parameters are considered. These parameters are waiting cost, vertex capacity and time windows. In order to obtain dynamic networks, all these parameters are functions of the time. The objective is to find an optimal schedule to send a flow from the supply nodes to its demand nodes so that satisfies a time window constraint with minimum cost and minimum waiting times at nodes, subject to the constraint that the flow must arrive at the demand node before a deadline. In this paper, the algorithm to be developed will search, successively, shortest paths from the supply node, s, to the demand node in a dynamic residual network and then transmit as much as possible flow along the paths so that satisfies the time window constraint.

Fathabadi (2011) proposes a minimum flow problem on network flows in which the lower arc capacities in the graph model vary with time [33]. For a set of time points, this problem is solved by at most n minimum flow computations. The solution method is based on combining of pre-flow-pull algorithm and re-optimization techniques. The complexity of the presented algorithm is $O(n^2m)$ where m is the number of arcs in the graph model.

3. Description of the MCF Problem in Container Terminals

The problem, here, is the same as the problem defined in [34]. The most important reason for choosing this problem is that the efficiency of a container terminal is directly related to the use of the AGVs with full efficiency (see [7, 35, 36, 37, 38, 39, 40]). The assumptions used are also the same as the assumptions in [34]. The MCF associated with the problem is presented as MCF-AGV model [41]. The MCF-AGV model was established on a directed graph. Figure 2 demonstrates an example of the problem for two AGVs and four container jobs. As in the paper mentioned, the problem was formalized with four different types of node: a supply node for each AGV (nodes 1 and 2 in Figure 2), a couple of nodes for each container job (nodes 3-10 in Figure 2) as transshipment nodes and a demand node (the node 11 in Figure 2).

The following four types of arc, namely Inward Arcs, Intermediate Arcs, Outward Arcs and Auxiliary Arcs with their properties connect the nodes in the graph model. The Inward Arcs are directed arcs from the each AGV node to the each Job-Input node. The Intermediate Arcs are directed arcs from the each Job-Output node to the others Job-Input node. The Outward Arcs are directed arcs from the each Job-Output node and the each AGV node to the SINK. The Auxiliary Arcs are directed arcs from every Job-Input node to its Job-Output node. For more details on the nodes and arcs refer to [41].

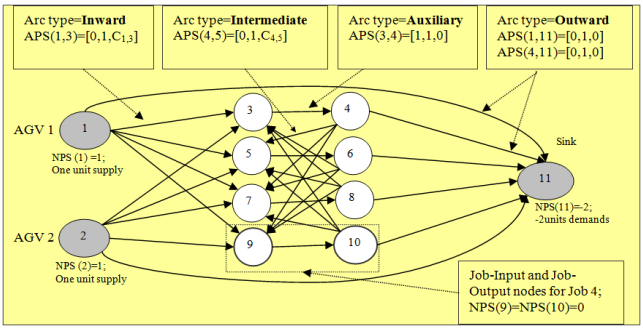

Figure 2. An example of the MCF-AGV model of two AGVs and four container jobs

Suppose that for some values of the arc costs in the model, the solution paths are 1→3→4→9→10→11 and 2→5→6→7→8→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.

4. Simulation Results and Comparisons

We implemented the simulation software in Borland C++Builder, running on Genuine Intel 3.081Ghz Processor. Figure 3 shows the main screenshot of the software. It shows a single vessel, four 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. Some important features of the software are described briefly as follows (for more detail see [42]):

- 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.
- A Job Generator was designed and implemented in the software. For static and dynamic fashion, a few container jobs are generated to transport from their source to their destination. Either the source or destination of each job is the quayside, which can be chosen randomly by the Job Generator. The initial time of the operation and the time window for the cranes and vehicles are defined by the user. 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.

Figure 3. The main screenshot of the simulation software

4.1 Memory Management of the Simulation Software

Given N jobs and M AGVs in the problem, there are M+2×N+1 nodes and M+M×N+N×(N-1)+2×N arcs in the MCF-AGV model [41]. 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-1 shows a memory map of the allocated space. As shown in the table, there were four different types of arc in the MCF-AGV model: Inward Arcs, Outward Arcs, Auxiliary Arcs, and Intermediate Arcs (see Figure 2). Additionally, the Artificial Arcs are needed to generate an initial Basic Feasible Solution [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 a certain type of arc, it is necessary to have an offset. The offset is the difference in the address from the beginning of the block.

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

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

In the software, 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 and prevent any missing job (when the Job Generator generates a job and the memory cannot be allocated).

