Applied Soft Computing 24 (2014) 414–422

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

# A dynamic version for the Network Simplex Algorithm

Hassan Rashidi*

*Department of Mathematics and Computer Science, Allameh Tabataba'i University, Tehran, Iran*

ARTICLE INFO

*Article history:*
Received 12 March 2012
Received in revised form 16 July 2014
Accepted 16 July 2014
Available online 28 July 2014

*Keywords:*
Dynamic scheduling
Dynamic Network Simplex Algorithm
Optimization methods
Container terminals

ABSTRACT

In most practical environments, scheduling is an ongoing reactive process where the presence of real time information continually forces reconsideration and revision of pre-established schedules. The objectives of the research reported in this paper are to respond to changes in the problem, to solve the new problem faster and to use some parts of the previous solution for the next problem. In this paper, based on Network Simplex Algorithm, a dynamic algorithm, which is called Dynamic Network Simplex Algorithm (DNSA), is presented. Although the traditional network simplex algorithm is at least one hundred times faster than traditional simplex algorithm for Linear Programs (through specialization), for dynamic scheduling with large scale problems it still takes time to make a new graph model and to solve it. The overall approach of DNSA is to update the graph model dynamically and repair its spanning tree by some strategies when any changes happen. To test the algorithm and its performance, an application of this algorithm to Dynamic Scheduling of Automated Guided Vehicles in the container terminal is used. The dynamic problem arises when new jobs are arrived, the fulfilled jobs are removed and the links or junctions are blocked (which results in distances between points being changed). The results show considerable improvements, in terms of reducing the number of iterations and CPU time, to solve randomly generated problems.



## Introduction

The minimum cost flow (MCF) problem is a well-known problem in the area of network optimization, i.e. 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 Refs. [2,6,16]). The MCF problem has numerous applications in scheduling, transportation, logistics, and telecommunication. This paper is motivated by a need to schedule a number of Automated Guided Vehicles (AGVs) in the container terminals with some dynamic changes. The components that are relevant to the problem include quay cranes, container storage areas, and a road network [11]. A transportation requirement in a port is described by a set of jobs, where each job is characterized by the source location of a container, the target location and the time of its picking up or dropping-off on the quay-side by the quay crane. Given a number of AGVs and their availability, the task is to schedule the AGVs to meet the transportation requirements. In this application, the problem changes dynamically (see Refs. [8,10,17]). Therefore, the challenge is to adapt one's strategy in response to the changes.

The Network Simplex Algorithm (NSA) is an adaptation of the bounded variable of traditional primal simplex algorithm in Linear Programming [6], specifically for the MCF problem. 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. In this research, the scheduling problem is solved using Dynamic Network Simplex Algorithm (DNSA). The algorithm makes an initial solution for the new problem by exploiting the previous solution and some operations on the current problem.

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 a DNSA is to update efficiently the solution of a problem after dynamic changes, rather than having to resolve it from scratch-line each time.

* Tel.: +98 21 88725400.
*E-mail addresses:* hrashi@gmail.com, hrashi@atu.ac.ir

http://dx.doi.org/10.1016/j.asoc.2014.07.017
1568-4946/

Rauch classified dynamic graph problems according to the types of updates allowed [18]. A graph is said to be fully dynamic if the update operations include unrestricted insertions as well as deletions of arcs and nodes. A graph is called partially dynamic if only one type of update, either insertions or deletions, is allowed. If only insertions are allowed, the graph is called incremental; if only deletions are allowed it is called decremental. In this research the graph is fully dynamic.

The structure of the remaining parts of this paper is as follows: Next section is the related works. The section following Related works is a description of the MCF problem in container terminals. The fourth section presents Dynamic Network Simplex Algorithm and shows how the algorithm behaves in the dynamic environment. Experimental results and comparisons section presents the experimental results and calculates the complexity of the algorithm. In this section, the algorithms of NSA+ (see Ref. [5]) and DNSA+ are compared. The final section is considered for the summary and conclusion.

## The related works

The dynamic flows networks over time and their variations are very challengng problems. These types of problems are arising in various real applications such as communication networks, air/road traffic control, and production systems. Some examples and further applications of the problems are found in the references (see Refs. [28–32]). Below we survey the results most closely related to the dynamic network flows and problems.

Afshari and Taghizadeh consider a dynamic version of the maximum flow network in the simplest kinds of interdiction problem [36]. 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 budget 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, an algorithm based on the Benders' decomposition and another based on the algorithm of Ratliff et al. (1975) for the most vital arcs [38]. Although they consider a dynamic problem of the network flow model, the algorithms are not dynamic; i.e. without having any exploitations the current solution to respond to the dynamic changes.

Geranis et al. develop a new Dual Network Exterior-Point Simplex Algorithm (DNEPSA) for the Minimum Cost Network Flow Problem (MCNFP) [37]. 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 utilize the dynamic trees data structure in the DNEPSA algorithm, in order to achieve an improvement of the amortized complexity per pivot [35]. In extensive computational studies, DNEPSA performed better than the classical Dual Network Simplex Algorithm (DNSA). 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. [34] and Zheng and Chiu [33] 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. 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 [33] 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 Refs. [34] and [33] are too strict for diverge nodes, the algorithms may lead to suboptimal and are not appropriate for most dynamic problems.

