Adaptations of the A* Algorithm for the Computation of Fastest Paths in Deterministic Discrete-Time Dynamic Networks

Ismail Chabini and Shan Lan

***Abstract*—This paper extends the A* methodology to shortest path problems in dynamic networks, in which arc travel times are time dependent. We present efficient adaptations of the A* algorithm for computing fastest (minimum travel time) paths from one origin node to one destination node, for one as well as multiple departure times at the origin node, in a class of dynamic networks the link travel times of which satisfy the first-in–first-out property. We summarize useful properties of dynamic networks and develop improved lower bounds on minimum travel times. These lower bounds are exploited in designing efficient adaptations of the A* algorithm to solve instances of the one-to-one dynamic fastest path problem. The developed algorithms are implemented and their computational performance is analyzed experimentally. The performance of the computer implementations of the adaptations of the A* algorithm are compared to a dynamic adaptation of Dijkstra's algorithm, stopped when the destination node is selected. Comparative computational results obtained demonstrate that the algorithms of this paper are efficient. Using a network containing 3000 nodes, 10 000 links, and 100 time intervals, the dynamic adaptations of the A* led to a savings ratio of 11, in terms of number of nodes selected, and to a savings ratio of five in terms of computation time. The effect of the network size on the performance of these adaptations is also studied. It is shown that the computational savings in term of both the number of nodes selected and the computation time, increase with the size of the network topology.**

***Index Terms*—A* algorithm, computer algorithms, dynamic networks, intelligent transportation systems, network optimization, time-dependent shortest paths.**

I. Introduction

THE computation of shortest paths in dynamic networks is at the heart of the computational needs of transportation applications involving networks equipped with information technologies. For instance, in the context of intelligent transportation systems (ITS) applications, the computation of shortest paths is a fundamental component in route guidance systems and in the development of solution algorithms for large-scale dynamic network flow models; such models are useful in supporting effective ITS decision-making. In such applications, there is usually a need to find a large number of shortest paths in networks the parameters of which are time-dependent. Furthermore, to meet the real-time operational requirement of ITS applications, the solution algorithms must be efficient enough in order to compute shortest paths in a running time faster than real-time.

Past research in the area of dynamic shortest path problems has mainly focused on the following two problem variants: the one origin-node to all destination-nodes shortest paths problem for a given departure time, and the all-nodes to one-destination-node shortest paths problem for all possible departure times (see [1]). In this paper, we examine the one origin-node to one destination-node problem variant, which is a topic that has been rarely studied in the literature. In this variant of the problem, one is interested in computing one-to-one shortest paths for a given departure time, or for many (typically all) possible departure times at the origin node.

Let us consider traffic networks where drivers seek a route that minimizes their travel time. To support travelers in selecting "optimal" routes, one can provide them with dynamic information in the form of shortest paths. The dissemination of this information can be done in various ways, for instance through devices installed in cars or through variable message signs installed at particular network nodes, such as major interchanges on a highway network. The computation of these shortest paths may be done on-board or in a traffic management and information center; note that the former approach is applicable only to vehicles equipped with on-board computing units.

In this paragraph and the next, we describe two route guidance situations where the need to compute one-to-one shortest paths in dynamic networks may arise. First, consider the following case of on-board route guidance provision and computation. A traffic management and information center forecast traffic conditions and distributes time-dependent link travel times to in-vehicle guidance processing units. These units compute fastest paths given a vehicle's current location in space and time (current node and current time), and the driver's destination node. This computational task is a one-to-one shortest path problem in a dynamic network. Note that, in general, on-board computation can be prohibitive in terms of in-vehicle storage memory and data transmission capacity, since a large amount of dynamic data, mainly consisting of time-dependent link travel times, needs to be transmitted to and stored in vehicles. These needs for fast communication speeds and in-vehicle memory space can, however, be met if one adopts methods that allow

Manuscript received June 1, 2000; revised October 17, 2001. This work was supported in part by the National Science Foundation under the CAREER Award Grant CMS-9733948 and in part by the U.S. Department of Transportation under Contract DTRS99-G-0001 and Contract DTRS95-G-0001. The Associate Editor for this paper was W. T. Scherer.

The authors are with the Department of Civil and Environmental Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139-4307 USA (e-mail: chabini@mit.edu; slan@mit.edu).

Publisher Item Identifier S 1524-9050(02)02702-3.

for a compact representation of dynamic data. The bit-stream method to represent dynamic network data described in [2], is an example of such method.

Now we describe a second situation where the need to compute one-to-one shortest paths in dynamic networks may arise. When routing information is disseminated via variable message signs installed at some nodes, such as major highway interchanges, one needs to compute shortest paths from these nodes to one or many major destination nodes. There are two possible ways to determine shortest paths for this application: find many one (guidance node)-to-one (destination node) shortest paths, or find many all-to-one (destination node) shortest paths for all departure times. The total number of nodes that are either a destination node or a node where information would be provided typically constitutes only a small fraction of the total number of network. For instance in a model of the highways and parkways network of New York's Lower Hudson Valley area, the total number of destination and route guidance candidate nodes is approximately 100, while the total number of nodes is in the order of several thousand nodes. Meeting the shortest path computational needs by solving many one-to-one shortest path problems, is therefore, in general, a more efficient approach than achieving the same objective by solving for each destination-node an all-to-one shortest paths problem for all possible departure times.

The A* algorithm, first proposed by [3], is an efficient algorithm to solve the one-to-one shortest path problem in static networks. It is well-known in the literature that the A* algorithm generally outperforms traditional static shortest path algorithms, to solve this variant of the static shortest path problem. The A* algorithm has been studied in the context of static networks only. In a commentary in [4], the question of whether the A* algorithm can be extended to solve one-to-one shortest path problems in dynamic networks was posed. To the best of our knowledge, the question has not been addressed in the literature, and the present paper contains first known results in this area.

Dynamic shortest path problems can be implicitly viewed as static shortest path problems in the time-space (or time-expanded) representation of a dynamic network (see [1]). This does not mean that one constructs a time-space network and apply a static shortest path algorithm. It rather means that one exploits the properties of time-space networks, as well as additional properties, to derive specialized algorithms for dynamic networks, without explicitly working on time-space networks. The main objective of this paper is to develop efficient adaptations of the A* algorithm by exploiting particular properties of dynamic network data. Thus, the paper extends the A* methodology to shortest path problems in dynamic networks.

There are various criteria to define the cost of a path. In this paper, the cost of a path is defined as its travel time. The term shortest path will then be interchangeably used to denote minimum-travel-time (or fastest) paths.

Consider the problem of computing fastest paths between an origin node and a destination node, for all possible departure times. A main algorithmic development of this paper is based on the following idea. Assume that one solves this problem as a series of one-to-one shortest path problems, each corresponding to a given departure time at the origin node. When computing a shortest path for a departure time t, one can exploit shortest path results that would have been obtained for earlier departure time instants (i.e., $t-1$). We take advantage of these past computational results by developing effective lower bounds on minimum travel times from a given node to a given destination node. As it is shown in the section on experimental results, this has led to meaningful improvements in the overall computational efficiency. This idea has some similarities with the idea of taking advantage of information from previous computations to speed up the search for an optimal path proposed in [5] in the context of static networks to solve one-to-one shortest path problems. The idea of this paper also has similarities with ideas behind the algorithm described in [6] to compute shortest paths from all nodes to one destination node in continuous-time dynamic networks, where shortest paths are determined in decreasing order of departure time at origin nodes, and the results for a departure time t are obtained using the results corresponding to later departure times to reduce the overall computation time.

The main contributions of this paper are summarized as follows. We have extended the A* methodology to shortest path problems to networks in which arc travels times are time-dependent, by presenting efficient adaptations of the A* algorithm computing fastest (or minimum-travel-time) paths from one origin node to one destination node, for one as well as for multiple departure times at the origin node. The adaptations are valid for a class of dynamic networks the link travel times of which satisfy the first-in–first-out (FIFO) property. We identified properties of dynamic data in such networks, and used them to develop improved lower bounds on minimum travel times. These lower bounds are exploited in the design of the efficient adaptations of the A* algorithm. The developed algorithms are implemented and their computational performance is analyzed experimentally. Numerical results show that the adaptations of the A* algorithm developed in this paper are more efficient than the dynamic adaptations of Dijkstra's algorithm stopped when the destination node is reached. The latter algorithm has been the fastest algorithm known in the literature to solve the one-to-one shortest path problem in dynamic networks. The computational results show that an adaptation of the A* algorithm is five times faster than the dynamic adaptation of Dijkstra's algorithm (stopped when the destination node is reached) for network containing 3000 nodes, 10 000 links, and 100 time intervals. Furthermore, computational experiments show that these computation-time savings increase with the network size. To the best of our knowledge, this is the first time that the A* algorithm has been studied in the literature in the context of dynamic network.

This paper is organized as follows. In Section II, a brief literature review on the A* algorithm and on dynamic shortest path problems is presented. In Section III, some definitions and notation are provided. Section IV describes the static A* algorithm. In Section V, some properties of dynamic networks are described. In Section VI, lower bounds on minimum travel times in dynamic networks are developed. Based on these lower bound results, adaptations of the A* algorithm to solve some dynamic shortest path problems efficiently are shown. Section VII presents results from an experimental study of computer imple-

mentations of the developed dynamic adaptations of the A* algorithm. Section VIII concludes this paper.

II. Literature Review

The A* algorithm was first proposed by [3] and further discussed and extended in [7], [8], and [9]. Hart *et al.* [3] pointed out that the A* algorithm is an admissible and optimal algorithm. Golden and Ball [10] empirically found that, on an infinite lattice network with diagonal arcs, the A* algorithm searches less than 8.3% of the area that would be searched by an optimal Label-Setting (LS) algorithm. Let n and m, respectively, denote the number of nodes and arcs in a network. Sedgewick and Vitter [11] proved that the A* algorithm finds a shortest path in many Euclidean graphs with an average computation effort in $O(n)$ compared to $O((m+n)*\log(n))$ required by a heap implementation of a LS algorithm. Bander and White [12] presented algorithm IA* (interruptible A*). This algorithm makes use of information about a collection of nodes, obtained from experts, which are likely to be on the optimal or near optimal path from the origin node to a destination node. Lark *et al.* [13] presented algorithm AG that uses a heuristic set, H, to guide the search. Such a set can represent natural language statements and bound information, such as Euclidean distance. Bander and White [5] presented the adaptive A* (AA*) algorithm, which generalizes both AG and IA*. This algorithm uses previously determined paths that are known to be optimal and all paths that experts have considered desirable (possibly optimal) to speed up the search. The idea of taking advantage of information from previous computations to speed up the search for an optimal path has some similarities with the basic ideas of the adaptations of the A* algorithm developed in this paper. Note that all the above developments of the A* algorithm have focused on static networks only. In this paper, we study dynamic networks and develop efficient specializations of the A* algorithm.

A first variant of dynamic shortest path problems was first proposed in [14], where the computation of shortest paths from all nodes to one destination node for all possible departure times was studied. The algorithm proposed in [14] can be viewed as an extended application of the Bellman–Ford algorithm to the time-expanded network, where the label of a node is a vector of scalar labels rather than a single scalar label as is the case in static networks. In [15], another algorithm is introduced to solve the same problem variant studied in [14]. This algorithm can be viewed as an extended application of a label correcting shortest path algorithm in the time-expanded network where the label of a node is a vector of scalar labels rather than a single scalar label as is the case in static networks. Both algorithms in [14] and [15], were proposed for the minimum-time path problem. In [1], it is shown that these algorithms both have a worst-case running time complexity of $O((M+n)(m+n)M)$, where M is the number of discrete-time intervals in the dynamic network.

