Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 08-039

Spatio-temporal Network Databases and Routing Algorithms

Betsy George, Shashi Shekhar, and Sangho Kim

November 17, 2008

Spatio-temporal Network Databases and Routing Algorithms

Betsy George, Shashi Shekhar, Sangho Kim
Department of Computer Science and Engineering
University of Minnesota
200 Union St SE, Minneapolis, MN 55455, USA
E-mail: $[bgeorge, shekhar, sangho]$@cs.umn.edu
WWW home page: http://www.spatial.cs.umn.edu/

This work was supported by the NSF III grant (IIS-0713214), Department of Defense, and Minnesota Department of Transportation. The content of this work does not necessarily reflect the position or policy of the government and no official endorsement should be inferred.

Abstract

Spatio-temporal networks are spatial networks whose topology and parameters change with time. These networks are important for many critical applications such as emergency traffic planning and route finding services and there is an immediate need for models that support the design of efficient algorithms for computing the frequent queries on such networks. This problem is challenging due to the potentially conflicting requirements of model simplicity and support for efficient algorithms. Time expanded networks, which have been used to model dynamic networks, employ replication of the network across time instants, resulting in high storage overhead and algorithms that are computationally expensive. In contrast, the proposed time-aggregated graph (TAG) does not replicate nodes and edges across time; rather it allows the properties of edges and nodes to be modeled as a time series. Since the model does not replicate the entire graph for every instant of time, it uses less memory and the algorithms for common operations (e.g. connectivity, shortest path) are computationally more efficient than those for time expanded networks. One important query on spatio-temporal networks is the computation of shortest paths. Developing efficient algorithms for computing shortest paths in a time varying spatial network is challenging because these journeys do not always display the optimal substructure, making techniques like dynamic programming inapplicable. In addition, shortest paths can be computed either for a given start time or to find the start time and the path that leads to least travel time journeys (best start time journeys). In this paper, we present algorithms for shortest path computations in both contexts using the proposed TAG and arrival time series transformation (ATST). We present the analytical cost models for the algorithms and provide an experimental comparison of performance with existing algorithms.

Keywords: time-aggregated graphs, shortest paths, spatio-temporal data bases

I. INTRODUCTION

The underlying data of interest for many significant applications such as transportation networks is structured as a spatio-temporal network, which consists of a finite collection of points (i.e., nodes) with location information, the line-segments (i.e., edges) connecting the points, and the time-varying attributes attached to these elements. For example, a spatio-temporal network database for a traveler's trip planning may store the intersections as nodes, the road segments as edges, and the time dependent travel time attached to the road segments. In the case of evacuation planning, time dependent capacity may be added to the road segments as another important attribute.

A. An Illustrative Application Domain

Transportation networks are the kernel framework of many advanced transportation systems such as the Advanced Traveler Information System and Intelligent Vehicle Highway Systems. Transportation networks are spatio-temporal in nature and require significant database support to handle the storage of their large amounts of multi-dimensional data. Many important applications based on transportation networks, including travelers' trip planning, consumer business logistics, and evacuation planning need to be built upon spatio-temporal network databases. To illustrate the significance of the temporal nature of these networks, we provide a categorization based on the nature of travel time variation, as shown in Figure 1. Let us consider the categories now.

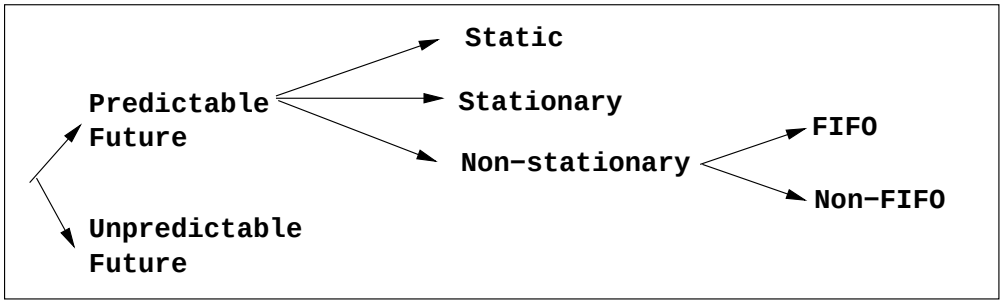

Fig. 1. Categorization of Spatio-temporal Networks (Note: FIFO is an acronym for First In First Out)

Incidents that happen without any warning and against all forecasts are part of *unpredictable future*. Random road accidents and consequent changes in traffic patterns fall under this category. In the absence of such events, changes in the network largely follow known patterns. In such *predictable future* scenarios, we assume that the changes in network parameters and topology are known in advance. One example would be planning for atypical events such as hurricane, whose paths are somewhat predictable for hours, if not days in advance. Evacuation managers may be interested in using spatio-temporal network databases to make informed decisions about evacuation route planning. Another example is public transport planning. In the absence of random events, these systems adhere to a schedule and their variations over the day are known in advance.

A *static network* would represent a network whose parameters and topology do not change with time whereas in a *stationary network*, the route preferences are time-invariant despite changes in network parameters. A static graph could represent a transportation network which does not

show significant changes (for example, during the night). An example of a stationary network would be a road network where driving on a highway from the origin to the destination is always faster than local roads, though the congestion on the highway and consequently the travel time can change over time. Here, the route preferences do not change with time ireespective of the variations in network parameters.

A transportation network displays *non-stationary* characteristics when route preferences change with time, possibly due to non-uniform congestion levels at different parts of the network or a time-dependent availability of certain road segments or faster services in a public transport system. Another such scenario is where commuters trying to find a suitable time to start their commute so that they spend the least time in traffic. Figure 2 illustrates traffic sensor networks on urban highways which measure congestion levels at two different times (e.g. 5:07pm and 9:37pm) illustrating possible changes in shortest routes at different times of day. With the increasing use of sensor networks to monitor traffic data on spatial networks and the subsequent availability of time-varying traffic data, it becomes important to incorporate this data in the models and algorithms related to transportation networks. However, existing spatio-temporal databases do not offer adequate support for spatio-temporal networks.

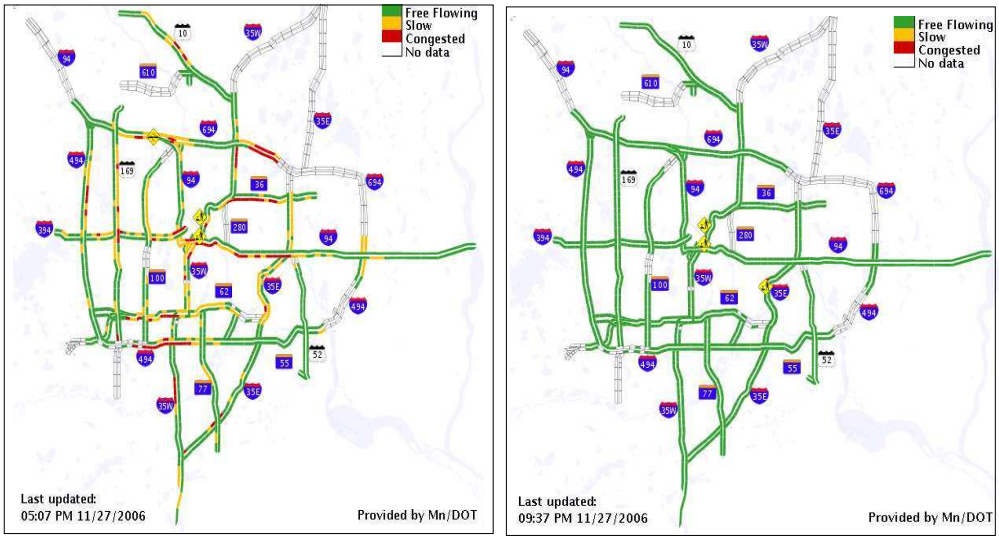

Fig. 2. Sensor networks periodically report time-variant traffic volumes on Twin Cities highways (Best viewed in color, Source: Mn/DOT)

In a network, if a flow [5] arrives at the end node of each edge in the same order as they started at the start node of the edge, the edge travel costs are said to follow the FIFO (First In

First Out) property and the edge is a FIFO edge. If every edge (and node) in a network follows FIFO propery, the network is a *FIFO network*. Common examples of FIFO traffic include single lane traffic and a bus schedule wgere all buses offer the same level of service with respect the routes and speed. In a FIFO network waits do not improve the total travel time whereas this need not always be the case in a non-FIFO network.

An edge that is not a FIFO edge is called a non-FIFO edge. If there is a non-FIFO edge in a network, it is a *non-FIFO network*. An example of a non-stationary non-FIFO scenario is the public transportation system where different services such as express and non-express services are offered between the same origin-destination pairs. A person who drives to a parking lot to transfer to a public transport vehicle very often has a choice of a variety of services such as express and non-express services. Here the arrival at the destination is not always ordered by the start times and if a faster service becomes available, waits at the intermediate stops could possibly benefit in terms of total travel time. Trip scheduling in such systems requires routing algorithms that account for the time variance in the transportation schedule to reduce the total travel time, allowing for waits to the extent that improves the total travel time. It becomes necessary to pose the problem as a time variant graph routing problem so that the commuters can find the schedule of least duration and minimize their travel times. Other examples of non-FIFO scenario include multilane highways where overtaking is allowed.

Another significant application of spatio-temporal routing algorithms is in the problem of finding the start time that leads to a journey which minimizes the time spent in the network. This can help reduce the fuel costs in applications such as freight delivery services. Companies such as United Parcel Service (UPS) are actively searching ways to reduce the fuel consumption by addressing factors such as idling at the left turns and in many metro areas left turn restrictions depend on the time of day. According to the New York Times “The research at U.P.S. is paying off. $\cdots$ saving roughly three million gallons of fuel in good part by mapping routes that minimize left turns” [7]. In addition to changes in the congestion levels, more predictable factors such as signal delays and waits at railroad crossings should not be ignored while computing travel routes.

B. Broad Challenges

Spatio-temporal networks that are encountered in numerous, significant areas of every day life display time dependence of edge and node properties and topological structure. Conventional graph algorithms cannot easily be applied to snapshot networks at discrete time instants to evaluate frequent queries without accounting for relationships among snapshots.

Spatio-temporal networks raise many challenges for database research. Due to their potentially large and evergrowing sizes, a storage-efficient representation is critical to reduce and possibly eliminate redundant information across different time-points. Second, new data model concepts need to be investigated to represent and classify potentially new alternative semantics for common graph operations such as shortest-path and connectivity. For example, a shortest path between a given pair of nodes may have at least two interpretations, one for a given start time-point and the other for the shortest travel-time for any start time in a given time interval. A third challenge is the design of efficient and correct query processing strategies and algorithms since some of the commonly assumed properties of graphs may not hold for spatio-temporal graphs. For example, one assumption concerns the optimal substructure (required in dynamic programming , [5]) of shortest paths in a graph. While each prefix path (path from a source node to an intermediate node in an optimal path) is optimal in a static graph, it may not be optimal in a spatio-temporal graph due to a potential wait at an intermediate node, and violation of the first-in-first-out (FIFO) property.

C. Related Work and Our Contributions

Related work in the field of databases falls into three broad categories (1)spatial network databases, (2) graph databases, and (3) spatio-temporal databases. The recent release of Oracle (version 10g) includes a network data model to store and maintain the connectivity of link-node networks and supports basic features such as shortest path [24]. The Network Analyst extension of ArcMap from ESRI supports a network geodatabase and provides basic algorithms (e.g., shortest path, service area, closest facility, etc.) [11]. However, these products do not address the time variance of spatial networks, which is crucial in applications such as route computations and emergency planning.

Graph databases [9]–[11], [31], [34], [35] also primarily deal with spatial networks that do not vary with time. Those graph databases that account for temporal variations, perform computations

over a snapshot of the network [8], [17], [29], and do not consider the interplay between edge travel times and the existence of edges. For example, Ding [8] proposed a model that addresses time-dependency by associating a temporal attribute to every edge and node of the network so that its state at any instant of time can be retrieved. This model performs path computations over a snapshot of the network. Since the network can change over the time taken to traverse these paths, this computation might not give realistic solutions. The model does not propose an algorithm for the least travel time paths. Although the need for live traffic information is increasing, there has been little work on the modeling and algorithms for spatio-temporal network databases. Chorochronos [22], studied various aspects of spatio-temporal databases including ontology, modeling, and implementation. However, this framework does not adequately address spatio-temporal networks.

Route computations under a first-in-first-out (FIFO) assumption, where arrival times are ordered by start times, were explored concurrently via travel time patterns [19] and our recent work [12] using ideas like the flow speed model [19] and the existence of some (though not all) paths with an optimal prefix property. Shortest path computation in FIFO networks was also handled by [2] which characterized the problem using consistency condition that requires the edge travel times to change at the rate slower than real time ($\frac{d(travelTime)}{d(realTime)} \leq 1$). Due to the FIFO assumption, these algorithms were not adequate for handling general non-FIFO cases such as multilane traffic where drivers are allowed to change lanes.

Research in Operations Research is based on the time expanded network [20], [21], [23], [25], [27], [33]. This model duplicates the original network for each discrete time unit $t = 0, 1, \ldots, T$ where T represents the extent of the time horizon. The expanded network has edges connecting a node and its copy at the next instant in addition to the edges in the original network, replicated for every time instant. This significantly increases the network size and is very expensive with respect to memory. Because of the increased problem size due to replication of the network, the computations become expensive.

As the first step towards the study of spatio-temporal network databases, we previously proposed a spatio-temporal network model named the time aggregated graph [13], [14]. We proposed the shortest path algorithms SP-TAG and BEST [12] to compute the shortest path for a fixed start time and the best start time respectively. The SP-TAG algorithm assumed that travel times followed the FIFO property and hence that arrival times are strictly ordered

by start times. In computing the best start time shortest path, the BEST algorithm used a label correcting approach [3], [5] and accounted for non-FIFO travel times. We also proposed an A* formulation of the fixed start time algorithm (SP-TAG*) [15] which showed a better computational performance compared to the greedy SP-TAG.