Parpalea presents an approach for solving bi-criteria minimum cost dynamic flow problem with continuous flow variables [19]. 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 source node to the sink node in the time-space network avoiding the explicit time expansion of the network.

Based on the previous research, Parpalea and Ciurea 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) [22]. 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 objective 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 source node to the sink node in the time-space network while avoiding the explicit time expansion of the network. Parpalea and Ciurea 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 travelling time [20]. Their approach is very similar to the one used in Ref. [19].

Hosseini introduces another class of dynamic network flows in which the flow commodity is dynamically generated at source nodes and dynamically consumed at sink nodes [21]. As a basic assumption in this research, the source nodes produce the flow according to time generative functions and the sink 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 prespecified time horizon. To solve the problems, some simple and efficient approaches based on the minimum cost static flow models are developed.

Nasrabadi and Hashemi 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 [23]. 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 present a dynamic solution method for dynamic minimum flow networks [24]. The solution method solves the problem for a special parametric bipartite network [24]. Instead directly work on the original network, the method uses the

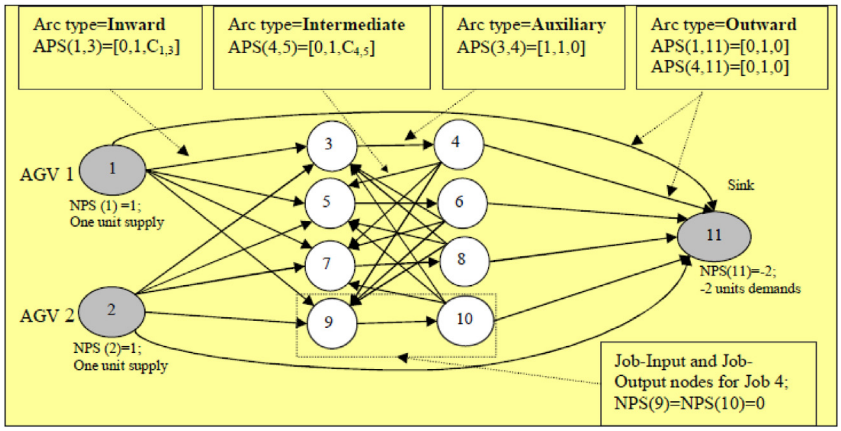


**Fig. 1.** An example of the MCF-AGV model of two AGVs and four container jobs.

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 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 [25]. 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.

Sherbeny propose a new version of the minimum cost flow problem on a time varying and time windows [26]. 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 source vertex to it's sink vertex satisfies a time window constraint with minimum cost and minimum waiting times at vertices, subject to the constraint that the flow must arrive at the sink vertex before a deadline. In this paper, the algorithm to be developed will search, successively, shortest paths from the source vertex s to the sink vertex in a dynamic residual network and then transmit as much as possible flow along the paths so that satisfies the time window constraint.

Fathabadi proposes a minimum flow problem on network flows in which the lower arc capacities in the graph model vary with time [27]. 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.

## Description of the MCF problem in container terminals

The problem, here, is the same as the problem defined in Ref. [11]. 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 Refs. [1,3,4,7,9,12,13,14]). The assumptions used are also the same as the assumptions in Ref. [11]. The MCF associated with the problem is presented as MCF-AGV model [5]. The MCF-AGV model was established on a directed graph. Fig. 1 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 Fig. 1), a couple of nodes for each container job (nodes 3–10 in Fig. 1) as transshipment nodes and a demand node (the node 11 in Fig. 1).

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 $i$ and the each AGV node to the SINK. The Auxiliary Arcs are directeds arc from every Job-Input node to its Job-Output node. For more details on the nodes and arcs refer to [5].

Suppose that for some values of the arc costs in the model, the solution paths are $1 \rightarrow 3 \rightarrow 4 \rightarrow 9 \rightarrow 10 \rightarrow 11$ and $2 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 11$. This states that AGV 1 is assigned to serve container jobs 1 and 4, and AGV 2 is assigned to serve container jobs 2 and 3 respectively. This solution is illustrated in Fig. 2.

## The Dynamic Network Simplex Algorithm

The problem defined in the previous section 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 (which results in distances between points being changed). In order to response to the changes, dynamic algorithms must be employed. The overall approach of Dynamic Network Simplex Algorithm (DNSA) is to update the graph model dynamically and repair its spanning tree by some strategies when any changes happen. A sample spannaing tree for the problem shown in Fig. 1 is illustrated in Fig. 3 (note that this spanning tree is obtained by adding an artificial root node '0' to the graph). Since the status of the nodes and arcs in the graph model have important roles in the dynamic algorithm, we explain them before focusing on the Dynamic Network Simplex Algorithm.