Instead of constructing the vector of shortest path labels by iterating on nodes as is done in previous algorithms, Chabini [16], [17] observes that the vector of optimal shortest path costs can be constructed in decreasing order of time, where at a time step t the optimal labels for all nodes that correspond to departure time t are determined. This resulted in an algorithm known as algorithm DOT, which has a running time complexity of $\theta(nM + mM + SSP(n, m))$, where $SSP(n, m)$ is the running time of an all-to-one shortest path algorithm. In [16], [17], it is shown that algorithm DOT has an optimal running time complexity is the sense that no algorithm with a better worst-case running time complexity can be developed to solve the all-to-one dynamic shortest path algorithm for all possible departure times.

Another most studied variant of dynamic shortest path problems is the computation of shortest paths from one node to all other nodes for a given departure time. One of the celebrated results for this problem is: when the FIFO property (see Section III for the FIFO definition) is satisfied, any static shortest path algorithm can be generalized to solve the one-to-all dynamic shortest path problem, for a given departure time with the same time complexity as the static one-to-all shortest paths problem. Dreyfus [18] was the first to suggest this generalization heuristically. Later, Ahn and Shin [19] and Kaufman and Smith [4] proved that this generalization is valid only if the FIFO property holds. In [16], [17] another shorter proof of this result is given, which is provided later in this paper.

Chabini and Dean [1] extended the results established in [16] and [17]. They present a complete framework for classifying, formulating, and solving different variants of shortest path problems in discrete-time dynamic networks. The framework includes all problem variants previously studied in the literature as special cases. Extending algorithm DOT described in [16] and [17], Chabini and Dean [1] designed easy to implement algorithms that have optimal theoretical worst-case running time complexities. Chabini and Yadappanavar [2], [20] suggest a method to represent dynamic data in a compact form known as bit-streams, and developed algorithms that efficiently operate on this compact form of data representation.

The above literature review shows that most of the research developments in the area of dynamic shortest paths have mainly focused on one-to-all or all-to-one shortest path problems. Few efforts have been given to the one-to-one dynamic shortest path problem. Kaufman and Smith [4] suggest using static shortest path algorithm such as LS algorithms to solve the one-to-one shortest path problems for one departure time. In a commentary to the paper by Kaufman and Smith [4], Koutsopoulos posed the question of whether the A* algorithm can be extended to solve one-to-one shortest path problems in dynamic networks. The present paper reports on the first developments in this direction. It shows efficient adaptations of the A* algorithm to solve the one-to-one fastest path problem in dynamic networks.

III. Definitions and Notation

Let $G = (N, A, D)$ be a directed network, where $N = \{1, \ldots, n\}$ is the set of nodes, and $A = \{1, \ldots, m\}$ is the set of directed links. We denote by $D = \{d_{ij}(t) | (i, j) \in A\}$ the set of time-dependent link travel times. Functions $d_{ij}(t)$ have integer-valued domain and range. A function $d_{ij}(t)$ is then a discrete and time-dependent function which is assumed to take a static value after a finite number of intervals M. $T =$

$\{0, \ldots, M-1\}$ is, hence, the set of departure time intervals for which link travel times are time-dependent. $B(i)$ denotes the set of nodes having an outgoing link to node i and $A(i)$ denotes the set of nodes that are the end of a link outgoing from node i. Let node $o \in N$ denote an origin node, and node $q \in N$ denote a destination node.

The notation in this paragraph assumes a static network. Following are notation that we use in describing the A* algorithm (a more common notation can be found in [7] which gives a generic description of the A* algorithm that is valid for a variety of artificial intelligence applications):

- L_i minimum travel cost from origin node o to node i;
- e_{ij} minimum travel cost from node i to node j;
- F_i minimum travel cost among all paths from origin node o to destination node q constrained to go through node i;
- $\hat{L}_i$ upper bound on the minimum travel cost from origin node o to node i;
- $\hat{e}_{ij}$ lower bound on the minimum travel cost from node i to destination node j;
- $\hat{F}_i$ estimate of F_i; $\hat{F}_i = \hat{L}_i + \hat{e}_{iq}$;
- C set of nodes that have been reached and that are candidates for the selection of the next node;
- S set of nodes that have been selected, and that are not in set *C*. A node may be removed from set *S* and added to set *C* if its label $\hat{F}_i$ decreases; this can happen only if the consistency assumption, defined next, is not valid.

Consistency Assumption: The notion of consistency in the area of shortest path algorithms has been used to denote different concepts. In this paper, for a given arbitrary destination node q, we say that the consistency assumption is valid if and only if, for any pair of nodes i and j, the lower bounds on the minimum travel costs (times) from these nodes to a given destination satisfy the following inequality: $e_{ij} \geq \hat{e}_{iq} - \hat{e}_{jq}$. That is, the difference of the lower bounds for any pair of nodes i and j, is a lower bound on the minimum travel time from i to j. Some researchers in the transportation field (see [4]) also used the term consistency to refer to the FIFO property, which we define later in this section. Note that if the former definition of consistency assumption is used, we have $e_{ij} \geq \hat{e}_{iq} - \hat{e}_{jq}$. This implies that for every link $(i, j) \in A$, the (reduced) cost $d^r_{ij} = d_{ij} + \hat{e}_{jq} - \hat{e}_{iq}$ is nonnegative, as $d_{ij} \geq e_{ij}$. The relationship between the consistency assumption and the positivity of the links' reduced costs, will be used to alternatively interpret the A* algorithm as a classical LS algorithm if the consistency assumption is valid. This aspect is explained further later in this paper.

FIFO Definitions: In a traffic network, at an aggregate level, link travel times are usually such that travelers arrive at the end of a link in the same order in which they depart the beginning of the link. This is known as the FIFO property. More formally, we say that a link travel time function $d_{ij}(t)$ satisfies the FIFO property if the arrival time function $t + d_{ij}(t)$ is nondecreasing. In a discrete-time dynamic network, a link $(i, j) \in A$ is a FIFO link if and only if

$$t + d_{ij}(t) \leq t + 1 + d_{ij}(t+1) \qquad \forall t \in \{0, \ldots, M-1\}.$$

If the travel time function of a link $(i, j) \in A$ satisfies the FIFO property, we say that link (i, j) is a FIFO link. Similarly, if the travel time function of a path satisfies the FIFO property, we say the path is a FIFO path. If every link in a network is a FIFO link, then the network is said to be a FIFO network. Minimum-time dynamic shortest path problems are easier to solve in FIFO networks, than more general dynamic shortest path problems including minimum-time path problems in non-FIFO networks (which are defined as networks with at least one non-FIFO arc).

The problem considered in this paper is to find fastest paths from an origin node to a destination node for one, or for multiple, departure times in a FIFO dynamic network. To solve this problem, one can use various dynamic shortest path algorithms that were developed in the literature. For instance, one can adopt any LS one-to-all shortest path algorithm in a FIFO network [4], and stop the search when the label of the destination node is selected to be permanently set. These shortest path algorithms may be improved, however, since some nodes may unnecessarily be searched. The adaptations of the A* algorithm of this paper, aim at avoiding searching nodes that would not be on a shortest path. A mechanism to achieve this is based on the following observation: in a FIFO network, when one computes the shortest path at time interval t, one can take advantage of the information obtained when computing a shortest path for time interval $t-1$. This information is used to develop improved lower bounds on the fastest travel time from a node i to destination node q.

IV. A* Algorithm

We believe that typical readers of this journal are from the transportation area, and, hence, may not be familiar with the details of the A* algorithm. Thus, before developing the adaptations of the A* algorithm, we first provide a relatively brief introduction to the A* algorithm. We first illustrate the basic idea of the A* algorithm, and then describe it. Some properties of the A* algorithm are also summarized. Finally, we show that if the consistency assumption is valid, the A* algorithm can be viewed as Dijkstra's algorithm applied to an equivalent network. This network has the same topology as the original network and has as link costs the reduced-costs derived using the original link costs and the lower bounds that verify the consistency assumption.

A. Basic Idea of the A* Algorithm

To solve a one-to-one static shortest path problem, one can use any traditional LS algorithm designed for one-to-all shortest path problems and stop it when the destination node is reached. These algorithms are, however, not the most efficient algorithms possible to solve a one-to-one shortest path problem. For example, consider a city network where the origin node is located at the center of the city and the destination node is located at its far east. An LS algorithm would typically put the same effort to search to the east, to the south, to the west and to the north of the origin node. These algorithms may, hence, search areas through which a shortest path would never pass.

Since one knows the objective of the search which is to reach the destination node, intuitively, the efficiency of a LS algorithm may be improved if one takes advantage of this information to guide the search. For instance, the shortest path search may be constrained within a certain subarea of the whole network. The

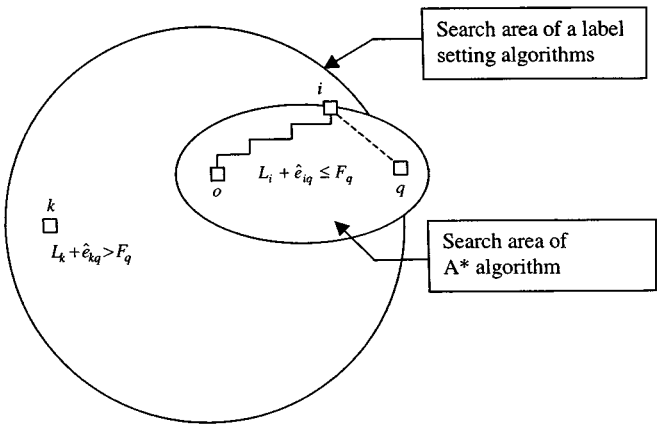

Fig. 1. Search area of A* algorithm and a LS algorithm.

resulting search area would then be much smaller than the area examined by a traditional LS algorithm.

Let us consider a traffic network where nodes are uniformly distributed in a given geographic area, and link travel distances are Euclidean distances. Assume that one needs to compute a shortest path from an origin node o to a destination node q (see Fig. 1). If Dijkstra's algorithm is used, at each new loop [in Step 2)], it selects a node that is closest to the origin node, i.e., a node with minimum $\hat{L}_i$. The resulting search area of Dijkstra's algorithm is roughly a circle. The A* algorithm orders the nodes in the candidate node set according to labels $\hat{F}_i = \hat{L}_i + \hat{c}_{iq}$, which is an estimate of the minimum travel cost among all paths from an origin node to destination node q constrained to go through node i. The A* algorithm selects a node with minimum label $\hat{F}_i = \hat{L}_i + \hat{c}_{iq}$. It will be proven in Corollary 1 of Propositions 1 and 2 that the A* algorithm selects only those nodes that satisfy $L_i + \hat{c}_{iq} \le F_q = L_q$. If the lower bound used in the A* algorithm is the Euclidean distance, the search area corresponding to the latter inequality will then be an ellipse [10]. As shown in Fig. 1, the search area of the A* algorithm would then be much smaller than the search area of a traditional LS algorithm.