In fact, 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 [5]) 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 belonging 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, a Remove-Node procedure is used [5]. When a new job arrives, the relevant nodes (which have 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 an Insert-Node procedure, which is presented in [5].

4.2. Simulation and Evaluation in Static Problems

To simulate and evaluate the performance of the algorithms, many jobs in static and dynamic fashion have been generated. In our experiment, it was assumed that there were fifty AGVs and seven cranes in the port. Other experimental parameters are the same as in [41]. Their sources, destinations and the distance between every two points in the port have been chosen randomly.

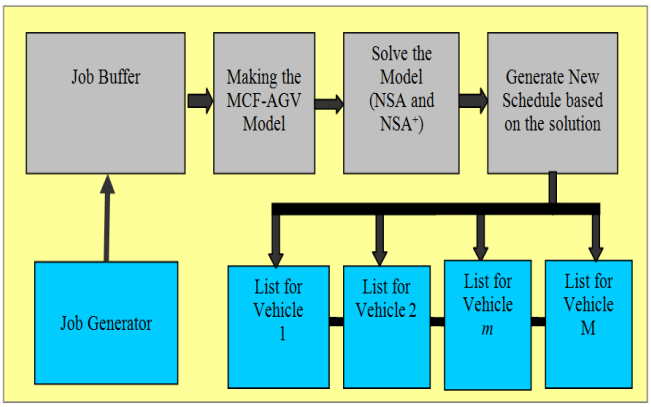

Figure 4. Block diagram of the software executed for solving static problems [43]

We generated 32 static random problems by which must be solved by the algorithms. Figure 5 shows the CPU-Time required to solve the problems by NSA, NSA^{+H} and NSA^{+R}, based on the number of container jobs in the static problem.

Although NSA^{+} is faster than NSA [41], it has some overhead as a cost. In 'Step 1' of the algorithm (see Figure 1), NSA^{+R} chooses an entering arc from the first block randomly. NSA^{+H} chooses an entering arc from the first block by a Heuristic method. This heuristic is based on the location of the largest cost in the graph model into which must be searched. In fact, it chooses the arc with the largest cost. Hence it has some overheads due to the search needed. Figure 5 shows the overhead of the algorithms NSA^{+H} and NSA^{+R} compared with zero for NSA, based on the number of container jobs in the static problem. The overhead is determined in the number of high level instructions needed to solve the problem.

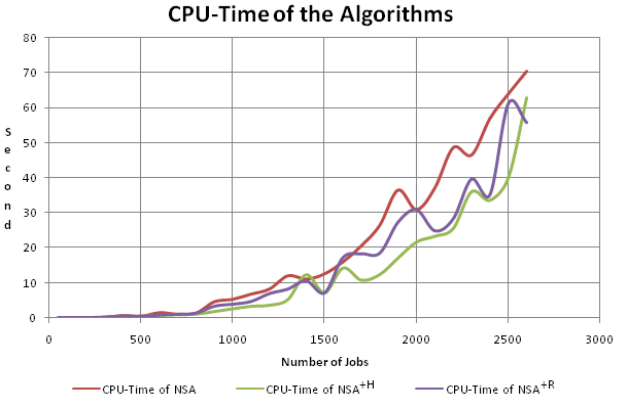

Figure 5. The overhead of NSA^{+H} and NSA^{+R} compared with that of NSA

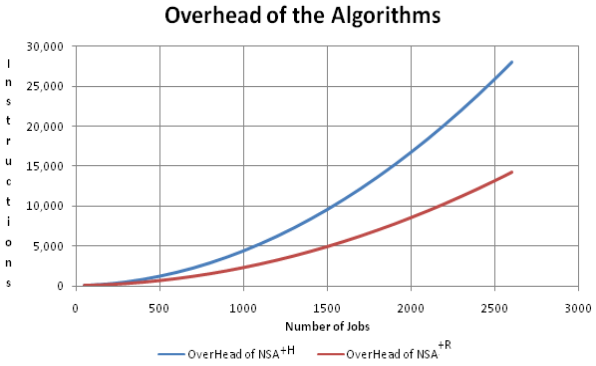

Figure 6. A comparison of CPU-Time required solving the same problems by NSA, NSA^{+H} and NSA^{+R}

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:

$\boldsymbol{CPU - T_i^{NSA}}$**:** The CPU-Time required to solve the problem i by NSA.

$\boldsymbol{CPU - T_i^{NSA^{+H}}}$:The CPU-Time required to solve the problem i by NSA^{+H}.

$\boldsymbol{CPU - T_i^{NSA^{+R}}}$**:** The CPU-Time required to solve the problem i by NSA^{+R}.