### *The nodes status in the graph model*

For each node the Node number, Predecessor, first Child, Right sibling (next Child of the Predecessor), Left sibling (previous Child of the Predecessor), Balance (amount of supply or demand of the node), Sub-tree's size, Basic arc of the node, Orientation of the Basic

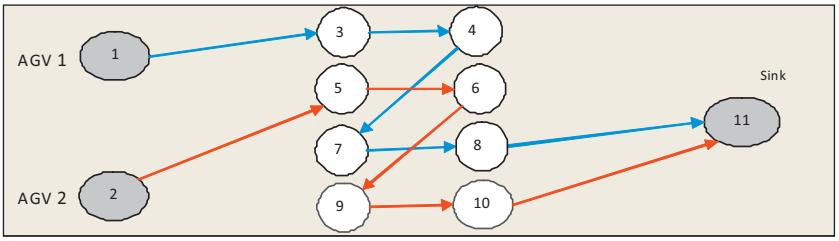


**Fig. 2.** A sample solution for the graph model shown in Fig. 1.

arc, Flow value of the Basic arc and Potential of the node are considered. These attributes are expalined below with an example.

Fig. 3 shows an example of the spanning tree for the graph model shown in Fig. 1 when the nodes 9 and 10 (the Job-Input and Job-Output nodes for Job 4) have been deleted. For this example, Table 1 shows the attributes of the spanning tree. Given a graph G = (N, A), let $T_t \subset A$ be a spanning tree in G at time $t$. The Root is identified with the node '0'. Consider some node $v \in N-\{0\}$:

- There is a unique (undirected) path, denoted by $P(v)$, from $v$ to the Root node '0'. The arc in $P(v)$, which is incident to $v$, is called the Basic arc of $v$.
- The Orientation of the Basic arc is called Upward/Downward if $v$ is the tail/head node of its Basic arc.
- The other terminal node $u$ of the Basic arc is called the Predecessor (node) of $v$. If $v$ is the Predecessor of some other node $u$, we call $u$ a Child (node) of $v$.
- The number of nodes in the sub-tree, rooted by $v$, including itself, is called the Sub-tree size of $v$.
- Every node may have a Right and/or Left sibling, but it has at most one Child reference. The other children of the node are accessible by traversing the Siblings.
- The Sub-tree's size and Predecessor variables are used to find a cycle and pivoting. The Orientation, Child, and Sibling variables are used for the computation of the node Potentials [15].
- The Predecessor, Child, Left sibling, Right sibling, Sub-tree's size, Basic arc of each node, and Orientation of the Basic arc are shown in Table 1.

For the status of the nodes, the following property is introduced:

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

- $FN_t$: The set of 'FIXED' nodes of the current graph model at time $t$.
- $DN_t$: The set of 'UNFIXED' nodes after repairing the solution at time $t$.

At each stage of the dynamic problem, a few existing jobs are fulfilled and a few new jobs are arrived. Based on the fulfilled jobs, a set of nodes for deletion is collected (it is called 'DELETION' nodes). The elements of this set are a couple of nodes associated with every fulfilled job (These nodes are called the Job-Input and Job-Output nodes; see Ref. [5]). The nodes of this set have to be removed from the graph model in the next stage. Additionally, when a new job arrives, a set of new nodes associated with the job are collected (it

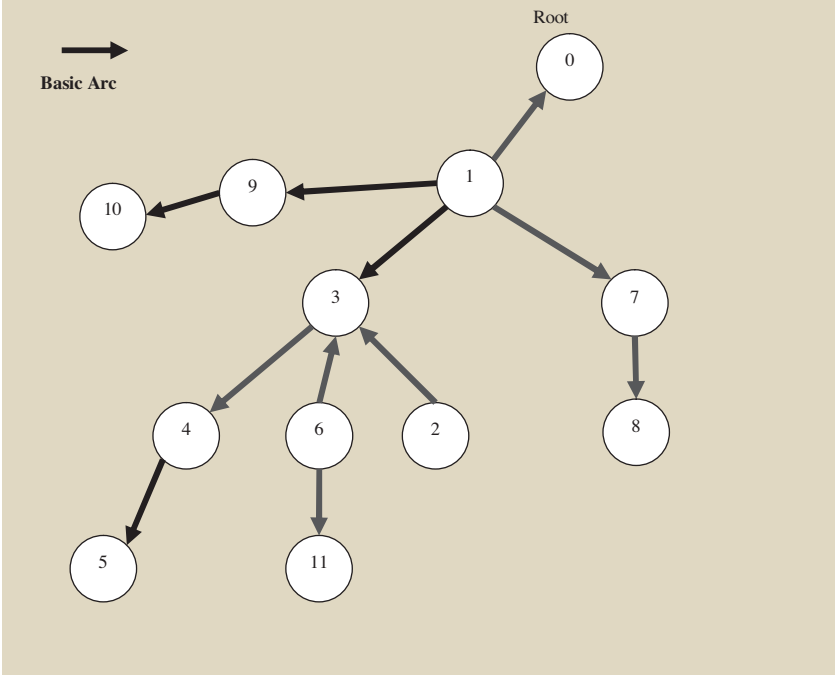


**Fig. 3.** A sample spanning tree for the graph model shown in Fig. 1.

**Table 1**
The attributes of the spanning tree depicted in Fig. 3 (F = 'FIXED', U = 'UNFIXED').

| Node | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Sub-tree' size | 9 | 8 | 1 | 6 | 2 | 1 | 2 | 2 | 1 | – | – | 1 |
| Predecessor | Null | 0 | 3 | 1 | 3 | 4 | 3 | 1 | 7 | – | – | 8 |
| Child | 1 | 3 | Null | 4 | 5 | Null | 11 | 8 | Null | – | – | Null |
| Right sibling | Null | Null | 7 | 7 | 6 | Null | 2 | Null | Null | – | – | Null |
| Left sibling | Null | Null | 6 | Null | Null | Null | 4 | 3 | Null | – | – | Null |
| Orientation | – | Up | Up | Down | Down | Down | Up | Down | Down | – | – | Down |
| Identification | F | F | F | F | F | F | F | F | F | D | D | F |