Our Contributions

This paper is an expanded version of our recent symposium paper [12] with the following additional results. First, we introduce a categorization of spatio-temporal networks (See Figure 1) which reveals a computational structure of routing problem. We proceed to present an algorithm for the fixed start time shortest path for the general case that removes the FIFO restriction in our own and other related work and avoids precomputation of shortest paths from every node to the destination required by the existing non-FIFO network algorithms. We propose a transformation called Arrival Time Series Transformation (ATST). Based on the transformation, a greedy algorithm called Non-FIFO SP-TAG (NF-SP-TAG) is proposed to find fixed start time shortest paths in a non-FIFO network. In addition, we introduce CP-ATST-BEST that computes the best start time, a refined algorithm which exploits ATST and the idea of prioritized concurrent version of NF-SP-TAG.

We present an expanded experimental analysis of the fixed start time shortest path algorithms and best start time shortest path algorithms and compare their performances to existing approaches. The new experimental results include

(a) Performance analysis of all algorithms with respect to the average degree of the network.

(b) Analysis of the NF-SP-TAG algorithm with respect to network size (number of nodes) and length of time series.

(c) Experimental analysis of the CP-NF-SP-TAG algorithm to assess its performance with respect to network size (number of nodes) and length of time series.

(d) Experimental performance comparison of time iterated SP-TAG* and SP-TAG algorithms with respect to network size and time series length.

The proposed model and algorithms are evaluated with a real world static graph appended with a synthetically generated travel time series.

D. Scope and Outline of the Paper

Scope: The TAG representation and various algorithms presented in this paper assume that the future value of edge attributes in a spatio-temporal network are largely predictable. This assumption may not be unreasonable in planning contexts that use a set of scenarios. Thus the results of the proposed algorithms may be considered as a view of the expectation. This assumption may also not be unreasonable in operational situations such as routing in a schedule-based public transportation sustem on days of high-schedule-adherance, when there is a lack of random events such as major accidents causing unanticipated delays. We assume that the edge costs are available at discrete instants of time and do not consider continuous variations with respect to time.

Outline of the paper: The rest of the paper is organized as follows. The paper focuses primarily on routing algorithms in a spatio-temporal network where route choices can change with time. Routing algorithms to compute shortest paths are presented in two different contexts: 1) for a fixed start time (earliest arrival time journeys) and 2) for the best start time which minimizes the travel time over the entire time horizon (least duration journeys). The rest of the paper is organized as follows. Section 2 provides a brief description of the time aggregated graph model that is used to represent time dependent graphs. Section 3 describes the shortest path algorithm for a given start time (earliest arrival shortest paths) for FIFO and non-FIFO spatio-temporal networks. The results from experimental analysis of these algorithms are also included in this section. Section 4 describes algorithms to compute the best start time at a given source node for any destination node (least duration journeys). We also present the experimental evaluation of least duration journey algorithms in Section 4. For the sake of completeness, we have included previously proposed shortest path algorithms SP-TAG and BEST [12] along with new algorithms, NF-SP-TAG and CP-ATST-BEST. In Section 5 we conclude and describe the direction of future work.

II. Representation of Spatio-temporal Networks

This section provides a brief description of time aggregated graphs used to model time dependent networks, a classification of shortest path algorithms based on start time choice and network characteristics, and a discussion of the challenges that formulation of these algorithms pose.

A. Time Aggregated Graphs

Spatial networks that show time-dependence serve as the underlying networks for many applications such as routing in transportation networks. Traditionally, graphs have been extensively used to model spatial networks (e.g. road networks) [31]; weights assigned to nodes and edges are used to encode additional information. In a real world scenario, it is not uncommon for these network parameters to be time-dependent. It is important to be able to formulate computationally efficient and correct algorithms for the shortest path computation that take into account the dynamic nature of the networks. Models of these networks need to capture the possible changes in topology and values of network parameters with time and provide the basis for the formulation of computationally efficient and correct algorithms for the frequent computations like shortest paths.

Given a set of frequent queries posed by an application on a spatial network and the pattern of variations of the spatial network with time, we need to find a model that supports efficient and correct algorithms for computing the query results, while trying to minimize the storage and cost of computation. In this section we discuss the basics of the model used to represent time dependent spatial networks called "time aggregated networks" [13]. The algorithms presented in this paper are formulated based on this model. Time aggregated graphs can not only capture the time-dependence of network parameters, but also account for the possibility of edges and nodes being absent during certain instants of time. A graph $G = (N, E)$ consists of a finite set of nodes N and edges E between the nodes in N. If the pair of nodes that determines the edge is ordered, the graph is directed; if it is not, the graph is undirected. In most cases, additional information is attached to the nodes and edges. In this section, we discuss how the time dependence of these edge/node parameters are handled in the proposed time-aggregated graph model.

We define the time-aggregated graph as follows.

$taG = (N, E, TF, f_1 \dots f_k, g_1 \dots g_l, w_1 \dots w_p | f_i : N \rightarrow \mathbb{R}^{TF}; g_i : E \rightarrow \mathbb{R}^{TF}; w_i : E \rightarrow \mathbb{R}^{TF})$

where N is the set of nodes, E is the set of edges, TF is the length of the entire time interval, $f_1 \dots f_k$ are the mappings from nodes to the time-series associated with the nodes, $g_1 \dots g_l$ are mappings from edges to the time series associated with the edges, and $w_1 \dots w_p$ indicate the time dependent weights (eg. travel times) on the edges.

Each edge has an attribute, called an edge time series, that represents the time instants for

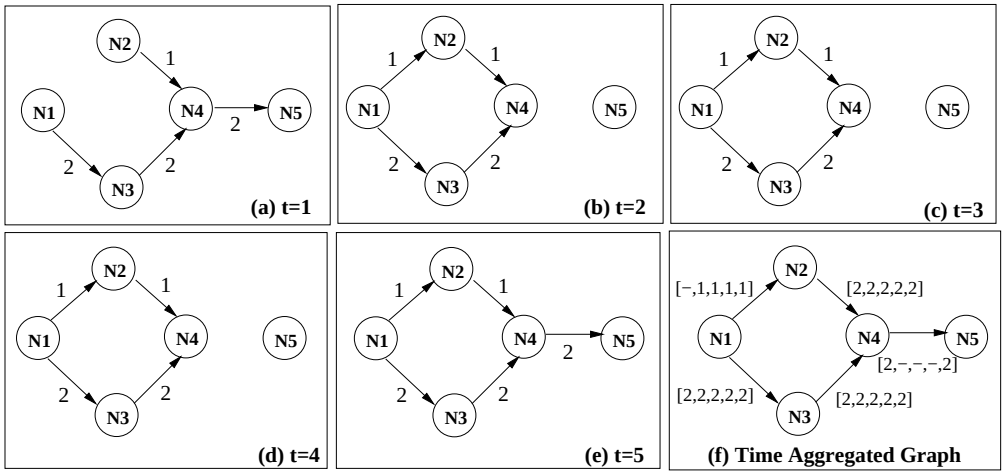

Fig. 3. Network at Various Time Instants and the Time Aggregated Graph

which the edge is present. This enables the time aggregated graph to model the topological changes of the network with time. We assume that each edge travel time has a positive minimum and the presence of an edge at time instant t is valid for the closed interval $[t, t+\sigma]$.

Figure 3(a,b,c,d,e) shows a network at five time instants. The network topology and parameters change over time. For example, edge N4-N5 is present at time instants $t = 1, 5$, and absent at $t = 2, 3, 4$. The time aggregated graph that represents this dynamic network is shown in Figure 3(f). In this figure, edge N4-N5 has an attribute, $[2, -, -, -, 2]$, which is its weight time series, indicating the weight of the edge at instants $t = 1, 2, 3, 4, 5$. Though this model can include spatial properties at nodes and edges, these properties are not incorported in the algorithms presented in this paper. Figure 4(a) shows the time aggregated graph (corresponding to Figure 3(a),(b),(c)) and a time expanded graph that represents the same scenario. Edge weights in a time expanded graph are not explicitly shown as edge attributes; instead they are represented by edges that connect the copies of the nodes at various time instants. For example, the weight 1 of edge N1-N2 at $t = 1$ is represented by connecting the copy of node N1 at $t = 1$ to the copy of node N2 at time $t = 2$. The time expansion for the example network needs to go through 7 steps since the latest edge traversal in the network ends at $t = 7$. The traversal of the edge N3-N4 that starts at $t = 3$ ends at $t = 7$, the travel time of the edge being 4 units. The number of nodes is larger by a factor of T, where T is the number of time instants and the number of edges is also larger in number compared to the time-aggregated graph. If the value of T is

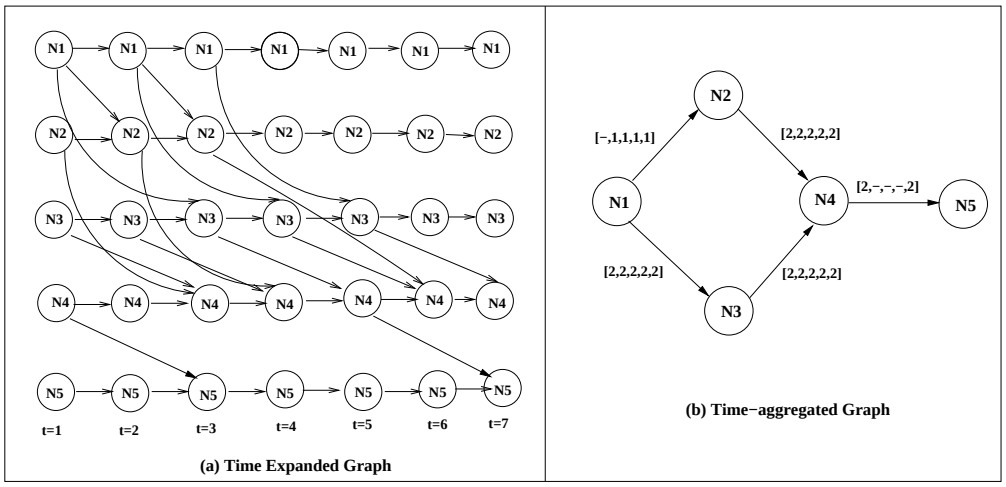

Fig. 4. Time-aggregated Graph vs. Time Expanded Graph

very large in a spatial network, it would result in enormously large time expanded networks and consequently slow computations.

Comparison of Storage Costs with Time Expanded Networks:

According to the analysis in [32], the memory requirement for time expanded network is $O(nT)+$ $O((n+m)T)$, where n is the number of nodes and m is the number of edges in the original graph. The framework of a time aggregated graph would require a memory of $O(n+m)$, where n is the number of nodes and m is the number of edges. Edges and nodes with time-varying attributes have attribute time series associated with them. If the number of edges and nodes that display time dependence is fraction α of the the total number, the memory required is $O(\alpha m + \alpha n)$, assuming an adjacency list representation. This condition can occur in many cases such as in a network that consists of a large number of residential streets away from the densely populated areas where the traffic levels remain more or less the same throughout the day. The total memory requirement for a time aggregated graph is $O(n + m + \alpha m + \alpha n)$. This comparison shows that the memory usage of time-aggregated graphs is less than that of time expanded graphs if $\alpha < T$.

B. TAG and Categorization of Spatio-temporal Networks

This section provides a brief description of the categorization of spatio-temporal networks based on the nature of the temporal variations and consequently the feasibility of algorithm design techniques such as dynamic programming. Spatio-temporal networks fall into three cat-

egories, namely, static networks stationary networks, and non-stationary networks as illustrated by Figure 1. Figure 5(a), (b), and (c) show an example of static, stationary and non-stationary networks. A static network does not display a time dependence in its parameters and topology as illustrated by Figure 5. Here the edge parameters are represented by scalars rather than a time series. The classification of spatio-temporal networks into staionary and non-stationary networks is based on the changes route preference order with time. Stationarity means the following: if two reward sequences $R_1, R_2, \cdots$ and $S_1, S_2, \cdots$ begin with the same reward then the sequences should be preference ordered the same way as the sequences $R_2, R_3, \cdots$ and $S_2, S_3, \cdots$ [1], [30]. In shortest route computation, if two journeys $e1, e2, e3, \cdots$ and $f1, f2, f3 \cdots$ start at the same node, then the preference order of the journeys should not change for another start time. In the network shown in Figure 5(b) the shortest path for all start times is N1-N2-N4-N5. Table I shows the tabulation of paths for different start times and illustrate that in the case of the stationary network shown in Figure 5(b), the preference order does not change with time; neither does the arrival time order depend on the start times. In the network shown in Figure 5, the shortest path for a start time $t = 1$ is N1-N2-N4-N5 which takes a travel time of 3 units. However, when the start time is moved to $t = 2$, the shortest path changes to N1-N3-N4-N5 with a travel time of 4 units whereas the route N1-N2-N4-N5 takes 7 time units. Here a change in start time leads to a change in preference order, which is a display of non-stationarity. The lack of stationarity eliminates the possibility of using dynamic programming as a design technique. In cases of single start time, in a stationary network, ranking of alternative nodes by arrival time is independent of time. In a non-stationary nerwork, arrival times at any node are not strictly ordered by the start time. In the case of the non-stationary network shown in Figure 5(c),

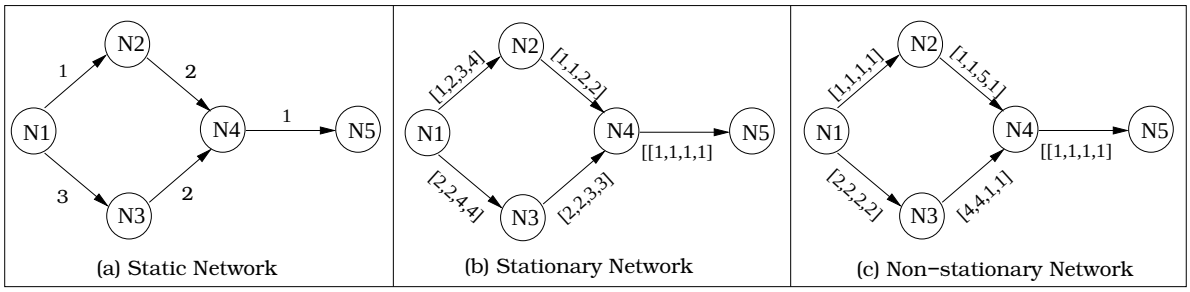

Fig. 5. Stationary & Non-stationary Graphs

TABLE I

TABULATION OF ROUTES AND TRAVEL TIMES FOR FIGURE 5

	Stationary Network (Fig 5(b))		Non-stationary Network (Fig 5(c)		
Start Time	Travel Time	Optimal Route	Travel Times		Optimal Route
			N1-N2-N4-N5	N1-N3-N4-N5	
1	3	N1-N2-N4-N5	3	7	N1-N2-N4-N5
2	5	N1-N2-N4-N5	7	4	N1-N3-N4-N5
3	6	N1-N2-N4-N5	3	4	N1-N2-N4-N5
4	7	N1-N2-N4-N5	3	4	N1-N2-N4-N5

the route preference changes with start time. The shortest path algorithms show significantly different properties based on their formulations. For example, shortest path computation for a fixed start time might display optimal prefix property under the assumption of FIFO travel times. Computation of best start time shortest paths in non-stationary networks with non-FIFO travel times might not always display optimal prefix property, ruling out popular design techniques such as greedy and A* based algorithms. Based on these characteristics, algorithms to compute shortest paths can be categorized as shown in Figure 6.