PIH$_i$: The Percentage of Improvement in CPU-time required to solve the problem i by NSA^{+H} compared with that of NSA.

PIR$_i$: The Percentage of Improvement in CPU-time required to solve the problem i by NSA^{+R} compared with that of NSA.

TPIH: The Total Percentage of Improvement in CPU-Time required to solve the problems by NSA^{+H} compared with that of NSA.
TPIR: The Total Percentage of Improvement in CPU-Time required to solve the problems by NSA^{+R} compared with that of NSA.
TPIHR: The Total Percentage of Improvement in CPU-Time required to solve the problems by NSA^{+H} compared with that of NSA^{+R}.
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×N+N×(N-1)+2×N.

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

$$TPIH = \frac{\sum_{i=1}^{32} W_i \times \left(CPUT_i^{NSA^{+H}} - CPUT_i^{NSA}\right)}{\sum_{i=1}^{32} W_i} \times 100 = 32.99 \quad (1)$$

$$TPIR = \frac{\sum_{i=1}^{32} W_i \times \left(CPUT_i^{NSA^{+R}} - CPUT_i^{NSA}\right)}{\sum_{i=1}^{32} W_i} \times 100 = 21.94 \quad (2)$$

$$TPIHR = \frac{\sum_{i=1}^{32} W_i \times \left(CPUT_i^{NSA^{+H}} - CPUT_i^{NSA^{+R}}\right)}{\sum_{i=1}^{32} W_i} \times 100 = 14.15 \quad (3)$$

The percentages of overhead in the number of high level instructions used to solve the problems by NSA^{+H}, NSA^{+R}, and NSA are calculated by the similar expressions. In this comparision, the average overhead of the algorithms NSA^{+H} and NSA^{+R} are comapred with that of NSA. Table-2 shows the results of the comparison between the algorithms in their CPU-Time and overheads.

Table 2. The results of the comparison between the algorithms in their CPU-Time and their overhead

	CPU-Time			Overhead		
	NSA	**NSA^{+H}**	**NSA^{+R}**	**NSA**	**NSA^{+H}**	**NSA^{+R}**
NSA	0	-32.99	-21.94	0	14	8
NSA^{+H}	32.99	0	-14.15	-14	0	6
NSA^{+R}	21.94	14.15	0	8	-6	0

Observation-1: NSA^{+H} and NSA^{+R} are 33 and 22 percents, respectively, faster than NSA. NSA^{+H} is 14 percent faster than NSA^{+R}

Observation-2: The overhead of NSA^{+H} and NSA^{+R} are around 14 and 8 percents, respectively, compared with that NSA. The overhead of NSA^{+H} is 6 percent more than NSA^{+R}
The CPU-Time and time complexity of the algorithms can be examined in the experiments. We did a rgression on the CPU-Time required in running the algorithms. Given N as the the number of jobs in the graph model, we obtained the following equations to estimate the CPU-Time:

$$CPU\text{-}Time_{NSA}(N) = 3E\text{-}09N^3 + 3E\text{-}06N^2 - 0.001N \quad R^2=0.991 \quad (4)$$

$$CPU\text{-}Time_{NSA^{+H}}(N) = 6E\text{-}09N^3 - 9E\text{-}06N^2 + 0.005N \quad R^2=0.962 \quad (5)$$

$$CPU\text{-}Time_{NSA^{+R}}(N) = 3E\text{-}09N^3 - 4E\text{-}07N^2 + 0.001N \quad R^2=0.959 \quad (6)$$

The coefficient R^2 in the regression reveals how closely the values of the estimated curve correspond to the actual data. Its value is more than 0.95 for the estimations.

Observation-3: According to the equations (4), (5) and (6), the complexity of the algorithm , NSA, NSA^{+H} and NSA^{+R}, are in order 3 of the number of jobs.
The overhead of the algorithms, NSA^{+H} and $NSA^{+R}$are examined in the experiments. We did a rgression on the CPU-Time required in running the algorithms. Given N as the the number of jobs in the graph model, we obtained the following equations to estimate the CPU-Time:

$$OV_{NSA^{+H}}(N) = 0.004N^2+0.366N \quad R^2=0.999 \quad (7)$$
$$OV_{NSA^{+R}}(N) = 0.002N^2+0.264N \quad R^2=0.999 \quad (8)$$

Observation-4: According to the equations (7) and (8), the overhead of NSA^{+H} and NSA^{+R} are in order 2 of the number of jobs.
Note that for any prediction the equation for the CPU-Time in practice depends on other factors, such as the speed of processor, active programs when the problem is being solved in multi-task operating systems, and so on. Our program has been run on a Windows XP computer with a Genuine Intel 3.081Ghz Processor in the normal situation.
The performnce of running the two algorithms has been analyzed statistically. We tested the nullhypothesis that the means produced by the two algorithms werestatistically indifferent ($\alpha = 5\%$). Table-3 provides the test's resultalong with the values of T-distribution for a particular degree offreedom. Since we cared if the change (the difference between thetwo means) was positive or negative, 'One-tail' test was chosen.