B. Description of the A* Algorithm

Following is a description of the A* algorithm. The (origin, destination) node pair is denoted (o, q).

Step 1) Initialization:
Set $i = o$, $\hat{L}_i = 0$, $\hat{F}_i = \hat{c}_{iq}$; $\hat{L}_j = \infty$, $\hat{F}_j = \infty$, $\forall j \neq i$; $C = \{i\}$; $S = \phi$.

Step 2) Node selection:
Choose $i \in \text{Arg}\min_{j \in C} \hat{F}_j$, $S = S \cup \{i\}$, $C = C \backslash \{i\}$.

Step 3) Stopping rule:
If $i = q$, then stop. Otherwise, continue.

Step 4) Update $\hat{F}_j$ and distance labels $\hat{L}_j$:
For each $j \in A(i)$: If $\hat{L}_i + d_{ij} + \hat{c}_{jq} < \hat{F}_j$ then $\hat{L}_j = \hat{L}_i + d_{ij}$; $\hat{F}_j = \hat{L}_i + d_{ij} + \hat{c}_{jq}$;
If $j \notin C$, $C = C \cup \{j\}$. Go back to Step 2).

In Step 4) of the algorithm, labels $\hat{F}_j$ and $\hat{L}_j$ are updated for every $j \in A(i)$. If the consistency assumption is satisfied by the lower bounds, one does not need to reconsider those nodes j belonging to the selected node set (S) (see Proposition 1). Also note that in an implementation of the algorithm, for a node j, one can keep only label $\hat{F}_j$ instead of both labels $\hat{L}_j$ and $\hat{F}_j$; since $\hat{F}_j$ can be obtained as a function of $\hat{F}_i$ only ($\hat{F}_j = \hat{L}_i + d_{ij} + \hat{c}_{jq} = \hat{L}_i + \hat{c}_{iq} + d_{ij} + \hat{c}_{jq} - \hat{c}_{iq} = \hat{F}_i + d_{ij} + \hat{c}_{jq} - \hat{c}_{iq}$). This would lead to some savings in computation time and memory space. The term $d_{ij} + \hat{c}_{jq} - \hat{c}_{iq}$ is known as the reduced cost (in this paper travel costs are the travel times).

Descriptions of Dijkstra's shortest path algorithm can be found in classical books on network algorithms. This algorithm can be viewed as a particular case of the A* algorithm, where the lower bounds are equal to zero. If link costs are nonnegative, the zero lower bounds verify the consistency condition, and, thus, there is no need to reconsider nodes that have been selected in Step 2). We then omit providing a separate description of the one-to-one Dijkstra's shortest path algorithm, as it can be easily derived from the description of the A* algorithm.

C. Some Properties of the A* Algorithm

In this section, we present some properties of the A* algorithm. These properties are useful for understanding the A* algorithm and for developing the adaptations of the A* algorithm of this paper. Since these properties appear in the literature, we state them without proofs.

Proposition 1 [3, Lemma 2]: Suppose that the consistency assumption is satisfied. If node i is selected by the A* algorithm, then $\hat{L}_i = L_i$.

If the consistency assumption is not satisfied, then it is possible that the $\hat{F}_i$ label of a node i selected in earlier steps of the A* algorithm will be lowered in later steps of the algorithm. Hence, if the $\hat{F}_i$ label of a node i is lowered, then this node needs to be put back to the candidate set C. Proposition 1 however implies that, if the consistency assumption is satisfied, as soon as a node is selected by the A* algorithm, a shortest path from the origin node to that node has been identified. Therefore, nodes that have been selected will not re-enter the candidate set. The last statement in the A* algorithm need not be performed for those nodes that are already in the set of selected nodes S.

Proposition 2 [3, Corollary of Lemma 3]: For any node i selected by the A* algorithm, $\hat{F}_i = \hat{L}_i + \hat{c}_{iq} \le F_q$.

Proposition 3 [3, Theorem 1]: When the A* algorithm terminates, it always finds a shortest path from the origin node to the destination node.

Proposition 3 shows the correctness of the A* algorithm.

Proposition 4 [3, Theorem 2]: Consider the set of lower bounds verifying the consistency assumption. If a node is selected by the A* algorithm for a given a lower bound, then this node will be selected by the A* algorithm using any smaller lower bound.

The nodes selected in Step 2) of the A* algorithm, determine the arcs that will be explored in Step 4). Assuming that the effort to compute lower bounds are similar and that the number of arcs leaving a node is evenly distributed among nodes, the overall computational effort is an increasing function of the overall number of nodes selected in Step 2) of the A* algorithm. Proposition 4 implies that if the lower bounds satisfy the consistency assumption, then the total number of nodes selected by, and consequently the overall computational effort of, the A* algorithm is a nonincreasing function of the values of $\hat{c}_{id}$.

The next two corollaries further show the relationship between the number of nodes selected by the A* algorithm and the quality of a lower bound on e_{iq}.

Corollary 1: If the consistency assumption is satisfied, then the A* algorithm selects only those nodes with $L_i + \hat{e}_{iq} \leq F_q$.

Proof: Since the consistency assumption is satisfied, then using Proposition 1 we have, $\hat{L}_i = L_i$. Using Proposition 2, if node i is selected by the A* algorithm, then $\hat{L}_i + \hat{e}_{iq} \leq F_q$. Thus, we have $L_i + \hat{e}_{iq} \leq F_q$. ♦

Corollary 2: If $\hat{e}_{iq} = e_{iq}$ for every node i, then the A* algorithm selects only those nodes on a shortest path.

Proof: First, we prove that, if $\hat{e}_{iq} = e_{iq}$ for every node i, then $\hat{e}_{iq}$ satisfies the consistency assumption. Since e_{iq} is the minimum travel time from i to q, for any two nodes i and j we have $e_{iq} \leq e_{ij} + e_{jq}$. Hence, we have $e_{ij} \geq e_{iq} - e_{jq}$.

Since the consistency assumption is satisfied, then, by Corollary 1, the A* algorithm selects only those nodes with $L_i + \hat{e}_{iq} \leq F_q$. Since $\hat{e}_{iq} = e_{iq}$, the A* algorithm selects only those nodes that verify the inequality $L_i + e_{iq} \leq F_q$. Since we know that any node i verifies $L_i + e_{iq} \geq F_q$, if $\hat{e}_{iq} = e_{iq}$ for all nodes, the A* algorithm selects only those nodes that verify the equality $L_i + e_{iq} = F_q$. This means that any selected node i is on a shortest path from the origin node the destination node. Therefore, if $\hat{e}_{iq} = e_{iq}$ for every node i, the A* algorithm selects only those nodes on a shortest path. ♦

In summary, the efficiency of the A* algorithm depends on the quality of the lower bound on e_{iq}. If $\hat{e}_{iq} = 0$ for every node i, then $\hat{F}_i = \hat{L}_i$. This means that the A* algorithm is identical to Dijkstra's algorithm. If $\hat{e}_{iq} = e_{iq}$, the A* algorithm selects only those nodes on a shortest path. If $0 < \hat{e}_{iq} < e_{iq}$, the number of nodes selected by the A* algorithm is a nonincreasing function of $\hat{e}_{iq}$, if the consistency assumption is valid. Therefore, in order to make the algorithm select fewer nodes, one needs to find a tighter lower bound on e_{iq}.

D. Reduced Cost Presentation of A* Algorithm

The A* algorithm is generally stated as shown in Section IV-B. We now show that the A* algorithm is identical to Dijkstra's shortest path algorithm, if the original link travel costs are replaced by the reduced link costs obtained using the lower bounds satisfying the consistency assumption. For all $(i, j) \in A$, the expression of the reduced cost is $d^r_{ij} = d_{ij} + \hat{e}_{jq} - \hat{e}_{iq}$. The following lemma establishes a relationship between A* algorithm and Dijkstra's algorithm. It is assumed that Dijkstra's algorithm is stopped as soon as the destination node is selected.

Lemma 1: If the consistency assumption is satisfied, then applying the A* algorithm in a network with link travel times d_{ij} to solve a one-to-one shortest path problem is the same algorithm (however stated "differently") as applying Dijkstra's algorithm in the same network with link costs d^r_{ij}.

Proof: If the consistency assumption is satisfied, for all $(i, j) \in A$, we have $d_{ij} \geq e_{ij} \geq \hat{e}_{iq} - \hat{e}_{jq}$. Therefore, $d^r_{ij} = d_{ij} + \hat{e}_{jq} - \hat{e}_{iq}$ is nonnegative. Hence, we can apply Dijkstra's algorithm in the network with link travel times d^r_{ij} to solve a one-to-one shortest path problem. Note that Dijkstra's algorithm is stopped when the destination node is selected.

Except at their last step, the A* algorithm and Dijkstra's algorithm follow the same steps. Since the reduced costs are nonnegative, the label of a node selected in earlier stages of these algorithms will not be decreased in later stages. The statements in the last step of the A* algorithm are then identical to the last step of Dijkstra's algorithm which considers only the nodes that have been reached but have never been selected. ♦

The above equivalency in the statements of the algorithms could also serve as a basis for yet another proof of the validity of the A* algorithm if the consistency assumption is valid. Since replacing d_{ij} by d^r_{ij} or replacing d^r_{ij} by d_{ij} does not affect the shortest paths between any pair of nodes (see [21]), a shortest path found by Dijkstra's algorithm in the network with link travel times d^r_{ij} is also a shortest path found in the original network, and vice versa. Therefore, applying the A* algorithm in a network with link travel times d_{ij} to solve a one-to-one shortest path problem is a valid algorithm and is the same as applying Dijkstra's algorithm in a network with the same topology but with link travel times d^r_{ij} instead of d_{ij}.

In the case when the consistency assumption is valid, Lemma 1 also implies that one does not necessarily need to write a separate computer code for the A* algorithm if one already has a computer code of Dijkstra's algorithm. One can run Dijkstra's algorithm on a network by replacing link travel times d_{ij} by d^r_{ij}. Note, however, that replacing d_{ij} by d^r_{ij} for all links will induce extra computational time. In fact, to solve the one-to-one shortest path problem, one does not need to search all the links in a network. Thus, it is not necessary to replace d_{ij} by d^r_{ij} for all links prior to applying Dijkstra's algorithm. Instead, one needs to do so only when one needs to access an arc. This observation is of particular interest in the context of dynamic networks for the following reason. As it is shown later in this paper, the time-expanded network contains mM arcs, at most m arcs of which will be searched by a dynamic adaptation of LS algorithm. Since the LS algorithm is stopped when the destination node is selected, an even smaller number of arcs will be searched to solve the one-to-one fastest path problem in a dynamic network. Hence, one should compute d^r_{ij} for only those links actually searched by the algorithm. This means that if computational efficiency is sought, one needs to change a computer code of Dijkstra's algorithm in such a way that a link reduced cost is computed only when a link is accessed to update nodes not yet selected.

A* algorithm can be viewed as an algorithm that perturbs link costs, by adding $\hat{e}_{jq} - \hat{e}_{iq}$ to the cost of each arc (i, j). Thus, if $\hat{e}_{jq} < \hat{e}_{iq}$, the perturbation increases the cost of arc (i, j), while it increases the cost of arc (i, j) if $\hat{e}_{jq} > \hat{e}_{iq}$. If the node lower bounds are a measure of closeness to the destination node, the arcs that are pointing closer to the destination have their costs decrease, while the nodes pointing away from the destination have their costs increase. This leads to possible re-ordering in the selection of nodes in Step 2). Hence, nodes might not be selected, or even not reached before the destination node is selected.