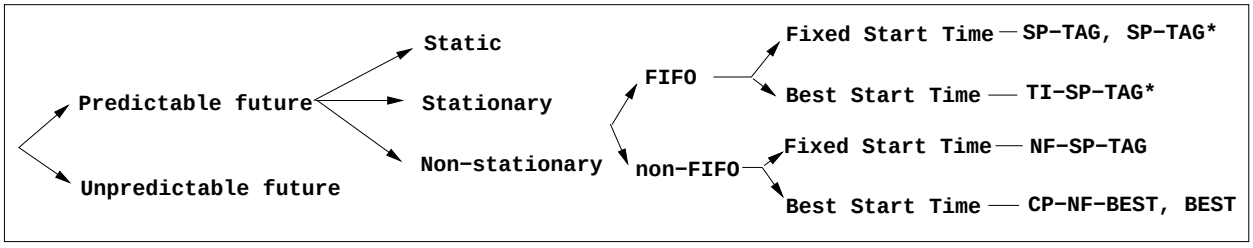

Fig. 6. Classification of Shortest Path Algorithms

C. Spatial Aspects of TAG

Time aggregated graph and the routing algorithms do have spatial components which set them apart from corresponding notions in pure graph theory. The routing algorithms presented in the paper handle concepts such as multi-lanes and turn restrictions which are spatio-temporal notions. In addition, based on the variations in edge parameter values a network may be classified as a FIFO or a non-FIFO network. This classification proves to have a significant impact in algorithm design choices and in gaining insights into the computational structure of routing problems.

Though these concepts might appear non-inituitive in graph theory tradition, they are critical for applications listed in Section I-A.

III. Shortest Path Algorithms for Fixed Start Time

In spatio-temporal networks, the shortest path and its traversal time are dependent on the start time at the source node. When the start time is fixed, the shortest paths result in earliest arrival at the destination node. In this section we outline the challenges encountered in algorithm design and the approaches to address them in stationary and non-stationary graphs. The algorithms use the time aggregated graph to represent time-dependent graphs.

A. Challenges

This result enables us to use a greedy approach to compute the shortest path.

(1)**Selecting earliest available time as the traversal time of an edge might not ensure correctness (optimality) of the shortest path:** If the travel times follow a non-FIFO variation, a selecting the edge at the earliest available time instant might not guarantee optimality. For example, consider computing the shortest path from node $N1$ to node $N5$ in Figure 8 for a start time $t = 1$. Traversing edge $N4 - N5$ as soon as $N4$ is reached (at $t = 3$) would result in a sub-optimal solution. Waiting at node $N4$ for a time unit and starting from node $N4$ at $t = 4$ would result in a total travel time of 5 units compared with 7 units if edge $N4 - N5$ was traversed at $t = 3$.

(2)**Termination of an algorithm is not guaranteed if there is a non-negative cycle over time:** If the graph has an infinite positive cycle and the travel times do not display stationary property, an optimal path finding algorithm might not terminate since it will continue searching for a shortest path indefinitely.

In the formulation of the algorithms presented here, we assume finite time windows.

(3)**Even in FIFO networks, not all shortest paths for a given start time show optimal prefix property:** Although a popular choice in most optimization problems, the application of a greedy strategy in the shortest path computation in a time-aggregated graph poses a challenge. Not all shortest paths display the optimal sub-structure even in a FIFO network, as illustrated by Figure 7. For the sake of simplicity, the travel times are constant in this example. It can be seen that a shortest path (N1-N3-N4-N5) from N1 to N5 for the start time $t = 1$, which takes 5

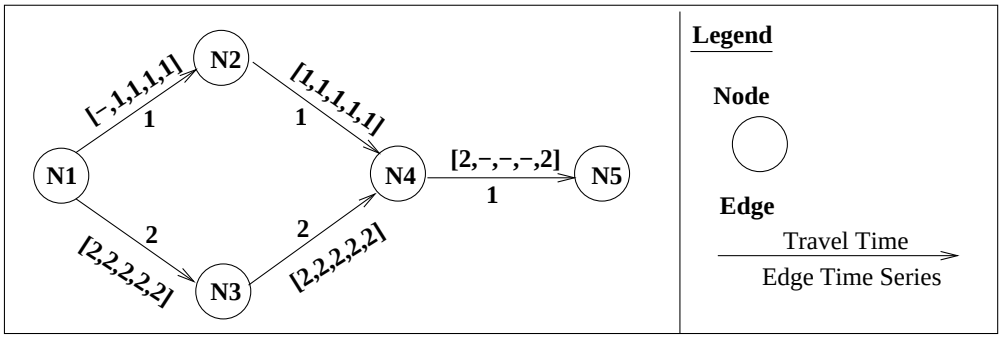

Fig. 7. Optimal Sub-structure of Shortest Paths

time units, does not display the optimal prefix property. The path from N1 to N4 following the above path is not optimal (the shortest path being N1-N2-N4).

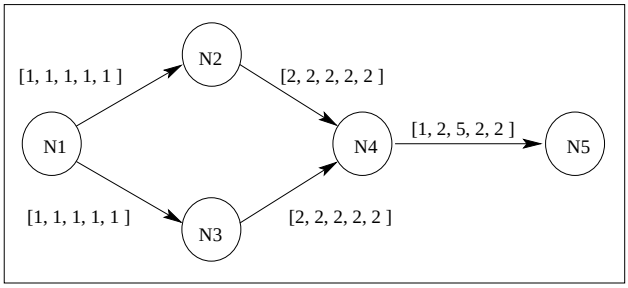

Fig. 8. Illustration of non-stationary Paths

B. Shortest Path Algorithms for Fixed Start Time

This section describes formulations of algorithms for a fixed start time in FIFO and non-FIFO networks. The FIFO algorithms namely SP-TAG and SP-TAG* employ earliest edge availability criteria and an A* search based on an admissible and monotone heuristic, respectively. The shortest path algorithm for non-FIFO network exploits a network transformation and employs an earliest arrival time strategy.

B.1. Case of Non-stationary FIFO Networks

This section describes the SP-TAG and SP-TAG* algorithms for shortest path computation in FIFO networks for a fixed start time.

SP-TAG Algorithm

TABLE II
TRACE OF THE SP-TAG ALGORITHM FOR THE NETWORK SHOWN IN FIGURE 7

Iteration	N1	N2	N3	N4	N5
1	1 (closed)	∞	∞	∞	∞
2	1	3 (closed)	3	∞	∞
3	1	3	3 (closed)	3	∞
4	1	3	3	4 (closed)	6
5	1	3	3	4	7 (closed)

Although paths that do not display optimal sub-structure could exist, it can be proved that there is at least one optimal path which satisfies the optimal sub-structure property.

Lemma 1: If there is an optimal route from source to destination, then there is at least one optimal route from source to destination that shows optimal sub-structure.
Due to space constraints, proofs are provided in Appendix II.
This result enables us to use an earliest availability criteria in selecting the edge traversal time in shortest path computation.

The SP-TAG algorithm [12], uses an earliest available traversal strategy to find the shortest path for a fixed start time. Every node has a cost associated with it which represents the travel time to reach the node from the source node. The algorithm picks the node with the least cost and updates the costs of its adjacent nodes. While finding the adjacent nodes, each edge is selected at its earliest available time instant (min_t operation in the algorithm description). A trace of the algorithm is given in Table II. The table entries are the costs associated with each node (representing the arrival times at the node) at each iteration. The node marked as "closed" is the node with the minimum cost selected for expansion. The travel times are assumed to follow the FIFO property. The pseudocode for the algorithm is provided in Appendix I.

Lemma 2: The SP-TAG algorithm is correct, assuming finite windows.
Due to space constraints, proofs are provided in Appendix II.

Lemma 3: The time complexity of the SP-TAG algorithm is $O(m(\log T + \log n))$ where T is the number of time instants, n is the number of nodes, and m is the number of edges in the time aggregated graph.

Due to space constraints, proofs are provided in Appendix II.

The SP-TAG algorithm is faster than the algorithm based on the time expanded graph if $m \log n < mT$. In other words, the SP-TAG algorithm is faster if $\log n < T$.

A* based Shortest Path Algorithm for a Fixed Start Time (SP-TAG* Algorithm)

The A* formulation of the shortest path algorithm [15] for a fixed start time discussed in this section finds an optimal solution for stationary travel times. The main challenge in formulating an A* search is the design of an admissible and monotone heuristic. We present the heuristic function used in the formulation and we prove the properties of the heuristic function which guarantee the optimality of the solution (admissibility) and the optimality of the search (monotonicity).

1) Proposed Heuristic Function: The evaluation function $f(n)$ of a node n is formulated as $f(n) = g(n) + h(n)$.

$g(n)$ is the actual cost to reach node n from the start node s, that is the time taken to reach the current node from the start node. $h(n)$ is the estimated cost from node n to destination node d. We propose $h(n)$ to be the shortest path travel time from node n to destination node d, computed based on the least travel time on each edge.

$h(n) = Min_{t=1,2,\cdots,T} d_{ij}(t), \forall ij \in E$

Lemma 4: The heuristic function $h(n)$ is admissible.

Due to space constraints, proofs are provided in Appendix II.

Lemma 5: The heuristic function $h(n)$ is monotone.

Due to space constraints, proofs are provided in Appendix II.

The pseudocode for the algorithm is provided in Appendix I. A trace of the algorithm is given in Table III. The table entries are the costs associated with each node (representing the arrival times at the node) at each iteration. The node marked as "X" is the node with the minimum cost selected for expansion. The travel times are assumed to follow the stationary property.

Lemma 6: SP-TAG* is correct, assuming finite time window.

Due to space constraints, proofs are provided in Appendix II.

Lemma 7: The time complexity of the SP-TAG* algorithm is $O(m(\log T + \log n))$, where T is the number of time instants, n is the number of nodes, and m is the number of edges in the time aggregated graph.

TABLE III

TRACE OF THE SP-TAG* ALGORITHM FOR THE NETWORK SHOWN IN FIGURE 7

Iteration	**N1**			**N2**			**N3**			**N4**			**N5**		
	g()	**h()**	**f()**	**g()**	**h()**	**f()**	**g()**	**h()**	**f()**	**g()**	**h()**	**f()**	**g()**	**h()**	**f()**
1	1	4	5 (**X**)	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	1	4	5	3	3	6 (**X**)	3	4	7	∞	∞	∞	∞	∞	∞
3	1	4	5	3	3	6	3	4	7	4	2	6 (**X**)	∞	∞	∞
4	1	4	5	3	3	6	3	4	7	4	2	6	6	2	8

Due to space constraints, proofs are provided in Appendix II.

B.2. General Case of Non-stationary, Non-FIFO Travel Times If the travel times do not exhibit the FIFO property, it is not guaranteed that an early start at any node ensures an early arrival at any subsequent node. There would be cases where postponing the start at an intermediate node (by waiting) might lead to a reduction in the total travel time. This is illustrated in Figure 9.

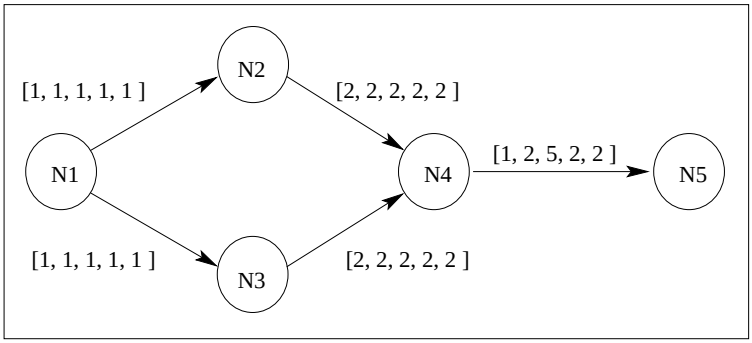

Fig. 9. Effect of Waits on Travel Time in non-stationary networks

Selecting the departure time at a node is done by choosing the earliest availability of the edge, since in some cases, a wait at an intermediate node can lead to a decrease in the total travel time. For example, in Figure 9 a greedy selection of the departure time at node N4 would lead to an arrival at node N5 at $t = 8$, as shown in Figure 9 (ii), resulting in a total travel time of 7 units, which is clearly a non-optimal solution. An optimal solution would be to start at node N1 at $t = 1$ at node $N1$, reach node N2 at $t = 2$, node N4 at $t = 3$, wait at node N4 for 1 time unit, leading to an arrival at node N3 at $t = 6$, and hence a decrease in total travel time.