Table 3. The statistical test over the results of the comparison in static aspect

	CPU-Time		Overhead	
	NSA^{+H} vs. NSA	NSA^{+R} vs. NSA	NSA^{+H} vs.NSA	NSA^{+R} vs. NSA
Observations	32	32	32	32
T-Test (Paired Two Sample For Means)	-4.11	-3.36	46.1	25.3
Degree of Freedom	31	31	31	31
Critical T-Value	1.69	1.69	1.69	1.69

Observation-5: Table-3 shows that although NSA^{+H} and NSA^{+R} statistically ate better than NSA, the overhead of these algorithms are significant comapred with that of NSA.

4.3 Simulation and Evaluation in Dynamic Problems

The problem defined in [34] is dynamic. In reality, the dynamic problem arises when several new jobs are arrived, the fulfilled jobs are removed and the links or junctions in the port layout are blocked. For the arriving jobs, the Job Generator has to generate a few new jobs, when it finds out any crane is in idle state. The fullfilling jobs must be removed from the graph model by the software. When the links or junctions in the port layout are blocked, the software must make the changes in distances between points in the source and destination of the jobs.

The architecture of the simulation software for running NSA and NSA^{+} is demonstrated in Figure 7. 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 are 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.

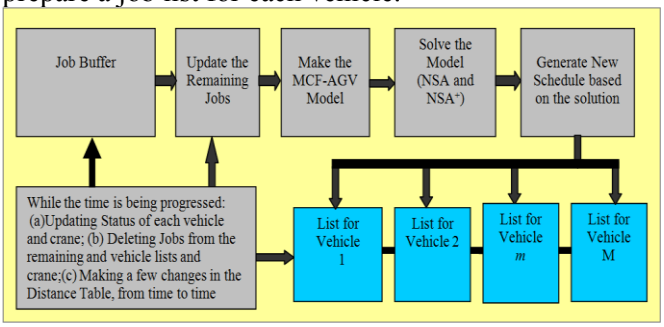

Figure 7. Block diagram of the simulation software and algorithm NSA and NSA^{+} for solving dynamic problems [43]

At the beginning, based on the solution to the problem at the current stage, 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 the status of the vehicles and cranes whereas the second one takes influence from any change in the problem or any idle crane. 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 are any). The new model will be tackled by the algorithms from scratch. Then, the new solution will be used for updating the list of jobs for every vehicle.

The main architecture of the simulation software for running the algorithms is demonstrated in Figure 8. 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 the algorithms. 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. While the time is being progressed, the vehicles and cranes are carrying and handling the containers.

As it is shown in the figure, every event is recorded in order to be processed later. The events include modification of the vehicle's position, the fulfilled jobs and new jobs, and any change in the distance table. A hole will be created in the Job Buffer when a job is fulfilled [42]. 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, does not affect the spanning tree. In this case, only the body of the algorithm is executed and finds out the optimal solution.

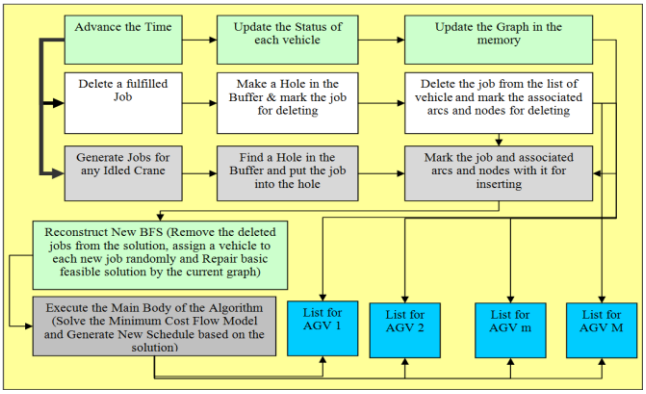

Figure 8. Block diagram of the simulation software for solving dynamic problems [43]

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 of the algorithm is executed and it finds out the optimal solution. Note that these tasks are not pre-emptive, i.e. when a task starts execution on the processor it finishes to its completion.