is called 'INSERTION' nodes). These nodes have to be inserted into the graph model in the next stage of the dynamic problem (Fig. 4).

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

*The Arcs in the graph model*

For the arcs in the MCF-AV model, including the Tail node, Head node, Lower bound, Upper bound, Cost and Value of the arcs. For the status of the arcs, another property is introduced below:

**Property 2.** Every arc has an Identification flag. The Identification of an arc specifies which set of the spanning tree structure the arc is in. There are four cases for the Identification of an arc at time $t$; the arc is either in the $T_t$ set, or the $L_t$ set, or the $U_t$ set (according to the spanning tree structure (T, L, U); see Ref. [6]) or in the $D_t$ set.

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

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

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

Note that the flow on every Basic arc in the spanning tree is between the Lower bound and the Upper bound of the arc. The flow of every arc in the set L is at the Lower bound of the arc. The flow of every arc in the set U is at the Upper bound of the arc. Moreover, the Artificial arcs connect the Root to the other nodes in the sets. These Artificial arcs were explained in Ref. [5].

*The Algorithms DNSA and DNSA+*

The Dynamic Network Simplex Algorithm is based on the Network Simplex Algorithm. DNSA is a standard dynamic form of NSA and DNSA+ is DNSA with the features of NSA+ (for more information on NSA+, refer to [5]). The inputs of the dynamic algorithms are:

- ***S***: it is Stage for the dynamic problem and is increased by the algorithm for each problem.
- **SODN**: a Set of 'DELETION' Nodes that determines which nodes have to be removed from the model.
- **SOIN**: a Set of 'INSERTION' Nodes that determines which nodes have to be put into the new model.

Fig. 5 shows the pseudo code of the Dynamic Network Simplex Algorithm. At the beginning of the algorithm when the software

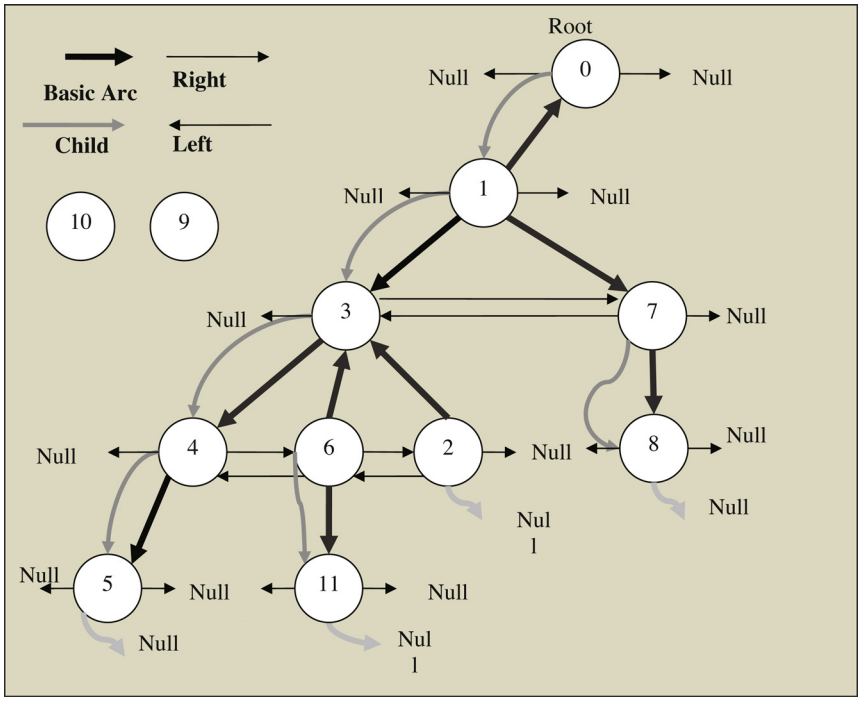


**Fig. 4.** A sample of the spanning tree for the model shown in Fig. 2 (after deleting the nodes 9 and 10).

```
Algorithm Dynamic Network Simplex Method (S:Stage, SODN: Set of DELETED Nodes,
                                          SOIN: Set of INSERTED Nodes);
Begin
  If ( !S) // S is zero
     Generate Initial BFS; // by (T0, L0, U0)
  Else
     Reconstruct New BFS ( SODN: Set of DELETED Nodes , SOIN: Set of INSERTED Nodes )
                         // by (Tt, Lt, Ut) - The main difference between NSA and DNSA
  End If
  (k, l) ← Entering Arc ∈ { Lt + Ut - DAt}
  While (k, l) ≠ Null Do
        Find Cycle W ∈ { Tt + (k, l) - DAt}
        θ ← Flow Change
        (p, q) ← Leaving Arc ∈ W
        Update Flow in W by θ
        Update BFS; Tree T
        Update node potentials
        (k, l) ← Entering Arc ∈ { Lt + Ut - DAt }
  End while
  s ← s + 1
End Algorithm
```