In this section, we propose an algorithm that computes a shortest path in a non-stationary network. The key idea behind the algorithm is the insight that by formulating the problem in

terms of arrival times at nodes rather than the earliest departure times, we can cast the problem in a transformed problem space, where optimal substructure is valid.

Arrival Time Series Transformation (ATST): The edge weight series in a time aggregated network represents the travel times at various instants. The time series on an edge ij in the transformed network would indicate the arrival time $a_{ij}(t)$at node j for each departure time t at node i, which would be the sum of departure time t and the travel time $\sigma ij(t)$ ($a_{ij}(t) = t+\sigma ij(t)$). Figure 10 illustrates the Arrival Time Series Transformation (ATST). The first figure shows a

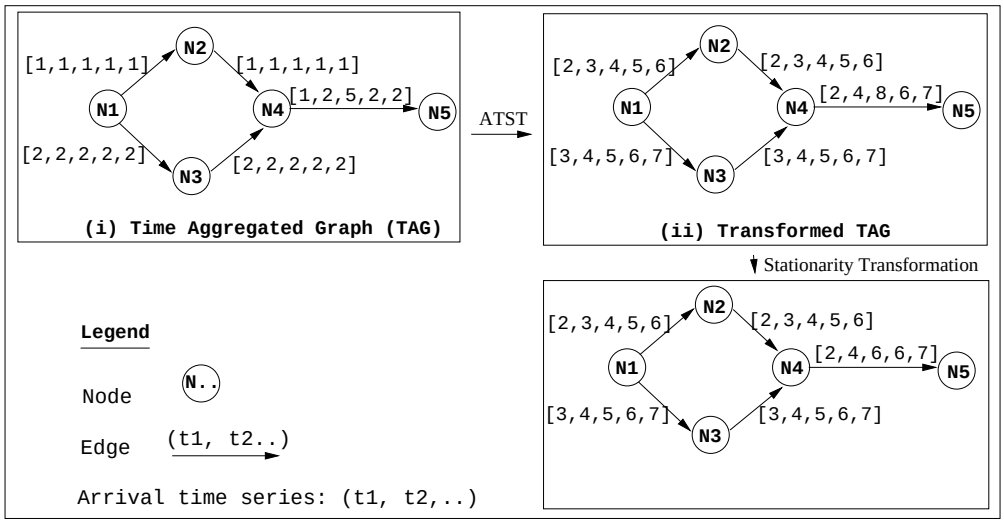

Fig. 10. Arrival Time & Earliest Arrival Time Transformations

time aggregated graph where the edge weights represent the travel time series. The second figure shows the transformed TAG (ATS_TAG) where the edge weights are modified to represent the arrival times at the end node of each edge. The algorithm proposed in this section is based on the ATS_TAG representation. The third figure shows a transformed TAG where the arrival time series is modified to a monotonic series, which gives the Earliest Arrival Time at the end node for each start time, which is stationary, but whose apriori computation is expensive. However, only a subset of this information is relevant when the start time is fixed. Computing the relevant subset on the fly is computationally cheaper than precomputing the entire graph that results from stationarity transformation. Hence the propsed algorithm computes the minimum of the series starting from the arrival time at each node.

The algorithm called the NF-SP-TAG computes the shortest path from a given source node to a destination for a given start time, for non-stationary networks, using the ATS_TAG representation. In this representation, the edge weights represent the arrival times. Every node has a cost associated with it which represents the arrival time at the node. The algorithm picks the node with the least cost and updates the costs of its adjacent nodes. While computing the costs of the adjacent nodes, the algorithm selects the earliest arrival time that is greater than the departure time (min_arrival operation in the algorithm) at the start node of the edge. The cost of the node is the arrival time at the node. Suppose the cost at the node is $c[u]$ and v is an adjacent node. The edge uv in ATS_TAG has a time series a_{uv} that indicates the arrival times at v. for every start time t at u. The update on the cost of v is done as $c[v] = c[u] + min_{t \geq c[u]} a_{uv}[t]$. A trace

Algorithm 1 Shortest Path Algorithm for Non-Stationary Non-FIFO case (NF-SP-TAG)

Input:
1) $G(N, E)$: a graph G with a set of nodes N and a set of edges E;
Each node $n \in N$ has a property:
Node Presence Time Series : series of positive integers;
Each edge $e \in E$ has two properties:
Edge Presence Time Series,
Arrival_time series : series of positive integers;
$a_{u,v}(t)$ - arrival time at v for a start time t at u.
2) s: Source node, $s \subseteq N$; 3) d: Destination node, $d \subseteq N$;
4) t_{start}: Start Time;
Output: Shortest Route from s to d for t_{start}
Method:

```
c[s] = t_start; ∀v ≠ s, c[v] = ∞;
// c[u] is the cost at the node u.
Insert s in priority queue Q.
while Q is not empty do {
   u = extract_min(Q);
   for each node v adjacent to u do {
      t = min_arrival((u, v), c[u]);
      if t + σ_{u,v}(t) < c[v] {
         c[v] = t + σ_{u,v}(t); parent[v] = u;
         if v is not in Q, insert v in Q;
      }
   update Q;
   }
}
}
Output the route from s to d.
```

of the algorithm is given in Table IV. The table entries are the costs associated with each node (representing the arrival times at the node) at each iteration. The node marked as "closed" is the node with the minimum cost selected for expansion.

TABLE IV
TRACE OF THE NF-SP-TAG ALGORITHM FOR THE NETWORK SHOWN IN FIGURE 10

Iteration	N1	N2	N3	N4	N5
1	1 (closed)	∞	∞	∞	∞
2	1	2 (closed)	3	∞	∞
3	1	2	3 (closed)	3	∞
4	1	2	3	3 (closed)	∞
5	1	2	3	3	6

Lemma 8: The NF-SP-TAG algorithm is correct assuming finite time windows.
Due to space constraints, proofs are provided in Appendix II.

To transform the problem space so that a correct greedy algorithm can be formulated, the travel times are added to the start times to compute the arrival times at the end node. These arrival times are the edge costs in T_TAG and the greedy algorithm selects the minimum of the arrival times. This method would give correct answers as long as the edge costs are positive and the parameters display the additive property.

Lemma 9: The time complexity of the NF-SP-TAG algorithm is $O(m(T + \log n))$, where T is the number of time instants, n is the number of nodes, and m is the number of edges in the time aggregated graph.
Due to space constraints, proofs are provided in Appendix II.

C. Experimental Analysis for Fixed Start Time Shortest Path Algorithms

Here we provide the experimental analysis of the the SP-TAG, SP-TAG* and NF-SP-TAG algorithms. The purpose of the performance evaluation of the algorithm is to compare the run-times with algorithms based on a time-expanded graph.

1) Experiment Design: Figure 11 illustrates the experiment design to compare the performance of the proposed algorithm and the algorithm based on a time expanded network. Time expanded graphs make copies of the original network for every time instant under consideration. The model used for the proposed algorithm is time-aggregated graphs. In our experiments the following were selected as the independent parameters: 1) network size represented by number of nodes; and

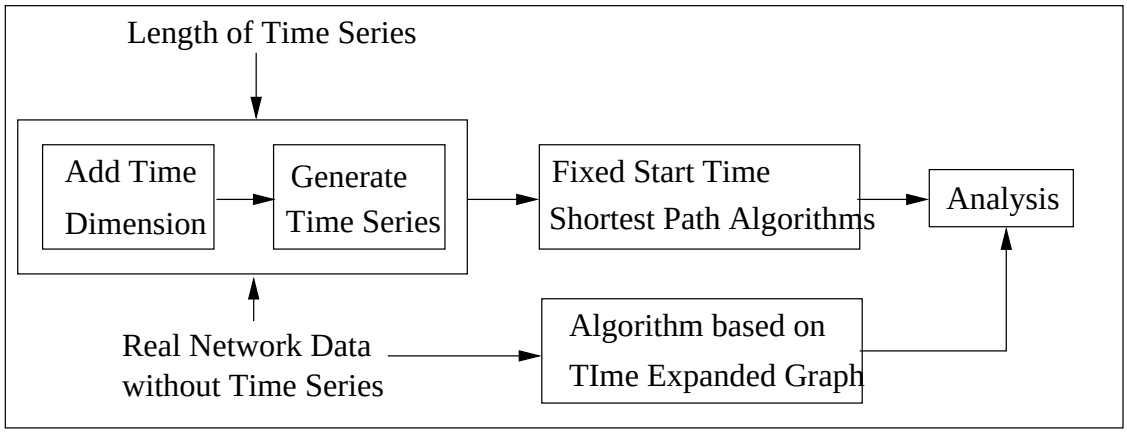

Fig. 11. Experiment Design

2)the length of the time interval in terms of number of time instants. The data sets have two main components: (1) the network data that consists of the graph structure and (2) the travel time series. The networks chosen are road maps from the Minneapolis downtown area with radii of .5 mile, 1 mile, 2 miles and 3miles. This is appended with travel time series of various lengths. The travel time series were synthetically generated. This data was fed to both a time expanded graph generator, which generates the expanded graph encoding the travel time information. An algorithm for for computing the shortest path for a given start time was run on this graph. The algorithms were was run on the same dataset and the results were compared.

The experiments were conducted on a SUN Solaris workstation with 1.77GHz CPU, 1GB RAM and UNIX operating system. Each experimental result reported in the following sections is the average over 10 experiment runs with networks generated using the same input parameters, but with different destination nodes.

2) Experimental Results and Anlaysis: We wanted to answer four questions: (1) How does the network size affect the storage cost of TAG? (2) How does the network size (number of nodes, number of edges) affect the performance of the algorithms? (3) How does the length of the time series affect the performance of the algorithms? (4) How does the average node degree of the network affect the performance?, and (4) How do the the two representations, time expanded graph and time aggregated graph, compare with respect to algorithm performance?

a) Question 1: How does the network size affect the storage cost of TAG?

: We evaluated the storage cost of TAG in comparison with time expanded graph. The experiment

was conducted for the four datasets listed in table 13. Figure 12 shows that TAG is less memory expensive than time expanded graph.

b) Question 2: How does the network size affect the performance of the algorithms?

: The purpose of the first experiment was to evaluate how the network size in terms of the number of nodes affects the performance of the algorithms. We fixed the length of the travel time series, and varied the network size to observe the run times of both, FIFO fixed start time algorithms (SP-TAG, SP-TAG*) and the non-FIFO algorithm (NF-SP-TAG algorithm), and time-expanded graph based algorithms.

The experiment was done with four datasets that represent the road maps from the Minneapolis downtown area of .5 mile, 1 mile, 2 mile and 3mile radius. The length of the time series was fixed at 240. The number of nodes and edges in these datasets are provided in Table 13. Figure 14

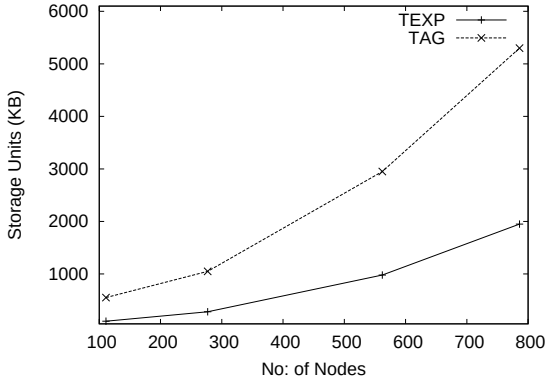

Fig. 12. Storage Cost Comparison: TAG Vs. TEG

Dataset	**Radius**	No: of Nodes	**No: of Edges**
1	0.5 mile	111	287
2	1 mile	277	674
3	2 miles	562	1443
4	3 miles	786	2106

Fig. 13. Description of Datasets

shows the run-time of the fixed start time algorithm based on the time aggregated graph and the performance of the algorithm based on the time expanded graph with the A* based algorithm displaying a faster runtime among the TAG algorithms. As Figure 15 illustrates, NF-SP-TAG algorithm runs faster than the time expanded graph algorithm.

c) Question 3: How does the length of the time series affect the performance of the algorithms?

: In the second experiment, we evaluated how the number of time instants affects the performance of the algorithms. We fixed the network size, and varied the length of the time series to observe the run-time. The number of time instants was varied from 120 to 480 and the network size parameters were fixed at 562 nodes and 1443 edges. As seen in Figure 16, the SP-TAG algorithms

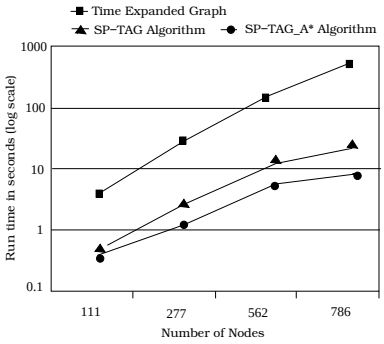

Fig. 14. SP-TAG Algorithms: Run-time With Respect to Network Size

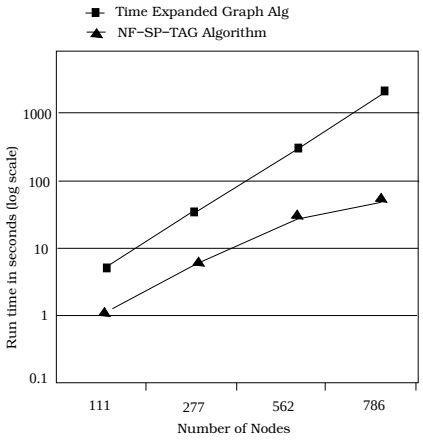

Fig. 15. NF-SP-TAG Algorithms: Run-time With Respect to Network Size

perform better, with the SP-TAG* being the fastest. On a non-FIFO network, NF-SP-TAG algorithm performs better.