Figure 9 shows the number of jobs arrived, the number of jobs fulfilled and the number of jobs remained in each stage of the dynamic problems. The relation between these numbers of jobs is as according to the equation (9):

$$\#JobsRemained\ (S) = \#JobsRamined(S-1) + \#JobsArrived(S) - \#JobsFullfilled(S) \quad (9)$$

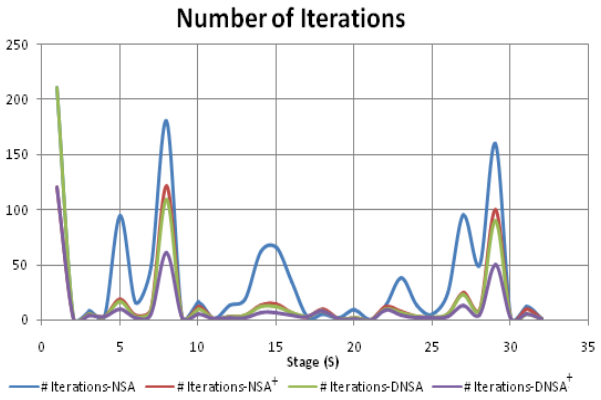

Figure 9. The number of jobs arrived, fullfilled and remained in the dynamic problems

Figure 10 shows the percentages of changes made in the graph model, due to the number of jobs arrived and the number of jobs fulfilled in each stage of the dynamic problems. The values in the figure are calculated based on the number of nodes and arcs in the graph model for insertion and deletion, according to the number of jobs arrived and fulfilled at each stage. The arcs and nodes for jobs arrived (fulfilled) must be inserted (deleted) into (from) the graph model. The number of nodes and arcs are calculated according to the simple equations like ones shown in Figure 2. Given *#ChIns(S)* as the value of changes due to insertion some nodes with their arcs, and *#ChDel(S)* as the value of changes due to deletion some nodes with their arcs at each stage, the percentage of changes in the graph is calculated according to equation (10):

$$ChangesInGraphModel\ (S) = \left| \frac{\#ChIns(S) + \#ChDel(S) - (\#ChIns(S-1) + \#ChDel(S-1))}{\#ChIns(S-1) + \#ChDel(S-1))} * 100 \right| \quad (10)$$

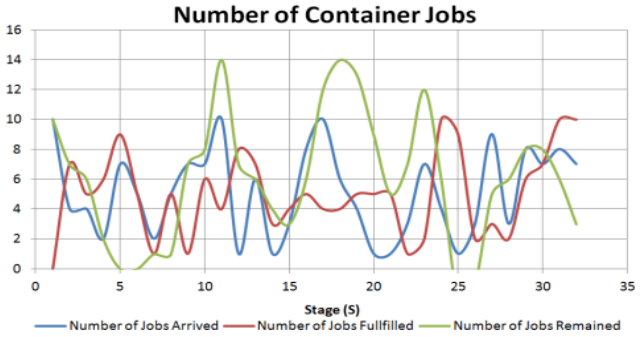

Figure 10. The percentages of changes in the graph model of the dynamic problems

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. Moreover, the CPU-Time required to solve the problem is too much small and is not convenient for the comparision. Hence, the number of iterations is considered as an indicator to compare the algorithms. The number of iterations required to solve the problems are drawn in Figure 11.

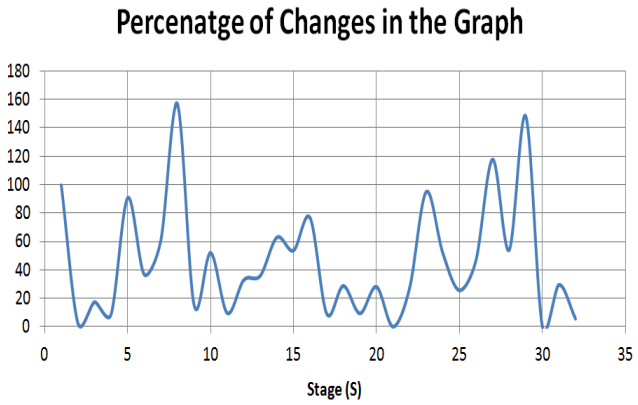

Figure 11. The number of iterations of the algorithms for solving the dynamic problems

From Figure 11, it is clear that the number of iterations are improved when we dynamic algorithm DNSA and DNSA$^+$ compared with that of NSA and NSA$^+$. Note that since NSA^{+H} perform better than NSA^{+R} (see Observation-2), we use only NSA^{+H} in this experiments. The percentage of improvement, in reduction of the number of iterations, is calculated by the following terms and equation:

NSA$_S$: The number of iterations in NSA for the dynamic problem at stage *S*.