**Fig. 5.** The pseudo code of the Dynamic Network Simplex Algorithm.

made a MCF-AGV model, an initial feasible solution is generated by the procedure *Generate-Initial BFS*. The operation of this procedure was described in Ref. [6]. In fact, in this step an initial feasible spanning tree solution ($T_0, L_0, U_0$) is created. The difference between NSA and DNSA is the *Reconstruct New BFS*. When *s* 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. The main body of the algorithms, NSA and DNSA, are the same. The operation of the main body was described in Refs. [5,6].

Here, the procedure *Reconstruct New BFS* is descibed. Fig. 6 shows the pseudo code of this procedure. There are three main steps in the procedure; Step D1, Step D2 and Step D3. In Step D1, all 'DELETION' nodes and their arcs are removed from the model, the spanning tree of the graph and the solution paths. After that, the 'INSERTION' nodes and their arcs are put into the model, its spanning tree and solution in the second step (Step D2). In Step D3, according to the current solution a value is assigned to the potential of each node in the new spanning tree. There is no challenge in Step D3 since it is an easy task. The Steps D1 and D2 are elaborated by some examples as follows.

*Step D1: Removing the subgraph associated with the fullfilled jobs*

There is a loop for this step. At first, a couple of nodes associated with every fulfilled job (from the 'DELETION' set) are selected and transferred into the $D_t$ set. These two tasks are performed in Lines 4 and 5, respectively. Then, in Lines 6 and 7 a procedure, which is called *Remove-Node*, is used to remove the nodes from the spanning tree consistently. After that in Line 8, the fulfilled job associated with the nodes is removed from the solution paths. Based on removing the job from the solution, some arcs may be transferred into the set $L_t$ or $U_t$.

Fig. 7 shows the pseudo code of the procedure *Remove-Node*. Removing a node from the spanning tree, splits $T_t$ into several clusters, say $T_1$, $T_2$, ... and so on. Depending on whether the deleted node has a Child or not, there is a branch in the algorithm. Based on the location of the deleted node in the spanning tree, appropriate operations are done.

If the 'DELETION' node does not have any Child (Line 3) and its Predecessor does not have any other Child (Line 4), then the Child of the Predecessor is set to Null. After that in Lines 5 and 6, the Right and left siblings of other nodes are adjusted and the Sub-tree's size of the new spanning tree is updated. If the 'DELETION' node has a Child, the Right and left siblings of other nodes are adjusted in Line 8. In Lines 9 and 10, the last Child of the Root is found out and the sub-trees (Children of the 'DELETION' node) are connected to the root with the Artificial arcs. Then in Lines 11 and 12, the Basic arc of the root in the sub-trees, and their Predecessor as well as the sub-tree's size is adjusted. Some examples for a 'DELETION' node are demonstrated below:

```
1: Procedure Reconstruct New BFS (SODN: Set of DELETED Nodes, SOIN: Set of INSERTED Nodes)
2: Begin
     // Step D1: Removing the subgraph associated with the fullfilled jobs
3 : While (SODN is not empty) Do
4 :       Select a couple of nodes from SODN (the Job-Input and Job-Output Node).
5:        Put the set of associated arcs with the nodes in the set Dt.
6:        Remove-Node (the Job-Input node).
7:        Remove-Node (the Job-Output node).
8:        Remove the job from the solution path.
9:        Remove the nodes from SODN.
10: End While
      // Step D2 : Inserting the subgraph associated with the new Jobs
11: While (SOIN is not empty) Do
12:       Select a couple of nodes from SOIN (the Job-Input and Job-Output Node).
13:       Put the set of associated arcs with the nodes in the set Lt.
14:       Insert-Node (the Job-Input node).
15:       Insert-Node (the Job-Output node).
16:       Assign the node for a vehicle randomly
17:       Remove the node from SOIN.
18: End While
    / Step D3: Assign a value to the potential of each node
19: Assign node potentials for each node of the spanning tree.
20: End Procedure.
```

**Fig. 6.** The pseudo code of the procedure Reconstruct New BFS in the Dynamic Network Simplex Algorithm.

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

**Fig. 7.** The pseudo code of the procedure Remove-Node in the Reconstruct New BFS.

As the first example, suppose that the job associated with nodes 7 and 8 is fulfilled in Fig. 1. Imagine the node 8 must be deleted, first. In this case, the 'DELETION' node does not have any Child, Right sibling or Left sibling. In this case $T_1$ is the rooted spanning tree and $T_2$ is empty. What is necessary to do is to delete the Child of its Predecessor and then update the Sub-tree's size from the Predecessor of the deleted node to the Root. The procedure 'Remove-Node' must be executed for deleting nodes 7 and 8. This procedure updates the spanning tree and then the solution paths are $1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 11$ and $2 \rightarrow 11$. According to Property 1, the sets of nodes in the graph at time $t$ are:

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

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

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

The best and fastest way to recover the spanning tree is to connect the minor fragmented sub-trees to the Root. As the second example, suppose that the job associated with nodes 3 and 4 is fulfilled in Fig. 1. Imagine the node 3 is deleted first. In this case, the 'DELETION' node had a Child and its Right sibling exists. The following operations were necessary for this case:

- Adjust the Child of the node 1 to the node 7.
- Connect the sub-trees $T_1$ (the nodes 4 and 5), $T_2$ (the nodes 6 and 11) and $T_3$ (the node 2) to the Root.
- Adjust the Right and Left siblings from the most left side (the node 1) to the most right side (the node 2).
- Recalculate the Sub-tree size for the node 1 and Root.

Note that in the experience, the artificial arcs are used for reconnecting $T_1$, $T_2$ and $T_3$ to the Root or main part of the spanning tree. The Orientation of the Artificial-Basic arc depends on the amount of supply/demand of the node. For the node $j$, if $j$ is a Job-Output node the arc $(j, 0)$ is included in $T_t$. If $j$ is a Job-Input node, the arc $(0, j)$ is included in $T_t$. After deleting the nodes 3 and 4, the solution paths are $1 \rightarrow 5 \rightarrow 6 \rightarrow 11$ and $2 \rightarrow 7 \rightarrow 8 \rightarrow 11$. According to Property 1, the sets of nodes in the graph at time $t$ are:

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

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

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

*Step D2: Inserting the subgraph associated with the new Jobs*

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

Fig. 8 shows the pseudo code of the procedure *Insert-Node*. The input of the procedure is a node, which is appended to the new spanning tree by an Artificial arc. The attributes of these arcs is the same as the Artificial arcs in the Basic Feasible Solution. Firstly, the most Right sibling of the Root's Child is found and the new node is put at the right side of the existing Children of the Root. These operations are performed in Lines 3–5. Then in Line 6, the Basic-arc, Predecessor, Child, Right sibling, Left sibling and Sub-tree's size of this node are adjusted.

As an example, suppose that the nodes 9 and 10 have to be inserted into the spanning tree of Fig. 2 and the new job is inserted in the second path for AGV 2. After updating the tree, the solution paths are $1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 11$ and $2 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 11$. Now, the sets of nodes in the graph at time $t$ are:

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

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

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

## Experimental results and comparisons

To test and compare the performance of the algorithms, many jobs in dynamic fashion have been generated. Their sources,

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

**Fig. 8.** The pseudo code of the procedure Insert-Node in the Reconstruct New BFS.

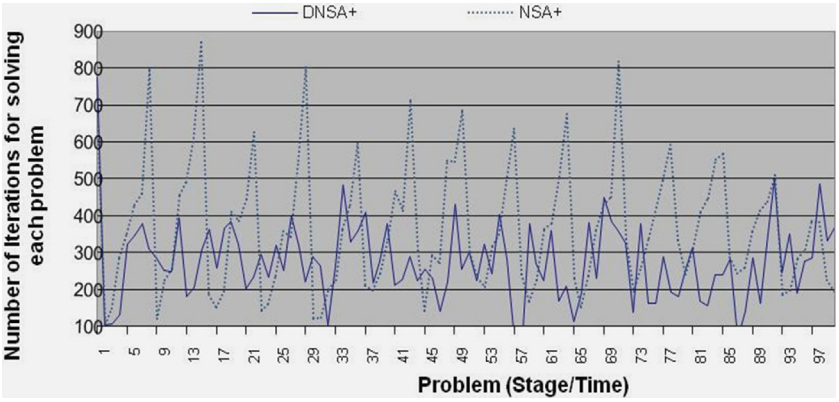


**Fig. 9.** A comparison of the number of iterations in running DNSA+ and NSA+ for the dynamic problems.

destinations and the distance between every two points in the port have been chosen randomly. While the time is being progressed, the vehicles and cranes are carrying and handling the containers. From time to time, the software makes a few random changes in the distance table between the source and destination of jobs in order to produce dynamic problems [15]. The Job Generator has to generate a few new jobs, when it finds out any crane is in idle state. During three hours simulation, one hundered problems with a condition of generating five jobs for any idle crane have been solved by DNSA+ and NSA+. We implemented the software in Borland C++ Builder, running on Pentium 2.4 Ghz. In these samples it was assumed that there were fifty AGVs and seven cranes in the port. Other experimental parameters are the same as in Ref. [5]. 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 requied to solve the problem is too much small and is not convenienet 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 Fig. 9. A sample was collected every time when there were changes in the problem and the algorithms had to solve the new problem.

From Fig. 9, it is observed that the number of iterations in DNSA+ has been greatly decreased compared with that of NSA+. Therefore, the average number of iterations in DNSA+ is less than NSA+ for the dynamic problem. For some experiments, NSA+ takes many more iterations than DNSA+. It is due to the problems (#8, #14, #29, #71, etc.) are some especial cases that the algorithms get much (less) benefit from the previous solutions and have less (more) works to solve the problems.