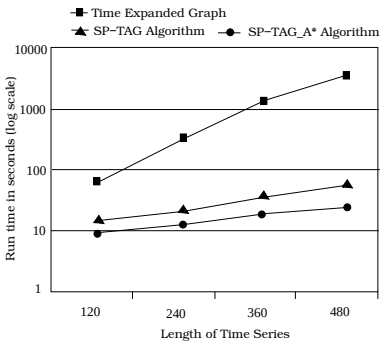

Fig. 16. SP-TAG: Run-time With Respect to Length of Time Series

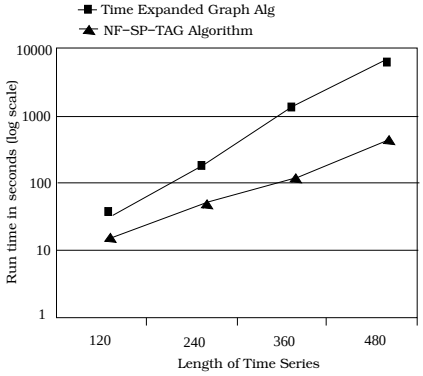

Fig. 17. NF-SP-TAG Algorithm: Run-time With Respect to Length of Time Series

d) Question 4: How does the edge/node ratio of the network affect the performance of the algorithms?

: In the third experiment, we evaluated how the edge/node ratio of the network affects the performance of the algorithms. We fixed the network size, and the length of the time series and varied the average degree of the network to observe the run-time. The edge/node ratio was varied from 2 to 6 and the network parameter was fixed at 1000 nodes and the number of time instants was fixed at 240. The networks were generated using SP-RAND network generator [16]. As

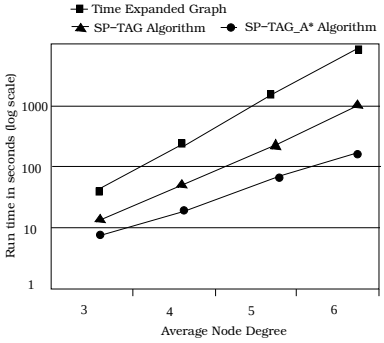

Fig. 18. SP-TAG Algorithm: Run-time With Respect to Average Node Degree

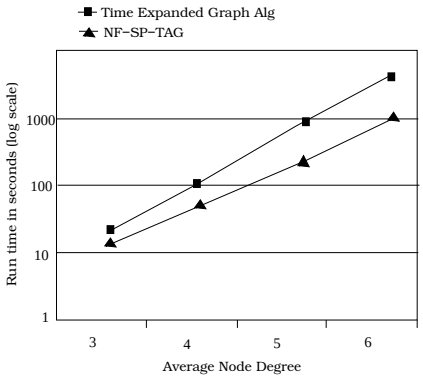

Fig. 19. NF-SP-TAG Algorithm: Run-time With Respect to Average Node Degree

seen in Figure 18, in FIFO networks, the SP-TAG algorithm performs better, with the A* based algorithm displaying a faster runtime among the TAG algorithms. The A* framework based on an admissible, monotone heuristic provides a better focus for the search than the greedy version. As illustrated by Figure 19 TAG based NF-SP-TAG algorithm performs better than time expanded based algorithm.

e) 4: How do the the two representations, time expanded graph and time aggregated graph, compare with respect to algorithm performance?

: Based on the results of Experiments (1),(2), and (3), it can be seen that algorithms based on the time aggregated graph perform better than those based on the time expanded graph.

IV. Best Start Time Shortest Paths

The time dependency of network parameters affects the connectivity and the shortest paths between nodes in a spatial network. As illustrated in Figure 20, the travel time from node N1 to node N3 changes with the start time. If the travel starts at $t = 1$, the commute time would be 6 units. A journey that starts at $t = 1$ reaches N2 at $t = 2$ and waits at N2 until edge N2-N3 becomes available at $t = 5$, thus taking a total travel time of 6 units to reach node N3. The travel on the same route would take 4 units if the start time is moved to $t = 4$. This shows that the shortest paths in a time-dependent network vary with time, which adds an interesting dimension to shortest path computation. A path that takes the smallest travel time for a source-destination traversal over the entire time horizon (called 'Best Start Time shortest Path') can be computed.

This is significant since it suggests that it is possible to reduce the travel time for the same source-destination pair if the travel starts at the ”right” time instant.

The formulation of algorithms to compute the paths that take the least commute time becomes non-trivial since most of the techniques that are used in static networks might not be applicable in dynamic scenarios. Since the network changes in its parameter values and the topology, meeting the requirements of efficiency and correctness can pose challenges. The potential waits

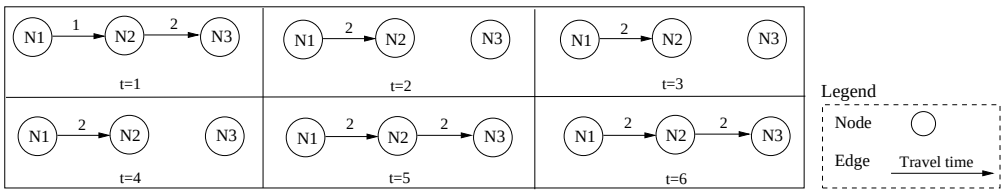

Fig. 20. Network at various instants

at intermediate nodes can increase the total journey time even if an initial part of the path turns out to be optimal. Figure 20 shows a spatial network that changes with time. The figure shows the snapshots of the network at various instants of time, and the edges are marked with the travel times. It is significant to note that the prefix journeys of the best start time shortest path journey are not always optimal since some optimal prefix journeys can lead to longer waits at intermediate nodes. The best start time for a journey from node N1 to Node N3 is $t = 4$, which takes 4 time units. The optimal path from N1 to N3 that starts at $t = 4$ is not optimal for the intermediate node N2. The best start time for a path from N1 to N2 is $t = 1$, which proves to be sub-optimal for a journey from N1 to N3. The lack of an optimal substructure in the best start time shortest paths rules out the possibility of using a greedy strategy in the algorithm design.

This section presents algorithms to compute shortest paths for the best start time and consequently the least commute time paths for both FIFO and non-FIFO graphs (TI-SP-TAG*, CP-ATST-BEST, and BEST algorithms respectively). The BEST algorithm for non-stationary networks uses a node cost time series instead of a scalar node cost. The entries in the time series are updated when a path of smaller cost is found. The algorithm iterates until every entry reaches a minimum value and hence does not depend on the greedy choice property. We also propose a logically concurrent version of NF-SP-TAG algorithm (CP-ATST-BEST)for the non-FIFO networks and compare its performance with the BEST algorithm which uses a

label-correcting strategy [5].

The algorithm to find the best start time path in a stationary graph essentially iterates the SP-TAG* algorithm, described in Section III-B, for every time instant. Since the edge costs follow the stationary property, the path computed for each step is correct and the start time instant that results in the least travel time would be the best start time. One key step in this algorithm is the computation of the estimated cost at each node and the algorithm tries to reuse computed costs from the previous iteration to reduce computational cost. Due to space constraints, proofs for the lemmas as provided in Appendix III.

A. Best Start Time Computation in non-FIFO Networks

This section describes two algorithms namely BEST and CP-ATST-BEST to compute the best start time shortest path for non-FIFO networks. While BEST algorithm makes use of label correcting strategy, CP-ATST-BEST exploits the arrival time series transformation and employs a logically concurrent execution of multiple runs of NF-SP-TAG algorithm, one for each start time.

A.1. BEst Start Time Shortest Path (BEST) Algorithm using Label Correcting Approach While computing the best start time, each node needs to keep track of the travel times to the destination for every start time instant. The proposed algorithm attributes each node with a time series, with i^{th} entry representing the current, least travel time to the destination node for the start time t_i. Due to the lack of optimality of prefix paths and lack of ordering of nodes based on the costs (ie. travel times), nodes cannot be selected and "closed" based on a minimum scalar cost. The algorithm uses an iterative, label correcting approach [3] and each entry in a node time series is modified according to the following condition.

$$C_u[t] = minimum\{C_u[t], \sigma_{uv}(t) + C_v[t + \sigma_{uv}(t)]\} \text{ where,} \quad \textbf{(1)}$$

$uv \in E$

$C_u[t]$ - Travel time from $u \in N$ to the destination for the start time t.

$\sigma_{uv}(t)$ - Travel time of the edge uv at time t.

The algorithm maintains a list of all nodes that change its cost according to the condition and terminates when there is no further improvement indicated by an empty list. Though the list can be implemented using several data structures, studies on static networks [3], [36] have shown that the Two_Q implementation [26] of label correcting algorithms performs the best on road

TABLE V
TRACE OF THE BEST ALGORITHM FOR THE NETWORK SHOWN IN FIGURE 21

Iteration	N1	N2	N3	N4	Queue
1	$\infty \cdots \infty$	$\infty \cdots \infty$	$\infty \cdots \infty$	$[0,0,0,0,0]$	$N1$
2	$\infty \cdots \infty$	$[1,1,2,2,1]$	$[4,4,2,4,3]$	$[0,0,0,0,0]$	$N2, N3$
3	$\infty \cdots \infty$	$[1,1,2,2,1]$	$[2,3,2,4,3]$	$[0,0,0,0,0]$	$N3$
4	$[4,3,3,2,3]$	$[1,1,2,2,1]$	$[2,3,2,4,3]$	$[0,0,0,0,0]$	$N1$
5	$[4,3,3,2,3]$	$[1,1,2,2,1]$	$[2,3,2,4,3]$	$[0,0,0,0,0]$	–

networks.

The search starts at the destination node and proceeds to update the remaining nodes, finally finding the best start time shortest paths from all nodes to the destination. Pseudocode is provided in Appendix III. Figure 21 illustrates the trace of the algorithm on a small network. In this example, the destination node is the node N4. The node cost series C_4 is initalized to $[0,0,0,0,0]$ and the cost series $C_i, i = 1,2,3$ are initialized to $[\infty,\infty,\infty,\infty,\infty]$. The nodes that have N4 in their adjacency lists (that is, all nodes N_i such that $N_iN4 \in E$), N2 and N3 are relaxed according to condition (1). These nodes are added to the queue since there is a change in their cost series. The steps continue until the queue is empty, indicating that there is no further cost improvement at any of the nodes. At every iteration, the node that contributes to a cost improvement is stored in a descendant array to facilitate the trace of the shortest paths when the algorithm terminates. At the termination, the cost time series has the travel times for every start time $t = 1,2 \cdots T$. For example, the cost time series of node N1 shows that the travel times from N1 to N4 for start times $t = 1$ is 4 time units, while the best start time at this node is $t = 4$, which results in a travel time of 2 time units and a best start time shortest path N1-N2-N4. N1-N2 takes 1 time unit at $t = 4$, reaches N2 at $t = 5$ and continues on N2-N4 at $t = 5$, reaching N4 at $t = 6$, taking a total travel time of 2 time units. A more detailed trace is shown in Table V.

Lemma 10: The algorithm terminates and computes the best start time paths from every node to the destination.

Lemma 11: The computational complexity of the BEST algorithm is $O(n^2mT)$, where n is the number of nodes, m is the number of edges and T is the length of the time series.

A.2. Best Start Time Algorithm Using ATST (CP-ATST-BEST Algorithm) This section describes a best start time algorithm where the benefits of arrivel time series transformation are utilized. The basic idea behind the algorithm is iterating the NF-SP-TAG algorithm, but with logical

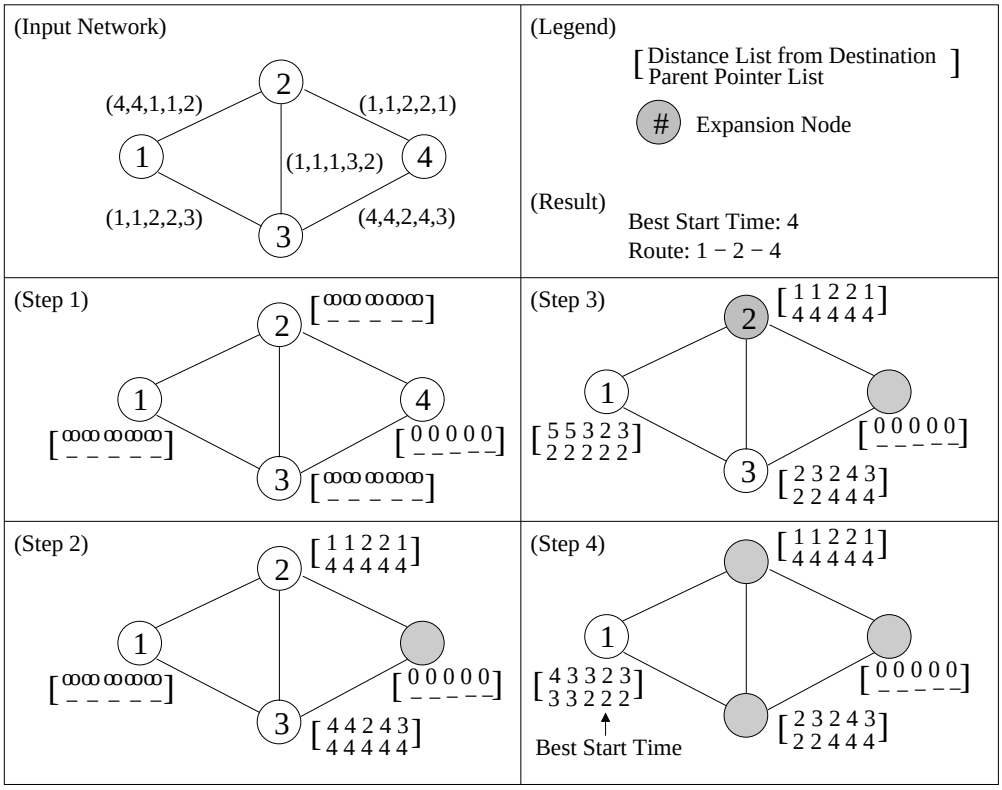

Fig. 21. Trace of the BEST Algorithm

concurrency. This concurrency is achieved by keeping an open list of nodes such that every node has a copy for every start time. While selecting the node for expansion, the copy of the node with the minimum cost is selected. The pseudocode is provided in Algorithm 2.