$\boldsymbol{NSA_S^+}$: The number of iterations in NSA$^+$ for the dynamic problem at stage *S*.

DNSA$_S$: The number of iterations in DNSA for the dynamic problem at stage *S*.

$\boldsymbol{DNSA_S^+}$: The number of iterations in DNSA$^+$ for the dynamic problem at stage *S*.

$\mathbf{TPR}_{\mathbf{NSA^+}}^{\boldsymbol{NSA}}$: The Total Percentages of Reduction in the number of iterations in the experiment.
$\boldsymbol{TPR}_{\boldsymbol{DNSA}}^{\boldsymbol{NSA}}$: The Total Percentages of Reduction in the number of iterations in the experiment.
$\boldsymbol{TPR}_{\boldsymbol{DNSA^+}}^{\boldsymbol{NSA}}$: The Total Percentages of Reduction in the number of iterations in the experiment.

$$\mathrm{TPR}_{NSA^+}^{NSA} = \frac{\sum_{S=1}^{32}(NSA_S - NSA_S^+)}{\sum_{S=1}^{32} NSA_S} \times 100 = -58.59\% \qquad (10)$$

Similar equations are used to compare the performance algorithms in the number of iteration required to solve the problems. Table-4 shows this comparisions.

Table 4. The percentages of the performace comparisons between the algorithms

Algorithms	NSA	NSA$^+$	DNSA	DNSA$^+$
NSA	0.00	-58.59	-63.02	-77.49
NSA$^+$	58.59	0.00	-10.68	-45.63
DNSA	63.02	10.68	0.00	-39.13
DNSA$^+$	77.49	77.49	39.13	0.00

From this table, we can obtaine the following observations:

Observation-6: The performace of DNSA$^+$, DNSA and NSA$^+$ are around 77.5, 63 and 60 percents better than that of NSA, respectively.
Observation-7: The performace of DNSA$^+$ and DNSA are around 45.6 and 10.60 percent faster than that of NSA$^+$, respectively.

Observation-8: Since the major process of the algorithms is performed in the body and the operations of the body are identical [42], the CPU-time required to solve the problems is also decreased practically.
The number of iterations of running the two algorithms, DNSA$^+$ and NSA$^+$, has been analysed statistically. We tested the null hypothesis that the means produced by the two algorithms were statistically indifferent (α=5%). Then, we got the following observation:

Observation-9: The Paired T-test determines the two means are significantly different at a ninety-five percent degree of confidence since the test's result is in the reject region.
It is seemed that there is strong correlation between the percentgaes made on the graph model and thenumber of iterations required to solve the problem. So, we decided to calcualte the correlation between them. Table-5 shows the result of this experiment.

Table 5. The correlation between the perentages of changes in the graph and the algorithms

	%Changes-in the Graph	# Iterations- NSA	# Iterations-NSA$^+$	# Iterations-DNSA	# Iterations-DNSA$^+$
%Changes in the Graph	1.00	0.87	0.77	0.63	0.62
# Iterations-NSA	0.87	1.00	0.94	0.89	0.88
# Iterations-NSA$^+$	0.77	0.94	1.00	0.94	0.93
# Iterations-DNSA	0.63	0.89	0.94	1.00	1.00
# Iterations-DNSA$^+$	0.62	0.88	0.93	1.00	1.00

Observation-10: From Table-5, it is clear that the order of the algorithms, NSA, NSA$^+$, DNSA and DNSA$^+$, to solve the dynamic problem have a proportion of 87, 63 and 62 percents, repectively, of changes made in the graph model. It shows the algorithms NSA and NSA$^+$ use more attemps to solve the dynamic problems. The complexity of the algorithms are the same (see [41]). In theory, the total complexity of the algorithms for the problem is:$O(N^6)$

5. Summary and Conclusion

This paper followed the research done in [5]. In fact, in order to determine to what extent these algorithms can be applied in practice, we did the experimental experiments and several comparisons in running NSA, NSA$^+$, DNSA and DNSA$^+$. To evaluate the performance of the algorithms, the dynamic scheduling problem of AGVs in the container terminal (the problem defined in [34] was considered. Many random problems have been generated and solved by both DNSA$^+$ and NSA$^+$. The results showed considerable improvements in DNSA$^+$, in terms of reducing the number of iterations, compared with that of NSA$^+$.
To conclude Network Simplex Algorithm and its three extensions (NSA$^+$, DNSA and DNSA$^+$), in dynamic problems, NSA and NSA$^+$ start from scratch and reconsider the pre-established schedules. Memory management in these two algorithms is an 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 lies in taking time to rebuild the graph and putting it into the 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 costs that have to be paid in return for the performance.