V. Some Properties of Dynamic Networks

This section presents some fundamental results and properties of dynamic networks upon which the adaptations of the A* algorithm of this paper are based. First, we show that dynamic networks can be viewed as static networks by using the

time-expanded network representation. Then some properties of time-expanded networks are highlighted. Finally, we present some FIFO properties that will be used to develop lower bounds on minimum travel times for the dynamic adaptations of the A* algorithm.

A. Time-Expanded Network

A discrete time dynamic network can be represented as a static network using a time-expanded network representation, which is a useful implicit tool for visualizing, formulating and solving discrete time dynamic shortest path problems. This network is formed by expanding the original dynamic network in the time dimension, and making a separate copy of all nodes for every integer value of time $t \in \{0, 1, 2, \ldots, M-1\}$. Every node in the time-expanded network represents a time-node pair consisting of a time $t \in \{0, 1, 2, \ldots, M-1\}$ and a node $i \in N$, where the nodes at the highest level of time are taken to represent not only time interval $M-1$, but all times greater than or equal to $M-1$. Every link in a time-expanded network is a directed link from a node-time pair (i, t) to another node-time pair $[j, \min\{T, t+d_{ij}(t)\}$, where $j \in A(i)]$.

Time-expanded networks have the following properties.

1) Along the time dimension, they are acyclic if arc travel times are positive, and multileveled if arc travel times are nonnegative.
2) Every path on the original dynamic network corresponds to a path on the time-expanded network with the same travel time and travel cost. Visiting a node i in the original dynamic network at time t corresponds to visiting node-time pair (i, t) in the corresponding time-expanded network.
3) A shortest path problem in a dynamic network can be solved by applying a static shortest path algorithm to its equivalent representation as time-expanded network.

A consequence of properties 2 and 3 above is that dynamic shortest path problems can be solved by (implicitly) applying static shortest path algorithms to the time-expanded representation of a dynamic network. This observation applies to the A* algorithm as well. Since the time-expanded network contains nM nodes and mM arcs, a trivial and direct application of a static shortest path algorithm may, however, not be the most efficient algorithm possible. Property 1 is exploited in [1] to formulate and efficiently solve a variety of discrete time dynamic shortest path problems in a common framework.

For each departure time, the adaptations of the A* algorithm presented in this paper search a subset of arcs less (practically much less) than m links among mM links, and select a subset of nodes less (practically much less) than n nodes among nM nodes. The main task that we address in the remainder of this paper is to show ways aimed at visiting a number of arcs, and nodes, as small as possible in practical implementations of the A* algorithm for the one-to-one dynamic shortest path problem.

B. FIFO Properties

Link travel times may possess some properties useful in studying and developing efficient algorithms for dynamic networks, such as, the FIFO property. Below, we describe some interesting consequences of the FIFO property, which are very useful in developing efficient adaptations of the A* algorithm.

Lemma 2: If every link on a path is a FIFO link, then the path is a FIFO path.

Proof: We prove this lemma by induction. Assume that the path has k links, and that the departure time index at the first node in this path is t. To simplify the presentation, without loss of generality, we assume that the indices of the k links are: 1, 2, ..., k.

The induction hypothesis is that the arrival time function at the end of the lth link, $a_l(a_{l-1}(a_{l-2}\cdots a_1(t)))$, is a nondecreasing function of t. First let us consider the base case. The first link on the path is a FIFO path, and, hence, the induction hypothesis is valid. Now, suppose that the hypothesis is true for the lth link. We need to prove that, for the $(l+1)$th link, the arrival time function $a_{l+1}(a_l(a_{l-1}\cdots a_1(t)))$ is a nondecreasing function of t. From the induction hypothesis, we know that $g(t) = a_l(a_{l-1}(a_{l-2}\cdots a_1(t)))$ is a nondecreasing function of t. Since every link on this path is a FIFO link, according to the definition of FIFO link, $f(t) = a_{l+1}(t)$ is a nondecreasing function. Let $h(t) = f(g(t))$. Let us prove that function $h(t)$ is a nondecreasing function of t. Since $f(\cdot)$ and $g(\cdot)$ are two nondecreasing functions, we have: $t \le t' \Rightarrow g(t) \le g(t')$ and $y \le y' \Rightarrow f(y) \le f(y')$. Let $y = g(t)$ and $y' = g(t')$, it follows that $f(g(t)) \le f(g(t'))$. Thus, $h(t) = f(g(t)) \le h(t') = f(g(t'))$. Hence, we have $t \le t' \Rightarrow h(t) \le h(t')$. Therefore, $h = f \circ g$ is a nondecreasing function. As $h(t) = a_{l+1}(a_l(a_{l-1}\cdots a_1(t)))$, the arrival time function of the path is a nondecreasing function of t, and the path is a FIFO path. ♦

Corollary: In a FIFO network, any path satisfies the FIFO property.

Proof: In a FIFO network, every link satisfies the FIFO property. Thus, all links of any path satisfy the FIFO property. Using lemma 2, any path satisfies the FIFO property. ♦

Lemma 3: If every path between origin node o and destination node q satisfies the FIFO property, then the minimum travel time function satisfies the FIFO property.

Proof: Suppose that p_t is a shortest path among all paths from origin node o to destination node q and departing node o at time t. The travel time of path p_t departing node o at time t is denoted by $L(p_t, t)$. Since p_t is a shortest path when departing node o at time t, we have $t + L(p_t, t) \le t + L(p_{t+1}, t)$. Since the FIFO property holds on path p_{t+1}, we have $t + L(p_{t+1}, t) \le t + 1 + L(p_{t+1}, t+1)$. Hence, we have $t + L(p_t, t) \le t + 1 + L(p_{t+1}, t+1)$. Therefore, if every path between origin node o and destination node q satisfies the FIFO property, then the minimum travel time function satisfies the FIFO property. ♦

Lemma 3 is used in Section VI-B to develop the dynamic lower bounds on the minimum travel times.

VI. Dynamic Adaptations of the A* Algorithm

In dynamic networks, one may need to solve a one-to-one shortest path problem for a given departure time t or for many or all departure times. In this section, we first study the formulation of the one-to-all dynamic shortest path problem in a FIFO

dynamic network. Then a static lower bound is developed to solve the one-to-one shortest path problem for a given departure time t, using an adaptation of the A* algorithm. We later develop dynamic lower bounds and mixed static-dynamic lower bounds, that exploit special characteristics of dynamic networks to solve the one-to-one shortest path problem for all departure times.

A. One-to-One Shortest Path Problem for a Given Departure Time

The formulation of the one-to-one shortest path problem is similar to the formulation of the one-to-all shortest path problem. The only difference is that, for the one-to-one shortest path problem, the algorithm stops when the destination node is selected if a LS algorithm is used. We first present a well-known formulation of the one-to-all dynamic shortest path problem.

1) Formulation of the Dynamic Shortest Paths Problem for a Given Departure Time: The objective of the one-to-all dynamic shortest path problem is to find shortest paths from an origin node o, departing at time interval 0, to all other nodes. Minimum travel time L_i is defined by the following equations ([17] and [18]):

$$L_j = \begin{cases} \min\limits_{i \in B(j)} \min\limits_{t \geq L_i} (t + d_{ij}(t)), & j \neq o \\ 0, & j = o. \end{cases}$$

Lemma 4 (Borrowed From [18]): If the FIFO property is satisfied, the above formulation of the shortest path problem is equivalent to the following equation:

$$L_j = \begin{cases} \min\limits_{i \in B(j)} (L_i + d_{ij}(L_i)), & j \neq o \\ 0, & j = o. \end{cases}$$

Proof: $\min_{t \geq L_i}(t + d_{ij}(t)) = L_i + d_{ij}(L_i)$, since the FIFO property holds. Therefore, the equivalence of these two formulations holds. ♦

This formulation is similar to the optimality conditions for static shortest path problem. It shows that static shortest path algorithms can be extended to solve the one-to-all shortest path problem in FIFO dynamic networks. The dynamic A* algorithm adaptations of this paper are derived from this formulation. The adaptations differ only in the way one determines a lower bound $\hat{e}_{iq}(t)$ on $e_{iq}(t)$, the minimum travel time between a (node, time) pair, (i, t), to a destination node say, q. The formulation is still valid if the departure time t_0 at the origin node is not necessarily equal to zero, as it suffices to change the definition of L_i to minimum arrival time instead of minimum travel time.

Following is a description of a dynamic adaptation of the A* algorithm, to find a fastest path between an (origin, destination) node pair (o, q), departing the origin node at time t_0.

Step 1) Initialization:

$$\hat{L}_o = t_o,\ \hat{F}_o = \hat{e}_{oq}(t_o);\ \hat{L}_j = \infty,\ \hat{F}_j = \infty,\ \forall j \neq o$$
$$C = \{o\};\ S = \phi.$$

Step 2) Node selection:

$$i = \operatorname{Arg}\min_{j \in C} \hat{F}_j,\ S = S \cup \{i\}.\ C = C \backslash \{i\}.$$

Step 3) Stopping rule:
If $i = q$, then stop. Otherwise, continue.

Step 4) Update $\hat{F}_j$ and distance labels $\hat{L}_j$
For each $j \in A(i)$: If $\hat{L}_i + d_{ij}(\hat{L}_i) + \hat{e}_{jq}(\hat{L}_i) < \hat{F}_j$ then
$\hat{L}_j = \hat{L}_i + d_{ij}(\hat{L}_i);\ \hat{F}_j = \hat{L}_i + d_{ij}(\hat{L}_i) + \hat{e}_{jq}(\hat{L}_i);$
If $j \notin C$, $C = C \cup \{j\}$. Go back to Step 2).

2) Static Lower Bounds: In the previous subsection we have shown a dynamic variant of the A* algorithm to solve the one-to-one dynamic shortest path problem in a dynamic FIFO network. As explained in Section IV-B, the efficiency of the A* algorithm depends on the quality of the lower bounds on e_{iq}. Therefore, in order to adapt the A* algorithm efficiently, we need to develop effective lower bounds on e_{iq}. As we will see later in this paper, it is possible to find better lower bound on e_{iq} than those commonly used in static networks, such as those based on Euclidean distance.

For all $(i, j) \in A$, let $d_{ij}^{\min} = \min_{t=0,\ldots,M-l}\{d_{ij}(t)\}$. We construct a virtual static network with $d_{ij}^{\min}$ as the link travel times. An all-to-one static shortest path algorithm applied to the virtual network and destination node q leads to minimum travel times denoted by $e_{iq}^{\min}$ from every node i to node q. Below we prove that $e_{iq}^{\min}$ is a lower bound on $e_{iq}(t)$ for every node-time pair (i, t). This lower bound is said to be static since it does not depend on the departure time at the origin node. Note that, for a given destination, $e_{iq}^{\min}$ needs to be computed only once, which can be done during a preprocessing step.