Lemma 12: CP-ATST-BEST algorithm is correct, assuming finite time windows.

Lemma 13: The compuational complexity of CP-ATST-BEST is $O(mT(T+\log n))$ where m is the number of edges, n is the number of nodes, and T is the length of time period.

B. Time Iterative SP-TAG (TI_SP-TAG*) Algorithm for FIFO Networks*

Under the assumption of stationary travel times, the least duration journey algorithm can be for mulated as an iterative formulation of SP-TAG* search algorithm (Time Iterative SP-TAG*). The SP-TAG* version was chosen instead of the greedy SP-TAG algorithm since it performed better in the fixed start time algorithms. A comparative experimental analysis is provided to evaluate this decision in Section IV-C.

Algorithm 2 Shortest Path (CP-ATST-BEST) Algorithm

Input:
1) $G(N, E)$: a graph G with a set of nodes N and a set of edges E;
Each node $n \in N$ has a property:
Node Presence Time Series : series of positive integers;
Each edge $e \in E$ has two properties:
Edge Presence Time Series,
Arrival_time series : series of positive integers;
$a_{u,v}(t)$ - arrival time at v for a start time t at u.
2) s: Source node, $s \subseteq N$; 3) d: Destination node, $d \subseteq N$;
Output: Shortest Route from s to d.
Method:

```
for (i=1 to T) do
   s_i = i; ∀v ≠ s, c[v] = ∞;
// c[u] is the cost at the node u.
Insert s_i in priority queue Q.
while Q is not empty do {
   u_i = extract_min(Q);
   if u = d break;
   for each node v adjacent to u do {
      t = min_arrival((u, v), c[u_i]);
      if t + σ_{u,v}(t) < c[v_i] {
         v_i = t + σ_{u,v}(t); parent[v_i] = u;
         if v_i is not in Q, insert v_i in Q;
      }
   update Q;
   }
}
}
Output the route from s to d.
```

The evaluation function $f(n)$ of a node n is formulated as $f(n) = g(n) + h(n)$. $g(n)$ is the actual cost to reach the node n from the start node s, which is the time taken to reach the current node from the start node. $h(n)$ is the estimated cost from the node n to a destination node d. We propose $h(n)$ to be the shortest path travel time from node n to the destination node d computed based the least travel time on each edge. This cost function is used to compute the shortest path for every start time and the least travel time is chosen from the shortest path travel times for all start times. The algorithm runs the A* algorithm for every start time and computes the shortest path for each start time. Since the heuristic function is admissible and monotone (See Lemma 4 and Lemma 5), the computed shortest paths are correct and the search is optimal. Due to the stationa ry property of the travel times, the best start time would be the start time that yields the least travel time among the shortest paths over all start times.

Performance Tuning: It is observed that in a stationary network, it is possible to take advantage of the results obtained from the previous steps in the iteration on time instants. When

Algorithm 3 TI_SP-TAG* Algorithm

Input:
1) $G(N, E)$: a graph G with a set of nodes N and a set of edge s E;
Each node $n \in N$ has a property:
Node Presence Time Series : series of positive integers;
Each edge $e \in E$ has two properties:
Edge Presence Time Series,
Travel_time series : series of positive integers;
$\sigma_{u,v}(t)$ - travel time of edge uv at time t.
2) s: Source node, $s \subseteq N$; 3) d: Destination node, $d \subseteq N$;
4) t_{start}: Start Time;
Output: Shortest Route from s to d for t_{start}
Method:

```
Preprocess: find the shortest path from node i to d in S_TAG.
for (i = 1 to T) {
   t_start = i;
   c[s] = t_start; ∀v ≠ s, c[v] = ∞; f[v] = ∞;        // c[u] is the arrival time at the node u.
   f[s] = SP[s]; C = Φ; S = s ;
   Insert s in priority queue Q.
   while u ≠ d {
      u = extract_min(Q);
      C = C ⋃ u; S = S − u;
      for each node v adjacent to u do {
         if(f[v] > c[v]
            c[v] = c[u] + d_uv(c[u]);
            f[v] = c[u] + d_uv(c[u]) + SP_vd(c[u]);
            if v is not in Q, insert v in Q;
         }
      update Q;
   }
}
Compute the travel time as the difference(start time, arrival time);
}
Output the route from s to d.
```

computing the shortest path for the start time t, the shortest path for the start time $t-1$ has already been computed. In a stationary network, the earliest arrival at the destination node (d) for a given start time at the source node (s) implies the least travel time for the same start time. This leads to the observation that an admissible estimate of the cost from a node i on the shortest path from node s to node d can be computed from the costs computed in the previous iteration as follows. The admissible estimate of travel time from node i to node d is $(t-1) + d_{si}(t-1) + SP_{id}(t-1+d_{si}(t-1)) - (t + d_{si}(t))$ where

$d_{si}(t)$ is the shortest travel time from s to node i for start time t,

$SP_{id}(t)$ is the shortest travel time from i to destination d for start time t.

$(t-1) + d_{si}(t-1)$ is the arrival time at node i for start time $t-1$ and hence $(t-1) + d_{si}(t-$

$1) + SP_{id}(t-1+d_{si}(t-1))$ is the earliest arrival time at the destination d for the start time $t-1$. The estimate is computed by subtracting the earliest arrival time at i for the (next) start time instant t (*(ie.)* $t + d_{si}(t)$). The fact that this is admissible can be proved as follows:
Since the network is stationary, the arrival time for start time t cannot be earlier than that for $t-1$. Or,
$t + d_{si}(t) \geq (t-1) + d_{si}(t-1)$.
$t + d_{si}(t) + SP_{id}(t + d_{si}(t)) \geq (t-1) + d_{si}(t-1) + SP_{id}(t-1+d_{si}(t-1))$
This uses the property that the arrival at the destination d for start time t cannot be earlier than that for a start at $t-1$.
Therefore, $SP_{id}(t + d_{si}(t)) \geq (t-1) + d_{si}(t-1) + SP_{id}(t-1+d_{si}(t-1)) - (t + d_{si}(t))$
and hence $(t-1) + d_{si}(t-1) + SP_{id}(t-1+d_{si}(t-1)) - (t + d_{si}(t))$ is an admissible estimate for the shortest path from an intermediate node i to d.
This estimate can be used only in the case of nodes that are a part of the shortest path at the earlier iteration. If this is not the case, we use the heuristic based on static TAG (S_TAG) as explained in Section III-B.1 to estimate the cost.

C. Experimental Analysis

In this section, the experimental analysis of the best start time algorithms, namely BEST, CP-NF-BEST, and TI-SP-TAG* algorithms are provided. The purpose of the performance evaluation of the algorithm is to compare the run-times with algorithms based on a time-expanded graph and to evaluate the performances of label correcting BEST algorithm and greedy CP-NF-BEST algorithm relative to each other.

1) Experiment Design: Figure 22 illustrates the experiment design to compare the performance of the proposed algorithm and the algorithm based on a time expanded network. Time expanded graphs make copies of the original network for every time instant under consideration. The model used for the proposed algorithm is time-aggregated graphs. In our experiments the following were selected as the independent parameters: 1) network size represented by number of nodes; and 2)the length of the time interval in terms of number of time instants. The data sets have two main components: (1) the network data that consists of the (3) average degree of the network. graph structure and (2) the travel time series. The networks chosen are road maps from the Minneapolis downtown area with radii of .5 mile, 1 mile, 2 miles and 3miles. This is appended

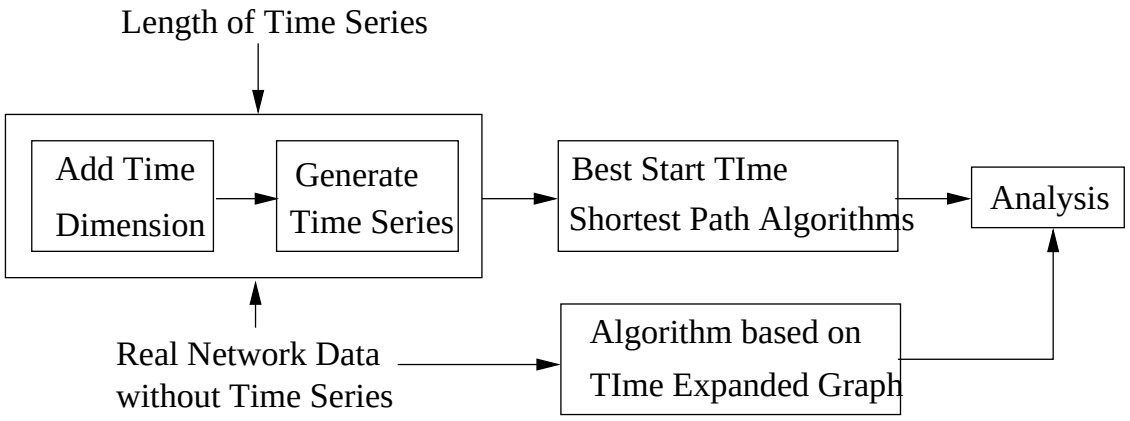

Fig. 22. Experiment Design

with travel time series of various lengths. The travel time series were synthetically generated. This data was fed to both a time expanded graph generator, which generates the expanded graph encoding the travel time information. An algorithm for for computing the shortest path for a best start time was run on this graph. The results were compared to the results from the BEST, TI-SP-TAG* and CP-NF-BEST algorithms.

The experiments were conducted on a SUN Solaris workstation with 1.77GHz CPU, 1GB RAM and UNIX operating system. Each experimental result reported in the following sections is the average over 5 experiment runs with networks generated using the same input parameters, but with different destination nodes.

2) Experimental Results and Anlaysis: We wanted to answer four questions: (1) How does the network size (number of nodes, number of edges) affect the performance of the algorithms? (2) How does the length of the time series affect the performance of the algorithms? (3) How does the network structure in terms of the edge/node ratio affect the performance? (4) How do the the two representations, time expanded graph and time aggregated graph, compare with respect to algorithm performance?

a) Experiment 1: How does the network size and time series length affect the performance of the BEST, CP-NF-BEST, and TI-SP-TAG algorithms?*

: The purpose of the first experiment was to evaluate how the network size and the time series length affect the performance of the BEST, CP-NF-BEST, and TI-SP-TAG* algorithms. To evaluate the scalability with respect to the network size, we fixed the length of the travel

time series, and varied the network size to observe the run times best start time(BEST and CP-NF-BEST) algorithms and time-expanded graph based algorithms. The experiment to study the effect of time series length was performed with a fixed network size and varying time series lengths.

The experiment was done with four datasets that represent the road maps from the Minneapolis downtown area of .5 mile, 1 mile, 2 mile and 3mile radius. The length of the time series was fixed at 240. The number of nodes and edges in these datasets are provided in Table 13. Figure 23

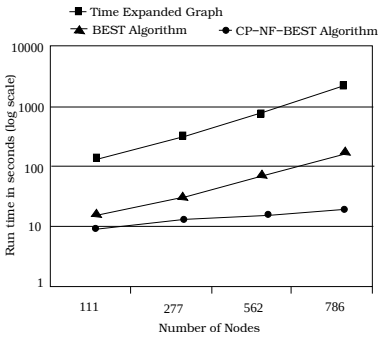

Fig. 23. BEST Algorithm: Run-time With Respect to Network Size

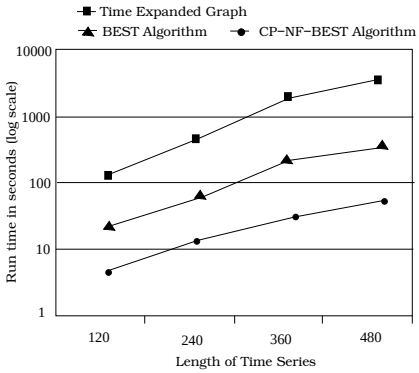

Fig. 24. BEST Algorithm: Run-time With Respect to Time Series Length

shows the run-time of the best start time algorithms based on the time aggregated graph and the performance of the algorithm based on the time expanded graph. The BEST and CP-NF-BEST algorithms run faster than the time-expanded graph based algorithm in all cases. Among the TAG based algorithms, CP-NF-BEST that uses a performance tuned iterative version of greedy NF-SP-TAG algorithm runs faster than the label correcting BEST algorithm. Since NF-SP-TAG algorithm employs a greedy approach and the expanded nodes are never reopened, it achieves a better performance compared to label correcting version where the nodes are not picked according to their costs leading to a possibility of multiple expansions of the same node.

b) Experiment 2: How does TI_SP-TAG perform with respect to the network size and the length of the time series?*

: In the second experiment, we evaluated the performance of TI_SP-TAG* algorithm. This was compared to an algorithm that runs on time expanded graph. In the evaluation with respect to series length, we fixed the network size, and varied the length of the time series to observe the run-time. The number of time instants was varied from 120 to 480 and the network size

parameters were fixed at 562 nodes and 1443 edges. In the other case, the length of the time series was held constant and the network sizes were varied. Figure 25 shows the run-time of

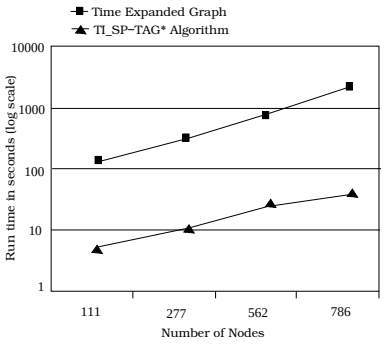

Fig. 25. TI-SP-TAG* Algorithm: Run-time With Respect to Network Size

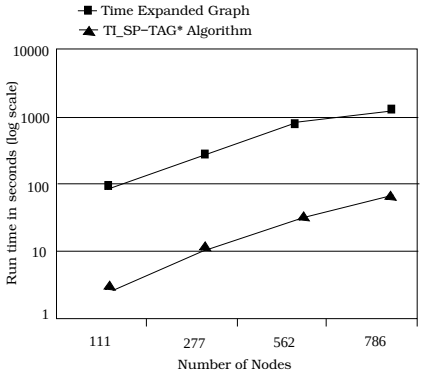

Fig. 26. TI-SP-TAG* Algorithm: Run-time With Respect to Length of Time series

the fixed start time algorithms based on the time aggregated graph and the performance of the algorithm based on the time expanded graph. As seen in Figure 26, the TI_SP-TAG* algorithm performs better, compared to the time expanded graph version. As the length of the time series increases, the number of copies of the entire network required in the case of the time expanded graph increases, resulting in a considerable increase in the size of the entire network, leading to almost exponential increases in run time.

c) Experiment 3: How does the edge/node ratio of the network affect the performance of the algorithms?