6. References

1- Grigoriadis, M. (1986). An Efficient Implementation of the Network Simplex Method. Mathematical Programming Study, 26, 83-111.

2- Ahuja, R., Magnanti, T., & Orlin, J. (1993). Network lows: Theory, Algorithms and Applications. Prentice Hall.

3- Kelly, D., & ONeill, G. (1993). The Minimum Cost Flow Problem and The Network Simplex Solution Method (Master degree dissertation).University College, Dublin.

4- Rashidi, H., & Tsang, E. (2011). A Complete and an Incomplete Algorithm for Automated Guided Vehicle Scheduling in Container Terminals. Journal of Computers and Mathematics with Applications, 61, 630-641. doi:10.1016/j.camwa.2010.12.009.

5- Rashidi, H. (2014). A Dynamic Version for the Network Simplex Algorithm. Journal of Applied Soft Computing, 24, 414-422. doi:10.1016/j.asoc.2014.07.017.

6- Parpalea, M., & Ciurea, E. (2011). Maximum Flow of Minimum Bi-Criteria Cost in Dynamic Networks. Recent researches in computer science, 118-123.

7- Wook, B., & Hwan, K. (2000). A pooled dispatching strategy for automated guided vehicles in port container terminals. International Journal of Management Science, 6(2), 47-60.

8- Goldberg, A., & Kennedy, R. (1993). An efficient cost scaling algorithm for the assignment problem.Technical Report, Stanford University.

9- Mulvey, J. (1978). Pivot Strategies for Primal Simplex Network Codes. Association for Computing Machinery Journal, 25, 266-270. doi:10.1145/322063.322070.

10- Bradley, G., Brown, G., & Graves, G. (1977). Design and Implementation of Large Scale Primal Transshipment Algorithms. Management Science, 24, 1-38. doi:10.1287/mnsc.24.1.1.

11- Eppstein, D. (1999). Clustering for faster network simplex pivots. In Proceedings of the 5th ACM-SIAM Symposium, Discrete Algorithms, 160-166.

12- Lobel, A. (2000). A Network Simplex Implementation. Technical Report, Konrad-Zuse-Zentrumfur Informations technik Berlin (ZIB).

13- Maros, I. (2003). A General Pricing Scheme for the Simplex Method. Technical Report, London, Department of Computing, Imperial College.

14- Cunningham, W. (1979). Theoretical properties of the network simplex method. Mathematics of Operations research, 4(2), 196-208. doi:10.1287/moor.4.2.196.

15- Aronson, J. (1989). A Survey of Dynamic Network Flows. Annal of Operation research, 20, 1-66. doi:10.1007/BF02216922.

16- Skutella, M. (2009). An Introduction to Network Flows Over Time. Research Trends in Combinatorial Optimization, Berlin: Springer.

17- Powell, W., Jaillet, P., & Odoni, A. (1995). Stochastic and Dynamic Networks and Routing. Handbooks in Operations Research and Management Science (pp. 141-295). Amsterdam: North-Holland.

18- Hoppe, B. (1995). Efficient Dynamic Network Flow Algorithms (Doctoral dissertation). Cornell University, New York.

19- Fonoberova, M., & Lozovanu, D. (2007). Optimal Dynamic Flows in Networks and Applications. The International Symposium the Issues of Calculation Optimization, Communications.Crimea, Ukraine, pp. 292-293.

20- Rauch, M. (1992). Fully Dynamic Graph Algorithms and Their Data Structures (Doctoral dissertation).Princeton University, New Jersey.

21- Afshari Rad, M., & Taghizadeh Kakhki, H. (2013). Maximum Dynamic Network Flow Interdiction Problem: New Formulation and Solution Procedures Original Research Article. Computers & Industrial Engineering, 65(4), 531-536. doi:10.1016/j.cie.2013.04.014.

22- Ratliff, H., Sicilia, G., & Lubore, S. (1975). Finding the n most vital links in flow networks. Management Science, 21, 531-539. doi:10.1287/mnsc.21.5.531.

23- Geranis, G., Paparrizos, K., & Sifaleras, A. (2012). On a Dual Network Exterior Point Simplex Type Algorithm and Its Computational Behavior. Operations Research, 46, 211-234. doi:10.1051/ro/2012015.