Lemma 5: For every node i and departure time t, $e_{ij}^{\min} \leq e_{ij}(t)$. Furthermore, $e_{iq}^{\min}$ satisfies the consistency assumption, i.e., for every link $(i, j) \in A$, $e_{iq}^{\min} \leq d_{ij}(t) + e_{jq}^{\min}$.

Proof: First we prove that for every node i and departure time t, $e_{ij}^{\min} \leq e_{ij}(t)$. Suppose that p is a shortest path from node i to node q on the virtual static network. We have $e_{iq}^{\min} = \sum_{(m,n) \in p} d_{mn}^{\min}$. Suppose that $p^*(t)$ is a shortest path from i to q, which departs node i at time t. Since $p^*(t)$ is also a feasible path between node i and node q on the virtual static network, we have $e_{iq}^{\min} = \sum_{(m,n) \in p} d_{mn}^{\min} \leq \sum_{(k,l) \in p^*(t)} d_{kl}^{\min}$. By the definition of $d_{kl}^{\min}$, for every $(k, l) \in p^*(t)$ at any time interval t, $d_{kl}^{\min} \leq d_{kl}(t)$. Hence, $e_{iq}^{\min} \leq \sum_{(k,l) \in p^*(t)} d_{kl}^{\min} \leq \sum_{(k,l) \in p^*(t)} d_{kl}(t') = e_{iq}(t)$. Therefore, $e_{iq}^{\min} \leq e_{iq}(t)$.

Let us now prove that $e_{iq}^{\min}$ satisfies the consistency assumption. In the virtual static network, according to the optimality conditions, we have $e_{iq}^{\min} \leq d_{ij}^{\min} + e_{jq}^{\min}$. According to the definition of $d_{ij}^{\min}$, we have $d_{ij}^{\min} \leq d_{ij}(t)$. Therefore, $e_{iq}^{\min} \leq d_{ij}(t) + e_{jq}^{\min}$. ♦

$e_{iq}^{\min}$ is used as a lower bound on e_{iq}. Since the static lower bounds satisfy the consistency assumption, whenever the algorithm selects a node, a shortest path from the origin to that node has been determined. Thus, in the last step of the algorithm, we do not need to consider those nodes that are already in the selected node set. We update label $\hat{F}_j$ for only the neighboring nodes that are not on the selected node set.

Consider two-dimensional grid networks where the travel time of a link is its Euclidean distance divided by a certain value of travel speed. Lower bounds based on the Euclidean distance are typically obtained by dividing the Euclidian distance between the current node and the destination node, by

the minimum value of the minimum speeds among all links. This "Euclidean" lower bound is always less than or equal to, and consequently can not outperform, the static lower bound developed in this subsection. Thus, we do not compare further these two lower bounds in the experimental section, as we know by theory that the static lower bound is always more effective. Note the Euclidean lower bound also verifies the consistency assumption.

B. One-to-One Shortest Path Problems for All Departure Times

The one-to-one shortest path problem for all departure times consists in finding shortest paths from an origin node to a destination node for every departure time t at the origin node. Although one can still use $e_{iq}^{\min}$ as lower bounds, we are interested in deriving improved lower bounds. Assume that when computing fastest paths for departure time t, we have already computed fastest paths for departure time $t-1$. In FIFO networks, one can take advantage of the results obtained for departure time $t-1$ to reduce the computational effort for time interval t. This observation forms the basis for the development of a dynamic lower bound, and of a mixed lower bound. As it is shown in the experimental section, these lower bounds impact positively the computational efficiency.

1) Dynamic Lower Bounds: Assume that when we consider a departure time t at the origin node, the minimum travel times before time t have already been determined. The minimum arrival time at a node i is $t + L_i(t)$, for a departure time t at the origin node. During the course of the algorithm, the current minimum arrival time at node i is $t + \hat{L}_i(t)$, which is greater than or equal to $t + L_i(t)$. We are interested in deriving a lower bound on $e_{iq}(t + \hat{L}_i(t))$, which we will derive from a conceptual lower bound $e_{iq}(t+L_i(t))$ and from the static lower bound described in the previous subsection. Lemma 6 provides a conceptual lower bound on $e_{iq}(t + L_i(t))$ for a node i, which is on a shortest path corresponding to departure time $t-1$. We refer to this lower dynamic lower bound as the dynamic lower bound.

Lemma 6: For every departure time node t at the origin node and for every node i on a shortest path corresponding to a departure time $t-1$, $(t-1+L_i(t-1)+e_{iq}(t-1+L_i(t-1)))-(t+L_i(t))$ is a lower bound on $e_{iq}(t + L_i(t))$. Furthermore, this lower bound denoted $\tilde{e}_{iq}(t + L_i(t))$, satisfies the consistency assumption.

Proof: We first prove that $e_{iq}(t-1+L_i(t-1))-1+L_i(t-1) - L_i(t)$ is a lower bound on $e_{iq}(t + L_i(t))$. From lemma 3, we know that the FIFO property holds for minimum travel times between any two nodes in a FIFO network. For minimum travel times between the origin node and node i, we have $t + L_i(t) \geq t-1+L_i(t-1)$. Since the arrival time at node q when departing node i at time $t+L_i(t)$ is not earlier than the arrival time at node q when departing node i at time $t - 1 + L_i(t-1)$, we have: $t + L_i(t) + e_{iq}(t + L_i(t)) \geq t - 1 + L_i(t-1) + e_{iq}(t-1+L_i(t-1))$. This leads to $e_{iq}(t+L_i(t)) \geq (t-1+L_i(t-1)+e_{iq}(t-1+L_i(t-1)))-(t+L_i(t))$. We have then a lower bound on $e_{iq}(t+L_i(t))$ denoted $\tilde{e}_{iq}(t+L_i(t))$: $\tilde{e}_{iq}(t+L_i(t)) = (t-1+L_i(t-1)+e_{iq}(t-1+L_i(t-1)))-(t+L_i(t))$.

We now proceed to prove that lower bound $\tilde{e}_{iq}(t + L_i(t))$ satisfies the consistency assumption. We want to show that the following inequality is valid:

$$\tilde{e}_{iq}(t+L_i(t)) \leq e_{ij}(t+L_i(t))+\tilde{e}_{jq}(t+L_i(t)+e_{ij}(t+L_i(t))).$$

Since $e_{iq}(t-1+L_i(t-1))$ is the minimum travel time to destination node q departing node i at time $(t-1)+L_i(t-1)$, we have:

$$e_{iq}(t-1+L_i(t-1)) \leq d_{ij}(t-1+L_i(t-1)) + e_{jq}(t-1+L_i(t-1)+d_{ij}(t-1+L_i(t-1))).$$

Adding $L_i(t-1) - L_i(t) - 1$ to both sides of this inequality leads to:

$$\begin{aligned} &e_{iq}(t-1+L_i(t-1)) - 1 + L_i(t-1) - L_i(t) \\ &\quad \leq d_{ij}(t-1+L_i(t-1)) + e_{jq}(t-1+L_i(t-1) \\ &\qquad + d_{ij}(t-1+L_i(t-1)) + L_i(t-1)) - L_i(t) - 1. \end{aligned}$$

The left-hand side of the inequality is equal to: $\tilde{e}_{iq}(t + L_i(t))$. The right-hand side of the inequality is equal to: $d_{ij}(t+L_i(t))+e_{jq}(t-1+L_i(t-1)+d_{ij}(t-1+L_i(t-1)))-1+L_i(t-1)+d_{ij}(t-1+L_i(t-1))-(L_i(t)+d_{ij}(t+L_i(t)))$. The latter term is equal to $d_{ij}(t+L_i(t))+\tilde{e}_{jq}(t+L_i(t)+d_{ij}(t+L_i(t))$. Thus, we have proved that the inequality $\tilde{e}_{iq}(t + L_i(t)) \leq d_{ij}(t + L_i(t)) + \tilde{e}_{jq}(t + L_i(t) + d_{ij}(t + L_i(t))$ is valid. Therefore, $\hat{e}_{iq}(t+L_i(t)) = e_{iq}(t-1+L_i(t-1))-1+L_i(t-1)-L_i(t)$ satisfies the consistency assumption. ♦

Lemma 6 can be generalized to derive a lower bound on $e_{iq}(t + L_i(t))$ for nodes i on a shortest path corresponding to a departure time $t-k$, where k is a positive integer. We do not dwell on these lower bounds as numerical results have shown that the most effective practical lower bounds were obtained for a value of $k = 1$. These lower bounds can be used for nodes that are not on a shortest path corresponding to departure time $t-1$, but that appear on a shortest path corresponding to a departure time $t-k$. This could be useful if it is not possible to use the static lower bound in conjunction with the dynamic lower bound as described next.

2) Mixed Lower Bounds: The dynamic lower bounds are valid only for nodes on the generated shortest path departing the origin node at time $t-1$. For nodes that are not on the previous shortest path, we use $e_{iq}^{\min}$ as a lower bound. This means that two types of lower bounds will be used, as not all nodes are guaranteed to be on a previous shortest path. This may cause the consistency assumption to be violated.

The dynamic lower bound is a conceptual result only, as one may not know $L_i(t)$ when one is doing the calculations at node i. In fact, one only knows $\hat{L}_i(t)$ which is an upper bound on $L_i(t)$. We now describe how the practical lower bound is derived from the dynamic lower bound. We know that $\hat{L}_i(t) \geq L_i(t)$. It results that $e_{iq}(t-1+L_i(t-1))-1+L_i(t-1)-L_i(t) \geq e_{iq}(t-1+L_i(t-1))-1+L_i(t-1)-\hat{L}_i(t)$. Therefore, we can use the following practical lower bound for the nodes on the generated shortest path corresponding to departure time $t-1$: $e_{iq}(t-1+L_i(t-1))-1+L_i(t-1)-\hat{L}_i(t)$.

The latter practical dynamic lower bounds may have a value smaller than $e_{iq}^{\min}$. Hence, for every node i on the generated shortest path for a departure of the origin at time $t-1$, it is

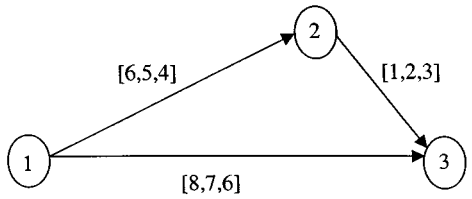

Fig. 2. A small network.

desirable to take the maximum of $e_{iq}^{\min}$ and the practical dynamic lower bound. Although the resulting lower bound does not verify the consistency assumption, as we will show later, it will eventually reduce the number of nodes selected.

The lower bounds that we adopt are given by

$$\begin{cases} e_{iq}^{\min}, & \text{if } i \text{ is not on the previous shortest path} \\ \max\left\{e_{iq}^{\min},\, e_{iq}(t-1+L_i(t-1))-1+L_i(t-1)-\hat{L}_i(t)\right\}, & \text{otherwise.} \end{cases}$$

These lower bounds are called mixed lower bounds, and are denoted $\hat{c}_{iq}(t+\hat{L}_i(t))$.

Lemma 7: The mixed lower bounds do not necessarily satisfy the consistency assumption.

Proof: We provide a counter example that shows that the consistency assumption is not satisfied by the mixed lower bound for the small network depicted in Fig. 2. This network consists of three nodes, three links and three time intervals. The numbers next to a link denote the travel times of that link for all departure times 0, 1, and 2. For instance, [8, 7, 6] on link (1, 3) means that the travel times on link (1, 3) are: 8 at time interval 0, 7 at time interval 1, and 6 at time interval 2. The travel times are assumed to be constant after time interval 2. Note that every link travel time satisfies the FIFO property.