: In the third experiment, we evaluated how the edge/node ratio of the network affects the performance of the algorithms. We fixed the network size, and the length of the time series and varied the average degree of the network to observe the run-time. The edge/node ratio was varied from 2 to 6 and the network parameter was fixed at 1000 nodes and the number of time instants was fixed at 240. The networks were generated using SP-RAND network generator [16]. As seen in Figure 27, the TI_SP-TAG* algorithm performs better. Figure 28 the performance of the BEST algorithm and that of the time expanded graph algorithm.

d) Experiment 4: How does the iterative version of Greedy SP-TAG algorithm compare to the iterative version of SP-TAG algorithm (TI_SP-TAG*)?*

: In the fourth experiment, we compared the iterative version of Greedy SP-TAG algorithm to the iterative version of SP-TAG* algorithm (TI_SP-TAG*), with respect to the (i) network size

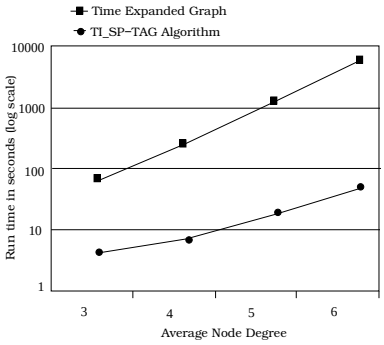

Fig. 27. TI_SP-TAG* Algorithm: Run-time With Respect To Average Degree of the Network

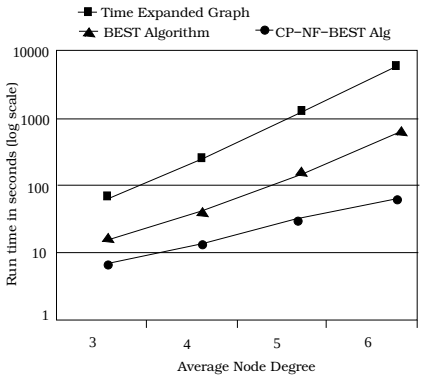

Fig. 28. BEST Algorithm: Run-time With Respect to Average Degree of the Network

and (ii) time series length. In case (i), the length of the time series was kept constant and the network size was varied, whereas in the second case the length of the time series was changed maintaining the network size constant. As seen in Figure 29 and Figure 30, the TI_SP-TAG* algorithm performs better than the iterative version of the greedy SP-TAG algorithm.

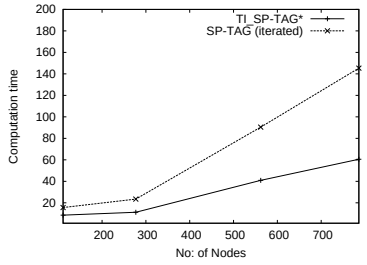

Fig. 29. Comparison of TI_SP-sTAG* with Iterated SP-TAG: Run-time With Respect To Network Size

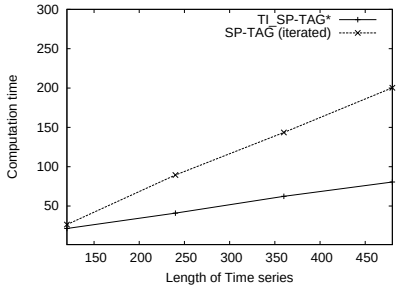

Fig. 30. Comparison of TI_SP-TAG* with Iterated SP-TAG: Run-time With Respect to Time Series Length

e) 5: How do the the two representations, time expanded graph and time aggregated graph, compare with respect to algorithm performance?

: Based on the results of Experiments (1) and (2), it can be seen that algorithms based on the time aggregated graph perform better than those based on the time expanded graph. Under the assumption of stationarity, the A* based algorithm based on an admissible, monotone heuristic shows the best performance among the three algorithms.

V. Conclusions and Future Work

Spatio-temporal networks form a key part of critical applications such as emergency planning and there is a great need for efficient in this area. The paper describes a model for spatio-temporal networks called the time aggregated graph, that uses a time series to represent time-varying attributes. We propose algorithms to compute shortest paths for a fixed start time and the best start time (Best Start Time Algorithm) and consequently the least commute time paths. These problems are challenging since common algorithm design techniques like greedy design cannot always be applied. This paper proposes shortest path algorithms for two start time choices : (1) fixed start time and (2) best start time, where the time spent in the network is minimized. These algorithms compute the shortest paths in these two contexts for both FIFO and non-FIFO travel times in non-stationary networks and provide insights into the computational structure of these problems. The formulation of these algorithms is based on a model for spatio-temporal networks called time-aggregated graphs. Figure 31 lists the algorithms presented in this paper.

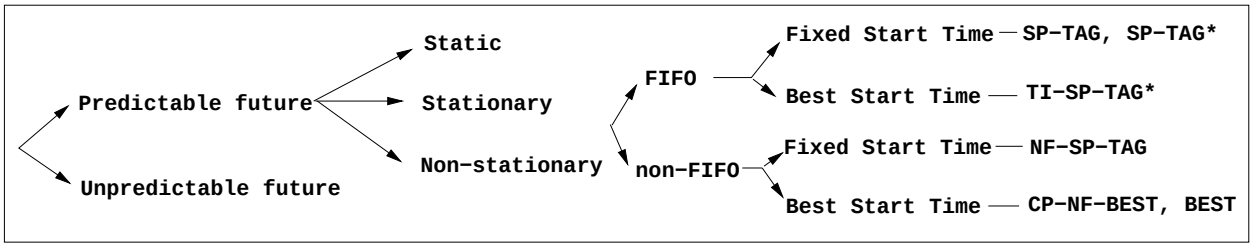

Fig. 31. Algorithms Proposed in the Paper

We also present an experimental analysis of the best start time (BEST, CP-NF-SP-TAG, TI_SP-TAG*) algorithms and the fixed start time algorithms (SP-TAG, SP-TAG*, NF-SP-TAG) experiments show that the algorithms based on time aggregated graphs significantly reduce the computational cost compared to similar algorithms based on time expanded networks.

We plan to evaluate the performance of the algorithms using real-traffic datasets shortly. We anticipate that this evaluation will give new insights into the average case run time of the algorithms. we expect to be significantly better than the worst case complexity, We would evaluate the effect of including spatial properties to achieve pruning in the algorithms. We will also explore the possibility of continuous time interpolation.

We plan to explore the computational structure of shortest path problems in the context of

various wait policies, start time flexibility, arrival time requirements, and turn minimization. We also intend to explore the possibility of modeling time-dependent multi-modal transportation systems using time aggregated graphs and formulating least travel time path algorithms for these networks. We also plan to work on developing cost functions (other than travel times) which would enable formulations to minimize fuel consumption.

The time aggregated graphs can accomodate the time-varying capacities of the road networks. The proposed algorithms need to be extended to give optimal solutions subject to the constraints of time-varying capacities. This would extend the use of the algorithms to domains such as evacuation planning in emergency management, where capacity constraints in the network pose significant challenges. We plan to incorporate the algorithms as building blocks that find the shortest paths in the CCRP evacuation planner [23]. We will also explore other graph problems in the context of time aggregated graphs.

A continuous interpolation model would be interesting where ideas from CapeCod model [19] would be very helpful. We plan to explore the possibilities of extending TAG to incorporate continuous changes in travel time.

ACKNOWLEDGMENTS

We are particularly grateful to the members of the Spatial Database Research Group at the University of Minnesota for their helpful comments and valuable suggestions. We would also like to express our thanks to Kim Koffolt for improving the readability of this paper.

This work was supported by the NSF SEI grant (grant number IIS-0431141), Department of Defense, and Minnesota Department of Transportation. The content does not necessarily reflect the position or the policy of the government and no official endorsement should be inferred.

REFERENCES

[1] D. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models*. Prentice Hall, Englewood Cliffs, NJ, 1987.
[2] I. Chabini. Discrete dynamic shortest path problems in transportation applications. *Transportation Research Record*, 1424, 1997.
[3] B.V. Cherkassky, A.V. Goldberg, and T. Radzik. Shortest Paths Algorithms: Theory and Experimental Evaluation . *Mathematical Programming*, 73:129–174, 1996.
[4] K.L. Cooke and E. Halsey. The shortest route through a network with time-dependent internodal transit times. *Journal of Math. Analy. Applications*, 14:492–498, 1966.
[5] T.H. Cormen, C.E.Leiserson, R.L.Rivest, and C.Stein. *Introduction to Algorithms (Chapter 26, Flow Networks)*. MIT Press, Cambridge, MA, USA, 2002.
[6] B. C. Dean. Algorithms for minimum-cost paths in time-dependent networks. *Networks*, 44(1):41–46, August 2004.
[7] Claudia H. Deutsch. U.P.S. Embraces High-Tech Delivery Methods. The New York Times (http://www.nyt.com/), July 12 2007.
[8] Z. Ding and R.H. Guting. Modeling temporally variable transportation networks. *Proc. 16th Intl. Conf. on Database Systems for Advanced Applications*, pages 154–168, 2004.
[9] M. Erwig. *Graphs in Spatial Databases*. PhD thesis, Fern Universität Hagen, 1994.
[10] M. Erwig and R.H. Guting. Explicit graphs in a functional model for spatial databases. *IEEE Transactions on Knowledge and Data Engineering*, 6(5):787–804, 1994.
[11] ESRI. ArcGIS Network Analyst. *http://www.esri.com/software/arcgis/extensions/*, 2006.

[12] B. George, S. Kim, and S. Shekhar. Spatio-temporal Network Databases and Routing Algorithms: A Summary of Results . *Proceedings of International Symposium on Spatial and Temporal Databases (SSTD'07)*, July 2007.
[13] B. George and S. Shekhar. Time-aggregated Graphs for Modeling Spatio-Temporal Networks - An Extended Abst ract . *Proceedings of Workshops at International Conference on Conceptual Modeling*, November 2006.
[14] B. George and S. Shekhar. Time Aggregated Graphs: A model for spatio-temporal network. *Journal on Data Semantics*, 11, December 2007.
[15] B. George and S. Shekhar. Sp-tag*: A routing algorithm in non-stationary transportation networks. *Proceedings of First International Workshop on Computational Transportation Science*, July 2008.
[16] A.V. Goldberg. Network Optimization Library. http://www.avglab.com/andrew/soft.html, 2002.
[17] T. Hamre. *Development of Semantic Spatio-temporal Data Models for Integration of Remote Sensing and in situ Data in Marine Information System*. PhD thesis, University of Bergen, Norway, 1995.
[18] P.E. Hart, N/J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions of System Science and Cybernetics*, 4(2), 1986.
[19] Y. Kanoulas, T. Du, T. Xia, and D. Zhang. Finding fastest paths on a road network with speed patterns. *Proc. of 22nd International Conference on Data Engineering (ICDE)*, 2006.
[20] D.E. Kaufman and R.L. Smith. Fastest paths in time-dependent networks for intelligent vehicle highway systems applications. *IVHS Journal*, 1(1):1–11, 1993.
[21] E. Kohler, K. Langtau, and Skutella M. Time-expanded graphs for flow-dependent transit times. *Proc. 10th Annual European Symposium on Algorithms*, pages 599–611, 2002.
[22] M. Koubarakis, T. K. Sellis, A. U. Frank, S. Grumbach, R. H. Güting, C. S. Jensen, N. A. Lorentzos, Y. Manolopoulos, E. Nardelli, B. Pernici, H.J. Schek, M. Scholl, B. Theodoulidis, and N. Tryfona, editors. *Spatio-Temporal Databases: The CHOROCHRONOS Approach*, volume 2520 of *Lecture Notes in Computer Science*. Springer, 2003.
[23] Q. Lu, B. George, and S. Shekhar. Capacity Constrained Routing Algorithms for Evacuation Planning: A Summary of Resu lts. *Proc. of 9th International Symposium on Spatial and Temporal Databases (SSTD'05)*, August 2005.
[24] Oracle. Oracle Spatial 10g, An Oracle White Paper. http://www.oracle.com/technology/products/spatial/, August 2005.
[25] A. Orda and R. Rom. Minimum weight paths in time-dependent networks. *Networks*, 21:295–319, 1991.
[26] S. Pallottino. Shortest-Path Methods: Complexity, Interrelations and New Propositions . *Networks*, 14:257–267, 1984.
[27] S. Pallottino and M. G. Scuttella. Shortest path algorithms in tranportation models: Classical and innovative aspects. *Equilibrium and Advanced transportation Modelling (Kluwer)*, pages 245–281, 1998.
[28] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison Wesley, 1984.
[29] J. Rasinmäki. Modelling spatio-temporal environmental data. In *5th AGILE Conference on Geographic Information Science*, Palma, Balearic Islands, Spain, April 2002.
[30] S. Russel and P. Norwig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, 1995.
[31] Shekhar S. and Chawla S. *Spatial Databases: Tour*. Prentice Hall, 2003.
[32] D. Sawitzki. Implicit Maximization of Flows over Time . *Technical report, University of Dortmund*, 2004.
[33] Dreyfus S.E. An appraisal of some shortest path algorithms. *Operations Research*, 17:395–412, 1969.
[34] S. Shekhar and D. Liu. CCAM: A Connectivity-Clustered Access Method for Networks and Networks Computations. *IEEE Transactions on Knowledge and Data Engineering*, 9, January 1997.
[35] S. Stephens, J. Rung, and X. Lopez. Graph data representation in oracle databese 10g: Case studies in life sciences. *IEEE Data Engineering Bulletin*, 27(4):61–66, 2004.
[36] F.B. Zhan and C.E. Noon. Shortest Paths Algorithms: An Evaluation Using Real Road Networks. *Transportation Science*, 32:65–73, 1998.