In these results for one hundered problems, there was about a 33 percent reduction in the number of iterations by DNSA+ compared with that of NSA+. The percentage of improvement, in reduction of the number of iterations, is calculated by the following terms and equation:

$NSA_i^+$: The number of iterations in $NSA^+$ for the dynamic problem at stage $i$.

$DNSA_i^+$: The number of iterations in $DNSA^+$ for the dynamic problem at stage $i$.

TPR: The Total Percentages of Reduction in the number of iterations in the experiment.

$$TPR = \frac{\sum_{i=1}^{100}(DNSA_i^+ - NSA_i^+)}{\sum_{i=1}^{100} DNSA_i^+} \times 100 = -33.03\%$$

Since the major process of the algorithms is performed in the body and the operations of the body are identical [15], 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 analyzed statistically. We tested the null hypothesis that the means produced by the two algorithms were statistically indifferent ($\alpha = 5\%$). Table 2 provides the test's result along with the values of T-distribution for a particular degree of freedom. Since we cared if the change (the difference between the two means) was positive or negative, 'One-tail' test was chosen. The Paired $T$-test determines the two means are significantly different at a 95 percent degree of confidence since the test's result is in the reject region.

The complexity of Network Simplex plus Algorithm was calculated in Ref. [5]. Here, it is shown that Network Simplex plus Algorithm and Dynamic Network Simplex plus Algorithm have the same complexity. Both the algorithms run the procedure BFS, which finds a Basic Feasible Solution at the beginning. The Dynamic Network Simplex plus Algorithm then calls the procedure Reconstruct New BFS to repair the spanning tree and current solution when $s$ (the input of the algorithm) becomes greater than zero. Given $n$ as

**Table 2**
The result of $T$-test after running the two algorithms, DNSA+ and NSA+.

| Statistical parameters | Values |
|---|---|
| Number of observations | 100 |
| T-test (paired two sample for means) | −4.76 |
| Degree of freedom | 99 |
| Critical $T$-value | −1.66 |

the number of nodes in the graph, it is easy to understand that the complexity of both BFS and Reconstruct new BFS are $n$ and $3n^2$, respectively. Based on the number of iterations and the complexity of each pivot (see Ref. [15]), the total complexity of this algorithm is determined as follows:

$$O(3n^2 + (m+n)mn^2C^2K\,\mathrm{Log}\,K)$$

Given $m$ as the number of arcs in the graph model, the following equations are obtained:

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

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

$$O(N^6)$$

## Summary and conclusion

In this research, the dynamic extensions of NSA and NSA+ were presented. These extensions are Dynamic Network Simplex Algorithm (DNSA) and Dynamic Network Simplex plus Algorithm (DNSA+). To evaluate the performance of the algorithms, the dynamic scheduling problem of AGVs in the container terminal (the problem defined in Ref. [11]) 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+), the important features of these algorithms as well as their advantages and disadvantages must be considered in practical situation. In dynamic problems, NSA and NSA+ start from scratch and reconsider the pre-established schedules. Our experience states that the 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 disadvantages that have to be paid in return for the performance. In order to determine to what extent these algorithms can be applied in practice, we are going to do more experiments in future research.