Consider the problem of finding fastest paths from origin node 1 to destination node 3 for all departure times. If we depart node 1 at time interval 0, the minimum travel time to arrive node 3 is 8, and an optimal path is 1–3 (the travel time for path 1–2–3 is $6+3=9$). Now, let us consider time interval 1 at node 1. Node 1 was on the previous shortest path. Thus, the mixed lower bound for node 1 is

$$\hat{c}_{13}\left(1+\hat{L}_1(1)\right) = \max\left\{c_{13}^{\min},\, c_{13}(0+L_1(0))-1+L_1(0)-\hat{L}_1(1)\right\}.$$

Since $\hat{L}_1(1)=L_1(0)=\hat{L}_1(0)=0$, and $c_{13}(0)=8$, we have $\hat{c}_{13}(1+\hat{L}_1(1))=\hat{c}_{13}(1)=\max\{c_{13}^{\min},\, c_{13}(0)-1\}=\max\{6,\, 8-1\}=7$. Since node 2 was not on the previous shortest path, we use $c_{iq}^{\min}$ as a lower bound: $\hat{c}_{23}(1+c_{12}(1))=c_{23}^{\min}=1$. For this small network, if the consistency assumption were valid we would have: $\hat{c}_{13}(1)\le c_{12}(1)+\hat{c}_{23}(1+c_{12}(1))$. However, this does not hold, since $\hat{c}_{13}(1)=7>c_{12}(1)+\hat{c}_{23}(1+c_{12}(1))=5+1=6$. Thus, in this network, the mixed lower bounds do not satisfy the consistency assumption. Therefore, the mixed lower bounds do not in general satisfy the consistency assumption. ♦

For the one-to-one shortest path problem for all departure times, we need to determine a shortest path from an origin node to a destination node for every departure time t. One may use static lower bounds or mixed lower bounds. If one uses static lower bounds, one does not need to reconsider those nodes on the selected node set, since these lower bounds satisfy the consistency assumption. Mixed lower bounds, on the other hand, do not satisfy the consistency assumption.

VII. Computer Implementations and Experimental Evaluation

The dynamic adaptations of the A* algorithm discussed in the previous section have been implemented for the purpose of computational testing. In the dynamic adaptation of the A* algorithm for the one-to-one shortest path problem for one departure time, only the static lower bounds are applicable. We refer to this implementation by DAA*_S. For the one-to-one shortest path problem for all departure times, two dynamic adaptations of the A* algorithm are possible, using the static lower bounds or the mixed lower bounds. We refer to the implementation based on mixed lower bounds as DAA*_M. Dijkstra's algorithm was also implemented for comparison purposes. Note that all these algorithms are stopped as soon as the destination node is selected, at which time a shortest path from the origin node to the destination node has been determined. In each iteration of the dynamic adaptations of the A* algorithm and Dijkstra's algorithm, one needs to select a node with the minimum label from the set of candidate nodes. This operation is implemented by using a heap data structure.

In the literature, the performance of the A* algorithm is usually characterized in terms of the number of nodes selected only. The number of nodes selected, however, may, by itself, not reflect the overall efficiency of this algorithm. This is particularly true in dynamic networks. The reason for this is that finding a better lower bound usually requires extra computational time. One then needs to balance the extra computational time spent to obtain a better lower bound with the execution time saving that this lower bound may lead to. Therefore, below we report not only the number of nodes selected by the dynamic adaptations of the A* algorithm, but also their computation times.

All algorithms were implemented using the C++ programming language and tested on randomly generated networks. All computational times were obtained by running the codes on a Sun SPARC 5 Workstation.

The number of nodes selected by either the dynamic adaptation of the A* algorithm or Dijkstra's algorithm may vary with different origin–destination (OD) pairs. The number of nodes selected and the computational time of Dijkstra's algorithm are nondecreasing functions of the minimum travel time between an OD pair. The validity of this observation can be intuitively explained as follows. Since the minimum travel times vary for different OD pairs, the search area for an OD pair with a longer travel time is generally larger than the search area for an OD pair with shorter minimum travel times. Thus, comparing two algorithms by computing a shortest path using one OD pair only may not lead to conclusive results. Therefore, when we analyze the performance of the algorithms of this paper, we compare their performance based on a number of randomly selected OD pairs, and report not only the numerical results obtained for individual OD pairs, but also the average of these numerical results.

We will first compare DAA*_S and DAA*_M with Dijkstra's algorithm using a network containing 3000 nodes, 10 000 links, and 100 time intervals to analyze the performance of these algorithms as a function of OD pairs. Then we test the algorithms on networks of different sizes to analyze the performance of each algorithm as a function of the size of the test networks. If the savings of the adaptations of the A* algorithm increase with the size of the network, then this would mean that for larger networks one may benefit even more by using the adaptations of the A* algorithm instead of using an algorithm such as Dijkstra's algorithm. We conduct the computational tests using networks having n nodes and $3*n$ links (the typical average degree of a road network is around 3). For all test networks, the number of time intervals is 100. In order to assess and analyze the overall performance of the dynamic adaptation of the A* algorithm compared to Dijkstra's algorithm, the effect of individual network parameters on the performance of the algorithms will be investigated by varying a given parameter, while keeping the other parameters constant.

A. Random Network Generator

A network generator that generates random dynamic FIFO networks was implemented using the C++ programming language. The user of this network generator inputs the size of the network (number of nodes, number of links and number of time intervals) and the range of the link travel times. One can also generate a non-FIFO network, but for the purpose of this paper, only FIFO networks will be generated, since the dynamic adaptations of the A* algorithms are valid for this class of networks only.

The topology of a random network is generated in two steps. We first construct a subnetwork that ensures connectivity, and then we add random links until a desired number of links is achieved. Link travel times are selected randomly from a range given by the user, using a uniform distribution. The link travel times generated in this way may not satisfy FIFO property. In order to obtain FIFO link travel times, the following transformation is applied to the generated link travel time functions: $d^*_{ij}(t) = \min_{s \geq t} s - t + d_{ij}(s)$, where t is the arrival time and s is the departure time at node i. Transformed link travel times $d^*_{ij}(t)$ satisfy the FIFO property.

B. Computational Results for the Dynamic One-to-One Shortest Path Problem for One Departure Time

As indicated earlier in this section, in order to analyze the behavior of the algorithms with respect to different OD pairs, we first compare DAA*_S and DAA*_M with Dijkstra's algorithm in a network with 3000 nodes, 10 000 links, and 100 time intervals.

Below, we show the computational results obtained using the dynamic adaptations of A* algorithm and Dijkstra's algorithm. Since we consider the computation of a one-to-one shortest path problem for one departure time, only the static lower bounds are applicable.

We compare the number of nodes selected by the dynamic adaptation of the A* algorithm with static lower bounds (DAA*_S) and Dijkstra's algorithm in the randomly generated dynamic network. As indicated above, Dijkstra's algorithm is implemented such that it stops whenever the destination node is selected (which means a shortest path from the origin node to the destination node has been determined). Fig. 3 depicts the number of nodes selected by the heap-based implementations of Dijkstra's algorithm and of DAA*_S as a function of randomly selected OD pairs sorted in increasing order of their minimum travel times. The figure shows that, for every OD pair, the number of nodes selected by DAA*_S is always less than the number of nodes selected by Dijkstra's algorithm. For the random network with 3000 nodes, 10 000 links, and 100 time intervals, the average number of nodes selected by Dijkstra's algorithm is 5.4 times as much as the average number of nodes selected by DAA*_S. The figure also shows that the number of nodes selected by Dijkstra's algorithm, as explained earlier in this section, is a nondecreasing function of the minimum travel time between an OD pair.

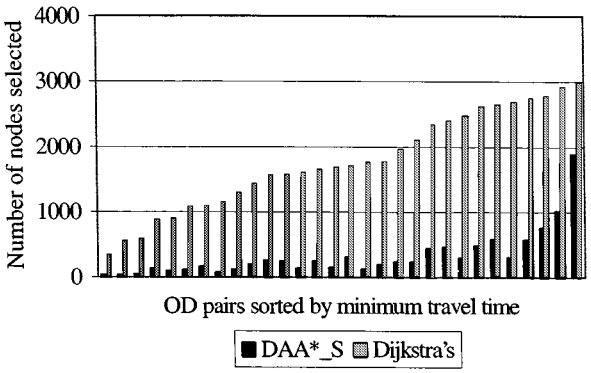

Fig. 3. Comparison of number of nodes selected by Dijkstra's algorithm and DAA*_S for a network with 3000 nodes, 10 000 links, 100 time intervals. On average, the number of nodes selected by Dijkstra's algorithm is 5.4 times the number of nodes selected by DAA*_S.

We now compare the computation times of DAA*_S and Dijkstra's algorithm. As Fig. 4 indicates, for every OD pair, the computation time of DAA*_S is less than the computation time of Dijkstra's algorithm. For this test network of 3000 nodes, 10 000 links, and 100 time intervals, the average computation time of Dijkstra's algorithm is 3.2 times the average computation time of DAA*_S.

C. Computational Results for the Dynamic One-to-One Shortest Path Problem for All Departure Times

Below, we show the computational results obtained using the dynamic adaptations of A* algorithm and Dijkstra's algorithm. Since we consider the computation of a one-to-one shortest path problem for all departure times, both static lower bounds and mixed lower bounds are applicable. We use the same network as in Section VII-B, that is a dynamic FIFO network with 3000 nodes 10 000 links, and 100 time intervals. The computation times reported in this subsection are the averages for one-to-one shortest path problem for all departure times. First, we compare the number of nodes selected by the dynamic adaptation of the A* algorithm based on mixed lower bounds (DAA*_M) to the number of nodes selected by the adaptation based on the static lower bounds (DAA*_S) for the randomly generated dynamic network. Fig. 5 shows that, for every OD pair, the number of nodes selected by DAA*_M is less than the number of nodes

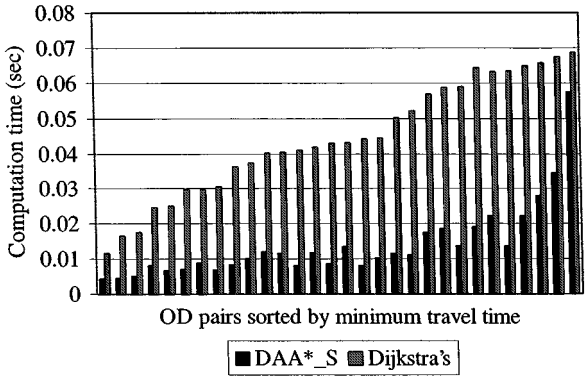

Fig. 4. Comparison of computation time of Dijkstra's algorithm and DAA*_S for a network with 3000 nodes, 10 000 links, 100 time intervals. On average, the computation time of Dijkstra's algorithm is 3.2 times the computation time of DAA*_S.