24- Shen, W., Nie, Y., & Zhang, H. (2007). A Dynamic Network Simplex Method for Designing Emergency Evacuation Plans. Transportation Research Record, 20(22), 83-93.

25- Zheng, H., & Chiu, Y. (2011). A Network Flow Algorithm for the Cell-Based Single-Destination System Optimal Dynamic Traffic Assignment Problem. Transportation Science, 45(1), 121-137. doi:10.1287/trsc.1100.0343.

26- Parpalea, M. (2011). A Parametric Approach to the Bi-criteria Minimum Cost Dynamic Flow Problem. Open Journal of Discrete Mathematics, 1(3), 116-126. doi:10.4236/ojdm.2011.13015.

27- Parpalea, M., & Ciurea, E. (2011). The Quickest Maximum Dynamic Flow of Minimum Cost. Journal of Applied Mathematics and Informatics, 5(3), 266-274.

28- Hosseini, S. (2010). An Introduction to Dynamic Generative Networks: Minimum Cost Flow. Applied Mathematical Modelling, 35(10), 5017-5025. doi:10.1016/j.apm.2011.04.009.

29- Nasrabadi, E., & Hashemi, S. (2010). Minimum Cost Time-Varying Network Flow Problems. Optimization Methods and Software, 25(3), 429-447. doi:10.1080/10556780903239121.

30- Ciurea, E., & Parpalea, M. (2010). Minimum Flow in Monotone Parametric Bipartite Networks. NAUN International Journal of Computers, 4(4), 124-135.

31- Fonoberova, M. (2010). Algorithms for Finding Optimal Flows in Dynamic Networks. S. Rebennack et

al. (eds.), Handbook of Power Systems II, Energy Systems, Springer-Verlag Berlin Heidelberg.

32- El-Sherbenym, N. (2012). A New Class of a Minimum Cost Flow Problem on a Time Varying and Time Window. Scientific Research and Impact, 1(3), 18-28.

33- Salehi Fathabadi, H., Khodayifar, S., & Raayatpanah, M. (2012). Minimum flow Problem on network flows with time-varying bounds. Applied Mathematical Modeling, 36(9), 4414-4421. doi:10.1016/j.apm.2011.11.067.

34- Rashidi, H., & Tsang, E. (2005). Applying the Extended Network Simplex Algorithm and a Greedy Search Method to Automated Guided Vehicle Scheduling. the 2nd Multidisciplinary International Conference on Scheduling: Theory & Applications (MISTA). New York, p. 677-693.

35- Grunow, M., Gunther, H., & Lehmann, M. (2004). Dispatching multi-load AVGs in highly automated seaport vontainer terminals. OR Spectrum, 26(2), 211-235. doi:10.1007/s00291-003-0147-1.

36- Murty, K., Jiyin, L., Yat-Wah, W., Zhang, C., Maria, C., Tsang, J., & Richard, L. (2002). A Decision Support System for operations in a container terminal. Decision Support System, 39, 309-332.

37- Huang, Y., & Hsu, W. (2002). Two Equivalent Integer Programming Models for Dispatching Vehicles at a Container Terminal. Report No. 639798, Nan yang Technological University, School of Computer Engineering.

38- Cheng, Y., Sen, H., Natarajan, K., Ceo, T., & Tan, K. (2003). Dispatching Automated Guided Vehicles in A Container Terminal. Technical Report, National University of Singapore.

39- Patrick, J., & Wagelmans, P. (2001). Dynamic Scheduling of Handling Equipment at Automated Container Terminals. Report No. EI 2001-33, Erasmus University of Rotterdam, Econometric Institute.

40- Bose, J., Reiners, T., Steenken, D., & Vob, S. (2000). Vehicle Dispatching at Seaport Container Terminals Using Evolutionary Algorithms. In Proceedings of the 33rd Annual Hawaii International Conference on System Sciences. Hawaii, p. 1-10.

41- Rashidi, H., & Tsang, E. (2011). A Complete and an Incomplete Algorithm for Automated Guided Vehicle Scheduling in Container Terminals. Journal of Computers and Mathematics with Applications, 61, 630-641. doi:10.1016/j.camwa.2010.12.009.

42- Rashidi, H. (2006). Dynamic Scheduling of Automated Guided Vehicles in Container Terminals (Doctoral dissertation).University of Essex, Colchester.

43- Rashidi H., Tsang E. (2016). Vehicle Scheduling in Port Automation: Advanced Algorithms for Minimum Cost Flow Problems, Second Edition. CRC Press, New York.