## References


[1] B.J. Wook, K.K. Hwan, A pooled dispatching strategy for automated guided vehicles in port container terminals, Int. J. Manage. Sci. 6 (2) (2000) 47–60.
[2] M.D. Grigoriadis, An efficient implementation of the network simplex method, Math. Program. Study 26 (1986) 83–111.
[3] M. Grunow, H.O. Günther, M. Lehmann, Dispatching multi-load AGVs in highly automated seaport container terminals, OR Spectrum 26 (2) (2004) 211–235.
[4] K.G. Murty, L. Jiyin, W. Yat-Wah, C. Zhang, C.L. Maria, J. Tsang, L. Richard, DSS (decision support system) for operations in a container terminal, Decis. Support Syst. 39 (2002) 309–332.
[5] H. Rashidi, E.P.K. Tsang, A complete and an incomplete algorithm for automated guided vehicle scheduling in container terminals, J. Comput. Math. Appl. 61 (2011) 630–641.
[6] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows: Theory, Algorithms and Applications, Prentice Hall, New Jersey, USA, 1993.
[7] Y. Huang, W.J. Hsu, Two Equivalent Integer Programming Models for Dispatching Vehicles at a Container Terminal. CAIS, Technical Report 639798, School of Computer Engineering, Nan yang Technological University, Singapore, 2002.
[8] H. Sen, Dynamic AGV-Container Job Deployment, Technical Report, HPCES Programme, Singapore-MIT Alliance, 2001.
[9] Y. Cheng, H. Sen, K. Natarajan, T. Ceo, K. Tan, Dispatching Automated Guided Vehicles in A Container Terminal, Technical Report, National University of Singapore, 2003.
[10] S.H. Chan, Dynamic AGV-Container Job Deployment (Master of science), University of Singapore, Singapore-MIT Alliance, 2001.
[11] H. Rashidi, E.P.K. Tsang, Applying the extended network simplex algorithm and a greedy search method to automated guided vehicle scheduling, in: The 2nd Multidisciplinary International Conference on Scheduling: Theory & Applications (MISTA), New York, vol. 2, 2005, pp. 677–693.
[12] J.M. Patrick, P.M. Wagelmans, Dynamic Scheduling of Handling Equipment at Automated Container Terminals, Technical Report EI 2001-33, Erasmus University of Rotterdam, Econometric Institute, 2001.
[13] J.M. Patrick, P.M. Wagelmans, Effective Algorithms for Integrated Scheduling of Handling Equipment at Automated Container Terminals, Technical Report EI 2001-19, Erasmus University of Rotterdam, Econometric Institute, 2001.
[14] J. Böse, T. Reiners, D. Steenken, S. Voß, Vehicle dispatching at seaport container terminals using evolutionary algorithms, in: Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, IEEE, 2000, pp. 1–10.
[15] H. Rashidi, Dynamic Scheduling of Automated Guided Vehicles in Container Terminals (Ph.D. thesis), Department of Computer Science, University of Essex, Colchester, UK, 2006.
[16] D.J. Kelly, G.M. ONeill, The Minimum Cost Flow Problem and The Network Simplex Solution Method (Master degree dissertation), University College, Dublin, 1993.
[17] C.Y. Leong, Simulation Study of Dynamic AGV-Container Job Deployment Scheme (Master of science), National University of Singapore, Singapore, 2001.
[18] M. Rauch, Fully Dynamic Graph Algorithms and Their Data Structures (Ph.D. thesis), Department of Computer Science, Princeton University, New Jersey, USA, 1992.
[19] M. Parpalea, A parametric approach to the bi-criteria minimum cost dynamic flow problem, Open J. Discrete Math. 1 (3) (2011) 116–126.
[20] M. Parpalea, E. Ciurea, Maximum flow of minimum bi-criteria cost in dynamic networks, Recent Res. Comput. Sci. (2011) 118–123.
[21] S.A. Hosseini, An introduction to dynamic generative networks: minimum cost flow, Appl. Math. Model. 35 (10) (2010) 5017–5025.
[22] M. Parpalea, E. Ciurea, The quickest maximum dynamic flow of minimum cost, J. Appl. Math. Inform. 5 (3) (2011) 266–274.
[23] E. Nasrabadi, S.M. Hashemi, Minimum cost time-varying network flow problems, Optim. Methods Softw. 25 (3) (2010) 429–447.
[24] E. Ciurea, M. Parpalea, Minimum flow in monotone parametric bipartite networks NAUN, Int. J. Comput. 4 (4) (2010) 124–135.
[25] M. Fonoberova, Algorithms for finding optimal flows in dynamic networks, in: S. Rebennack, P.M. Pardalos, M.V.F. Pereira, N.A. Iliadis (Eds.), Handbook of Power Systems. II. Energy Systems, Springer, Berlin, 2010, pp. 31–54.
[26] N.A. El-Sherbenym, A new class of a minimum cost flow problem on a time varying and time window, Sci. Res. Impact 1 (3) (2012) 18–28.
[27] H. Salehi Fathabadi, S. Khodayifar, M.A. Raayatpanah, Minimum flow problem on network flows with time-varying bounds, Appl. Math. Model. 36 (9) (2012) 4414–4421.
[28] J. Aronson, A survey of dynamic network flows, Ann. Oper. Res. 20 (1989) 1–66.
[29] M. Skutella, An introduction to network flows over time, in: Research Trends in Combinatorial Optimization, Springer, Berlin, 2009, pp. 451–482.
[30] W. Powell, P. Jaillet, A. Odoni, Stochastic and dynamic networks and routing Handbooks in Operations Research and Management Science, vol. 8, North-Holland, Amsterdam, The Netherlands, 1995, pp. 141–295 (Chapter 3).
[31] B. Hoppe, Efficient Dynamic Network Flow Algorithms (Ph.D. thesis), Cornell University, Ithaca, New York, 1995.
[32] M. Fonoberova, D. Lozovanu, Optimal dynamic flows in networks and applications, in: The International Symposium in the Issues of Calculation Optimization, Communications, Crimea, Ukraine, 2007, pp. 292–293.
[33] H. Zheng, Y. Chiu, A network flow algorithm for the cell-based single-destination system optimal dynamic traffic assignment problem, Transp. Sci. 45 (1) (2011) 121–137.
[34] W. Shen, Y. Nie, H.M. Zhang, A dynamic network simplex method for designing emergency evacuation plans, Transp. Res. Rec. 2022 (2007) 83–93.
[35] G. Geranis, Dynamic trees in exterior-point simplex-type algorithms for network flow problems, Electron. Notes Discrete Math. (2013), pp. 41, 93–100.
[36] M. Afshari Rad, H. Taghizadeh Kakhki, Maximum dynamic network flow interdiction problem: new formulation and solution procedures original research article, Comput. Ind. Eng. 65 (4) (2013) 531–536.
[37] G. Geranis, K. Paparrizos, A. Sifaleras, On a dual network exterior point simplex type algorithm and its computational behavior, Oper. Res. 46 (2012) 211–234.
[38] H.D. Ratliff, G.T. Sicilia, S.H. Lubore, Finding the n most vital links in flow networks, Management Science 21 (1975) 531–539.