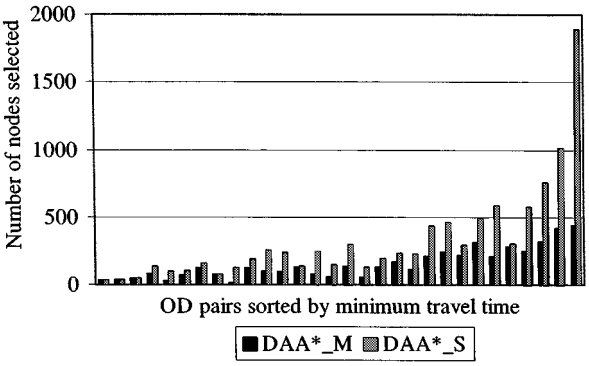

Fig. 5. Comparison of number of nodes selected by DAA*_S and DAA*_M for a network with 3000 nodes, 10 000 links, 100 time intervals. On average, the number of nodes selected by DAA*_S is 2.1 times the number of nodes selected by DAA*_M.

selected by DAA*_S. For this network, the average number of nodes selected by DAA*_S is 2.1 times the average number of nodes selected by DAA*_M. This shows that the dynamic lower bounds are more effective than the static lower bounds.

We now compare the computation times of DAA*_M and DAA*_S in the randomly generated network. Fig. 6 shows that, for most of OD pairs, the computation time of DAA*_M is less than the computation time of DAA*_S. For some nodes, the computation time of DAA*_M is greater than the computation time of DAA*_S. This is because the extra time spent to compute the lower bounds is more than the time saved by using these lower bounds. For this test network with 3000 nodes 10 000 links, and 100 time intervals, the average computation time of DAA*_S is 1.5 times the average computation time of DAA*_M.

From the computational results summarized above, we can also compare the performance of the A* algorithm with mixed lower bounds (DAA*_M) and the performance of Dijkstra's algorithm. We first compare the number of nodes selected by DAA*_M and Dijkstra's algorithm in the randomly generated dynamic network of 3000 nodes, 10 000 links and 100 time intervals. Fig. 7 shows that, for every OD pair, the number of nodes selected by DAA*_M is less than the number of nodes selected by Dijkstra's algorithm. For this network, the average number

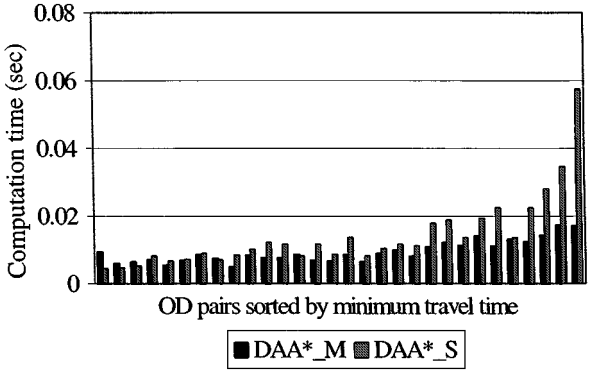

Fig. 6. Comparison of computation time of DDA*_S and DAA*_M for a network of 3000 nodes, 10 000 links, 100 time intervals. On average, the computation time of DDA*_S is 1.5 times the computation time of DAA*_M.

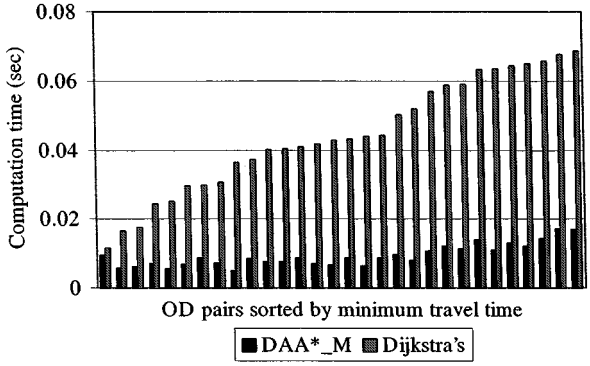

Fig. 7. Comparison of computation time of Dijkstra's algorithm and DAA*_M for a network of 3000 nodes, 10 000 links, 100 time intervals. On average, the computation time of Dijkstra's algorithm is 4.8 times the computation time of DAA*_M.

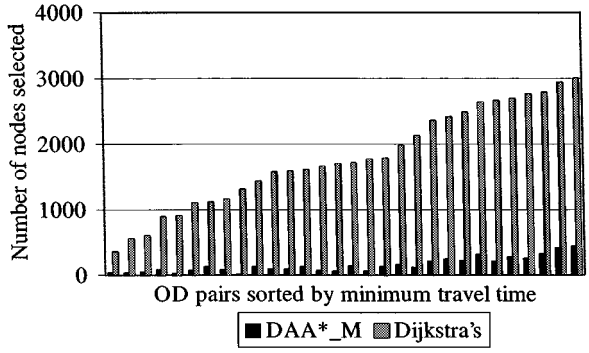

Fig. 8. Comparison of number of nodes selected by Dijkstra's algorithm and DAA*_M for a network of 3000 nodes, 10 000 links, 100 time intervals. On average, the number of nodes selected by Dijkstra's algorithm is 11.3 times the number of nodes selected by DAA*_M.

of nodes selected by Dijkstra's algorithm is about 11 times the average number of nodes selected by DAA*_M.

Fig. 8 shows that, for every OD pair, the computation time of DAA*_M is less than the computation time of Dijkstra's algorithm. For the test network used, the average computation time of Dijkstra's algorithm is 4.7 times the average computation time of DAA*_M.

Note that, similar to Figs. 3 and 4, both Figs. 7 and 8 show that the number of nodes selected by, and the computation times

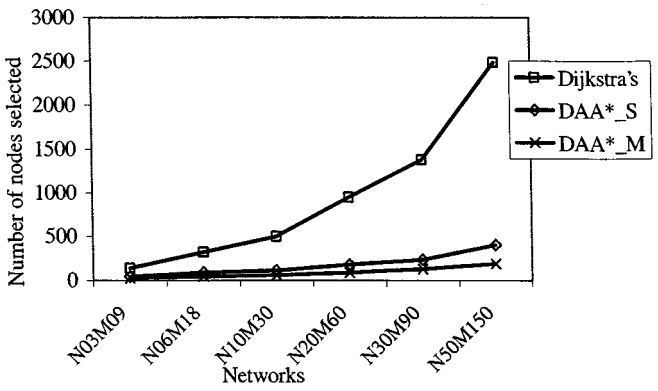

Fig. 9. Comparison of average number of nodes selected for networks with different sizes. NxxMyy means that the network has xx00 nodes and yy00 links. For instance, N50M150 represents the size of a network with 5000 nodes and 15 000 links.

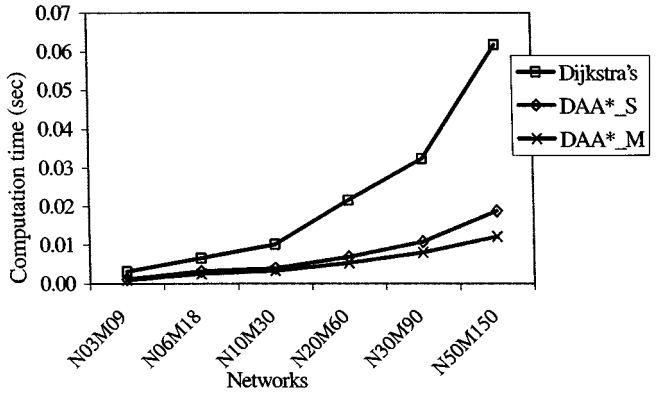

Fig. 10. Comparison of computation times for networks with different sizes. NxxMyy means that the network has xx00 nodes and yy00 links. For instance, N50M150 represents the size of a network with 5000 nodes and 15 000 links.

of, Dijkstra's algorithm are nondecreasing functions of the minimum travel time between the OD pair.

D. Performance Study as a Function of Network Size and Network Parameters

We first show the performance of the implementations of DAA*_S, DAA*_M and Dijkstra's algorithms for different network sizes while keeping the number of links in the test networks equal to three times the number of nodes. The reason for adopting test networks in which the number of arcs is three times the number of nodes, is to emulate traffic networks where the ratio between the number of links and the number of nodes is typically around 3.

Fig. 9 shows that the numbers of nodes selected by Dijkstra's, DAA*_S and DAA*_M algorithms increase with the size of the network, but at different rates. Note that the x-axis does not follow a linear scale in Figs. 9 and 10. The rate of increase of the number of nodes selected by Dijkstra's algorithm is higher than the rate of increase of the number of nodes selected by DAA*_S. This latter rate is higher than the rate of increase of the number of nodes selected by DAA*_M. Fig. 10 shows that the computation time savings for the different implementations behave similarly to the number of nodes selected by the algorithms.

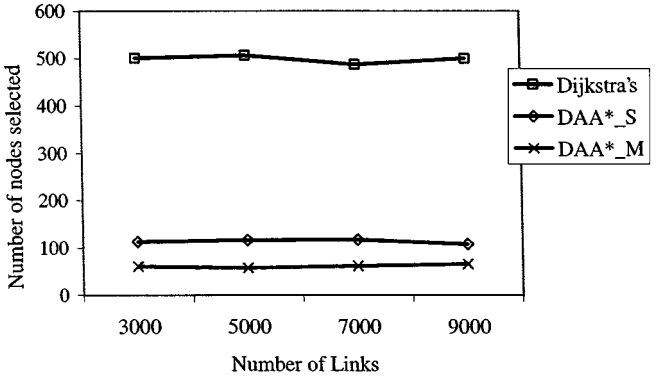

Fig. 11. Comparison of average number of nodes selected for networks composed of 1000 nodes, 100 time intervals, and a varying number of links.

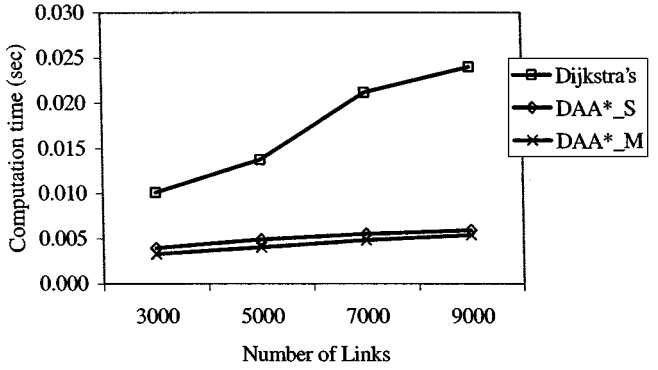

Fig. 12. Comparison of computation times for networks composed of 1000 nodes, 100 time intervals, and a varying number of links.

The results exhibited in Figs. 9 and 10 suggest that, for networks with larger sizes, the dynamic adaptations of A* algorithm lead to larger savings in terms of number of nodes selected as well as in terms of computation times.

In the rest of this subsection, we analyze the efficiency of the computer implementations of DAA*_S, DAA*_M and Dijkstra's algorithm, as a function of different network parameters while keeping the other parameters constant.

First, we consider the effect of the number of links while keeping the number of nodes and the number of time intervals constant. Fig. 11 shows that, for networks with 1000 nodes and 100 time intervals, the number of nodes selected by DAA*_S, DAA*_M or Dijkstra's algorithm are almost constant functions of the number of links. This can be explained as follows. If the number of nodes in the network is constant, the average number of nodes that fall in the search areas corresponding to the algorithms for a given destination would not change even if the number of links change. However, as one may expect, the number of nodes selected by Dijkstra's algorithm is greater than the number of nodes selected by either DAA*_S or DAA*_M algorithms.

Fig. 12 shows that, for networks with 1000 nodes, the computation time of Dijkstra's, DAA*_S and DAA*_M algorithms increase with the number of links, but at different rates. The rate of increase of the computation time of Dijkstra's algorithm is higher than the rate of increase corresponding to DAA*_S and DAA*_M algorithms. Although the numbers of nodes selected by different algorithms are almost constant, more links will be

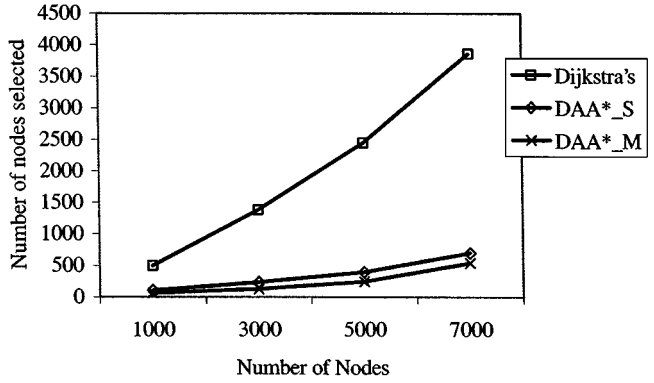

Fig. 13. Comparison of average number of nodes selected for networks composed of 9000 links, 100 time intervals, and a varying number of nodes.

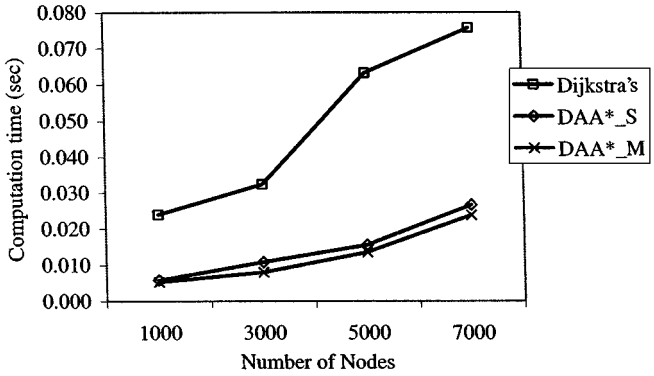

Fig. 14. Comparison of computation times for networks composed of 9000 links, 100 time intervals, and a varying number of nodes.

scanned whenever a node is selected if the number of links increases. Thus, the computation time increases with the number of links.

Now, we examine the effect of the number of nodes on computational performance while keeping the number of links and the number of time intervals constant. Fig. 13 shows that, for networks with 9000 links and 100 time intervals, the number of nodes selected by DAA*_S, DAA*_M or Dijkstra's algorithm increase with the number of nodes, but at different rates. The rate of increase of the number of nodes selected by Dijkstra's algorithm is higher than the rate of increase corresponding to DAA*_S and DAA*_M algorithms.

Fig. 14 shows that, for test networks with 9000 links and 100 time intervals, the computation times of the different algorithms behave similarly to the number of nodes selected, as a function of the number of nodes in the network, as was shown in Fig. 9.

Numerical results have shown that the computational saving ratios do not vary as a function of the number of time intervals. Numerical results supporting this conclusion are not included in this paper, but the conclusion is intuitively valid.

VIII. Conclusions

In this paper, we presented efficient adaptations of the A* algorithm for computing fastest paths between one origin node and one destination node in dynamic networks, for one or for multiple departure times at the origin node. These dynamic adaptations of the A* algorithm are based on effective lower bounds on minimum travel times, that exploit the FIFO properties of dynamic data and the special structure of the time-expanded implicit representation of a dynamic network. The adapted algorithms were implemented and their computational performance was experimentally evaluated and tested. The performance of the computer implementations of the dynamic adaptations of the A* algorithm are compared to a dynamic adaptation of Dijkstra's algorithm that is stopped as soon as the destination node is reached. Results using a network containing 3000 nodes, 10 000 links, and 100 time intervals showed a saving ratio of 11, in terms of number of nodes selected, and a saving ratio of 5 in terms of computation times. The effect of the network size on the performance of these adaptations was also computationally studied. It was shown that the computational savings, in terms of both the number of nodes selected and the computation time, increase with the network size. These encouraging results demonstrate the efficiency of the algorithms developed in this paper.

We now present some logical extensions to the research work presented in this paper. First, it would be interesting to extend the results of this paper to continuous-time dynamic networks. The static lower bound remains valid in this context, while the dynamic lower bound may need to be adapted for cases where one knows only the results of fastest path computations at some earlier finite number of time instants. The discussion at the end of Section VI-B-2 summarizes one such dynamic lower bound. Second, one may also consider investigating the extension of the A* algorithm to non-FIFO networks. Note that the static lower bound is valid in this context as well. Third, the algorithms of this paper offer possibilities for designing and developing efficient parallel implementations for the computation of dynamic shortest paths. Decomposition strategies can be developed in the domains of pairs of origin–destination nodes and the network topology. An efficient strategy should balance the communication time, the computation time and the idle time. Finally, the ideas presented in this paper can be extended to develop time-based algorithms, for discrete-time as well as continuous-time dynamic networks, to compute shortest path from one origin node to all other nodes if one changes the departure time at the origin. In [22], an example of such extensions is developed to compute single-origin minimum travel-time path trees for all possible departure times in continuous-time FIFO dynamic networks. The design of other time-based algorithms for other dynamic shortest path problems, is the subject of ongoing research. The first author and other collaborators have developed algorithms based on the concept of re-optimization, which can be viewed to share, to a certain degree, some fundamental similarities with the derivations in this paper. The algorithms led to encouraging computational results [23]. We will report on these algorithms in forthcoming papers.

References

[1] I. Chabini and B. Dean, "Shortest path problems in discrete-time dynamic networks: Complexity, algorithms, and implementations," Massachusetts Institute of Technology, Cambridge, MA, Tech. Rep., 1999.

[2] I. Chabini and V. Yadappanavar, "Advances in discrete-time dynamic data representation with applications to intelligent transportation systems," *Transport. Res. Rec.*, vol. 1771, pp. 201–208, 2001.

[3] E. P. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, 1968.
[4] D. E. Kaufman and R. L. Smith, "Fastest paths in time-dependent networks for intelligent-vehicle-highway systems application," *IVHS J.*, vol. 1, pp. 1–11, 1993.
[5] J. L. Bander and C. C. White, III, "A heuristic search algorithm for path determination with learning," *IEEE Trans. Syst., Man, Cybern. A*, vol. 28, pp. 131–134, 1998.
[6] I. Chabini, "Shortest paths in continuous-time traffic networks with time-dependent piece-wise linear link travel time and cost functions," Transport. Res. Part B: Methodological, 2001, to be published.
[7] J. N. Nilsson, *Problem-Solving Methods in Artificial Intelligence*. New York: McGraw Hill, 1971.
[8] I. Pohl, "Heuristic search viewed as path finding in a graph," *Artif. Intell.*, vol. 1, pp. 193–204, 1970.
[9] J. Pearl, *Heuristic Intelligent Search Strategies for Computer Problem Solving*. Reading, MA: Addison-Wesley, 1984.
[10] L. B. Golden and M. Ball, "Shortest paths with Euclidean distance: An explanatory model," *Networks*, vol. 8, pp. 297–314, 1978.
[11] R. Sedgewick and J. S. Vitter, "Shortest path in Euclidean graphs," *Algorithmica*, vol. 1, pp. 31–48, 1986.
[12] J. L. Bander and C. C. White, III, "A new route optimization algorithm for rapid decision support," in *Proc. SAE/IEEE-VTS VNIS*, Detroit, MI, 1991, pp. 709–728.
[13] J. W. Lark, C. C. White, III, and K. Syverson, "A best-first search algorithm guided by a set-valued heuristic," *IEEE Trans. Syst., Man, Cybern.*, vol. 25, pp. 1097–1101, 1995.
[14] A. Ziliaskopoulos and H. Mahmassani, "Time-dependent shortest path algorithms for real-time intelligent vehicle highway system applications," *Transport. Res. Rec.*, no. 1408, pp. 94–100, 1993.
[15] K. L. Cooke and E. Halsey, "The shortest route through a network with time-dependent internodal transit times," *J. Math. Anal. Appl.*, vol. 14, pp. 492–498, 1966.
[16] I. Chabini, "A new shortest paths algorithm for discrete dynamic networks," in *Proc. 8th IFAC Symp. Transport Systems*, 1997, pp. 551–556.
[17] ——, "Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time," *Transport. Res. Rec.*, pp. 170–175, 1998.
[18] S. E. Dreyfus, "An appraisal of some shortest path algorithms," *Oper. Res.*, vol. 17, pp. 395–412, 1969.
[19] B. H. Ahn and J. Y. Shin, "Vehicle routing with time windows and time-varying congestion," *J. Opl. Res. Soc.*, vol. 42, pp. 393–400, 1991.
[20] I. Chabini and V. Yadappanavar , "A bitwise algorithm for fastest paths in time-dependent networks," *Transport. Res. Rec.*, vol. 1771, pp. 209–218, 2001.
[21] R. Ahuja, T. Magnanti, and J. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1993.
[22] I. Chabini and V. Yadappanavar, "Algorithms for single-origin minimum time path problems in networks with piecewise linear time-dependent link travel time functions," Transport. Res. Rec., 2002, to be published.
[23] N. Grier and I. Chabini, "A new algorithm for determining minimum time path trees in FIFO time dependent networks," presented at the Transportation Research Board Annu. Meeting, Washington, DC, Jan. 2002.

Ismail Chabini received an Ingenieur d'Etat degree (Bachelor's and Master's degrees equivalent) in electromechanical engineering from the Ecole Nationale de l'Industrie et des Mines, Morocco, in 1986 and 1988, respectively. He received the M.S. and Ph.D. degrees, both in computer science, from the University of Montreal, Montreal, QC, Canada, in 1994 and 1990, respectively.

Currently, he is a faculty member at the Massachusetts Institute of Technology, Cambridge. He has been the Principal Investigator of multiple research projects funded by industry and government sponsors, such as the National Science Foundation. One of the foci of his research activities has been the design of models, algorithms, and software systems needed to solve transportation analysis and operation problems, in particular those that arise in dynamic modeling and real-time management of traffic flow networks equipped with monitoring, control, and computing technologies.

Dr. Chabini's research work has received various awards, including a 1998 CAREER Award from the National Science Foundation. He is an Associate Editor of the IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, and of the journal *Networks and Spatial Economics*. He is an Area Editor of the *ITS Journal*. He is also a member of the International Federation on Automatic Control, the Committee on Transportation Systems, and the Transportation Research Board Committee on Transportation Network Modeling.

Shan Lan received the B.S. and M.S. degrees in transportation from the Southeast University, Nanjing, China, in 1992 and 1995, respectively. He is working toward the Ph.D. degree at the Massachusetts Institute of Technology, Cambridge.

From 1995 to 1997, he was a Lecturer and Researcher in the Department of Transportation, Southeast University, Nanjing, China. His current research interests include computer algorithms and mathematical models for intelligent transportation systems and solving large-scale optimization problems for airlines.