APPENDIX I
PSEUDOCODE FOR SP-TAG AND SP-TAG* ALGORITHMS

Algorithm 4 Shortest Path (SP-TAG) Algorithm

Input:
1) $G(N, E)$: a graph G with a set of nodes N and a set of edges E;
Each node $n \in N$ has a property:
Node Presence Time Series : series of positive integers;
Each edge $e \in E$ has two properties:
Edge Presence Time Series,
Travel_time series : series of positive integers;
$\sigma_{u,v}(t)$ - travel time of edge uv at time t.
2) s: Source node, $s \subseteq N$; 3) d: Destination node, $d \subseteq N$;
4) t_{start}: Start Time;
Output: Shortest Route from s to d for t_{start}
Method:

```
c[s] = t_start; ∀v ≠ s, c[v] = ∞;
// c[u] is the cost at the node u.
Insert s in priority queue Q.
while Q is not empty do {
   u = extract_min(Q);
   for each node v adjacent to u do {
      t = min_t((u, v), c[u]);
      if t + σ_{u,v}(t) < c[v] {
         c[v] = t + σ_{u,v}(t); parent[v] = u;
         if v is not in Q, insert v in Q;
      }
   update Q;
   }
}
}
Output the route from s to d.
```

Algorithm 5 A* based Shortest Path (SP-TAG*) Algorithm

Input:
1) $G(N, E)$: a graph G with a set of nodes N and a set of edges E;
Each node $n \in N$ has a property:
Node Presence Time Series : series of positive integers;
Each edge $e \in E$ has two properties:
Edge Presence Time Series,
Travel_time series : series of positive integers;
$\sigma_{u,v}(t)$ - travel time of edge uv at time t.
2) s: Source node, $s \subseteq N$; 3) d: Destination node, $d \subseteq N$;
4) t_{start}: Start Time;
Output: Shortest Route from s to d for t_{start}
Method:
Preprocess: find the shortest path from node i to d in S_{TAG}.
$c[s] = t_{start}; \forall v \neq s, c[v] = \infty; f[v] = \infty;$
// $c[u]$ is the arrival time at the node u.
$f[s] = SP[s]; C = \Phi; S = s$;
Insert s in priority queue Q.
while $u \neq d$ {
$u = extract\_min(Q);$
$C = C \bigcup u; S = S - u;$
for each node v adjacent to u do {
$if(f[v] > c[u] + d_{uv}(c[u]) + SP_{vd}(c[u])$
$c[v] = c[u] + d_{uv}(c[u]);$
$f[v] = c[u] + d_{uv}(c[u]) + SP_{vd}(c[u]);$
if v is not in Q, insert v in Q;
}
update Q;
}
}
}
Output the route from s to d.

APPENDIX II
DETAILED PROOFS FOR SECTION 3

Lemma 1: If there is an optimal route from source to destination, then there is at least one optimal route from source to destination that shows optimal sub-structure.

Proof: As Figure 7 illustrates, the failure of optimal structure of the shortest path occurs due to a potential wait at the intermediate node ($N4$), after reaching this node traversing the optimal path from $N1$ to $N5$, assuming travel time variations allow arrival times to be ordered by start times. Consider the optimal path from $N1$ to $N4$. Append this path to the path $N4 - N5$ (allowing wait at the intermediate node $N4$) from the optimal path. This would be still the shortest path from $N1$ to $N5$. Otherwise, it would contradict the optimality of the original shortest path.

Lemma 2: The SP-TAG algorithm is correct.

Proof: The proof of correctness of the algorithm which follows a greedy strategy follows the proof of correctness for Dijkstra's algorithm to find the shortest path from a sour ce node to a destination. The key difference in time aggregated graph is that each edge has a presence series. SP-TAG employs a greedy approach where it selects the earliest available time instant as the traversal time of the edge. Since waits are permitted at intermediate nodes, this admissible approach does not violate the optimality of the shortest path even while considering the time-dependence of edge presence. To prove the correctness of the algorithm, we need to show that the cost of a node, when it is closed, is the shortest path distance to the node. This can be proved by induction on the set of closed nodes (S in Figure 32). Let v be the next node to be closed. Suppose the cost of node v was last updated when node x was added to S and v is adjacent to x. When x was added to S, a shorter path to v through x was discovered. Assume that the cost of v is not the shortest path cost. This would be due to the existence of a path $s \cdots y \cdots xv$ as shown in Figure 32. Since x was closed before y, the shortest path to x is inside S by inductive hypothesis. Therefore, the length of the path from s to v through y cannot be shorter that the path $s \cdots xv$. The cost of v cannot be further

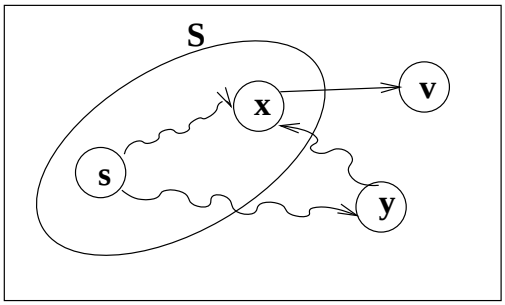

Fig. 32. Correctness of SP-TAG Algorithm

reduced by forming a path through nodes outside S. hence, the cost of the node when it is closed is the shortest path distance to the node.

Lemma 3: The time complexity of the SP-TAG algorithm is $O(m(\log T + \log n))$ where T is the number of time instants, n is the number of nodes and m is the number of edges in the time aggregated graph.

Proof: The cost model analysis assumes an adjacency list represe ntation of the graph with two significant modifications. The edge time series is stored in the sorted order. Attached to every adjacent node in the linked list are the edge time series and the travel time series.
For every node extracted from the priority queue Q, there is one edge time series look up and a priority queue update for each of its adjacent nodes. The time complexity of this step is $O(\log T + \log n)$. The asymptotic complexity of the algorithm would be
$O(\Sigma_{v \in N}[degree(v).(\log T + \log n]) = O(m(\log T + \log n))$.
The time complexity of the SP-TAG shortest path algorithm based on a time expanded network is $O(nT \log T + mT)$ [6]. Assuming a sparse graph where m is $O(n)$, $nT \log T < m \log T$. The SP-TAG algorithm is faster than the algorithm based on time expanded graph if $m \log n < mT$. In other words, the SP-TAG algorithm is faster if $\log n < T$.

Lemma 4: The heuristic function $h(n)$ is admissible.

Proof: A heuristic function $h(n)$ is admissible if it underestimates the cost from the node n to the destination node. Here, $h(n)$ is the shortest path from n to d based on the minimum travel times on each edge. Let S_{TAG} be the graph derived from a time aggregated graph where each edge cost time series has been replaced by a scalar cost equal to the minimum edge cost. Let P be the shortest path from node i to d in S_{TAG}.
Shortest path travel time $SP_{min} = \sum_{pq \in P} d_{pq}^{min}$
Let $P^*(t)$ be the shortest path in TAG that starts at node i at time t. For $h(n)$ to be admissible, $SP_{min} \leq SP(t)$.
$SP_{min} = \sum_{pq \in P} d_{pq}^{min} \leq \sum_{kl \in P^*} d_{kl}^{min}$ and $d_{kl}^{min} \leq d_{kl}(t)$.
$SP_{min} \leq \sum_{kl \in P^*} d_{kl}^{min} \leq \sum_{kl \in P^*} d_{kl}(t) = SP(t)$. The heuristic function is admissible.

Lemma 5: The heuristic function $h(n)$ is monotone.

Proof: A heuristic function h(n) is monotone if $h(i) \leq d_{ij} + h(j) \forall ij \in E$. Here, $SP_{id}^{min} \leq d_{ij}(t) + SP_{jd}^{min}$. $SP_{id}^{min} \leq d_{ij}^{min} + SP_{jd}^{min}$; else, it is a contradiction to the optimality of SP_{id}^{min}. Since $d_{ij}^{min} \leq d_{ij}^{(}t)$, $SP_{id}^{min} \leq d_{ij}(t) + SP_{jd}^{min}$.
Since the heuristic function is admissible and monotone, the A* algorithm finds an optimal solution and performs and optimal search [18], [28].

Lemma 6: SP-TAG_* is correct.

Proof: The A* algorithm finds the shortest path for a given start time. The heuristic function underestimates the shortest path travel time from the source to the destination since in the computation of the estimte h, the edge travel times are replaced by the minimum values over the entire time horizon. Hence it is not possible for h to exceed the actual travel time. Since the heuristic function is admissible, the A* algorithm finds an optimal solution. [18], [28]. Since the heuristic function is monotone, the search process will not open any search node that was once expended and closed. Hence the search is optimal.

Lemma 7: The time complexity of the SP-TAG_* algorithm is $O(m(\log T + \log n))$ where T is the number of time instants, n is the number of nodes and m is the number of edges in the time aggregated graph.

Proof: The cost model analysis assumes an adjacency list representation of the graph with two significant modifications. Attached to every adjacent node in the linked list is the travel time series.
For every node extracted from the priority queue Q, there is one edge time series look up and a priority queue update for each of its adjacent nodes. The time complexity of this step is $O(\log T + \log n)$. The asymptotic complexity of the algorithm would be
$O(\Sigma_{v \in N}[degree(v).(\log T + \log n]) = O(m(\log T + \log n))$.

Lemma 8: The NF-SP-TAG algorithm is correct.
Proof: The algorithm runs on the transformed TAG where the edge costs are the arrival times at the end node of the edge. Here we prove that the algorithm computes the shortest path using the greedy strategy. The key idea behind the algorithm is that, once the start time is fixed, the earliest arrival at any node implies a shortest path journey. If this is not the case, it contradicts the earliest arrival. The algorithm, at every step, picks the node with the least cost and expands theis node. While expanding the node, it selects the minimum of all the edge costs, that is greater than the arrival time at the node. The cost of a node is updated as follows using the minimum in the edge time series $c[v] = c[u] + min_{t \geq c[u]} a_{uv}[t]$. $Minimum_{\forall t > t1}(a_{ij}[t]) < Minimum_{\forall t > t2}(a_{ij}[t])$ if $t1 \leq t2$.
Since the algorithm always picks the minimum of the edge costs, it ensures the earliest possible arrival.

Here to transform the problem space so that a correct greedy algorithm can be formulated, the travel times are added to the start times to compute the arrival times at the end node. These arrival times are the edge costs in T_TAG and tge greedy algorithm selects the minimum of the arrival times. This method would give correct answers as long as the edge costs are positive and the parameters display additive property.

Lemma 9: The time complexity of the NF-SP-TAG algorithm is $O(m(T + \log n))$ where T is the number of time instants, n is the number of nodes and m is the number of edges in the time aggregated graph.
Proof: The cost model analysis assumes an adjacency list representation of the graph with one significant modification. Attached to every adjacent node in the linked list is the arrival time series with the start time instants.
For every node extracted from the priority queue Q, there is one edge series look up to find the earliest arrival, and a priority queue update for each of its adjacent nodes. The time complexity of this step is $O(T + \log n)$. The asymptotic complexity of the algorithm would be $O(\Sigma_{v \in N}[degree(v).(T + \log n]) = O(m(T + \log n))$.

APPENDIX III
PROOFS FOR LEMMAS IN SECTION 4

Lemma 10: The algorithm terminates and computes the best start time paths from every node to the destination.

Proof: The algorithm terminates because there is a positive min imum for the travel time over every path, for every pair of nodes in the network since the edge weights (travel times) are positive and each such path has a finite number of edges. The updates on the costs according to condition(1) will generate the optimal travel times from a node to the destination at the termination of the algorithm. This can be proved by induction on the number of edges on the path. The base condition would be for paths with two edges, say from any node u to the destination node d. Every path with two edges from u to d will transit to some node v and then traverse the edge to d which takes the least time. If we assume the inductive hypotheses is true for every path with k edges, the minimality must hold for a path from u with $(k+1)$ edges since we can reach node u that with a minimal $k-$edge path and append uv with travel time $\sigma_{uv}(t)$.

Lemma 11: The computational complexity of the BST algorithm is $O(n^2mT)$, where n is the number of nodes, m is the number of edges and T is the length of the time series.

Proof: The worst case computational complexity of the label corr ecting algorithm based on Two-Q data structure is $O(n^2m)$ when the node costs and edge weights are scalar quantities [3]. In the BST algorithm, the relaxation step operates on a time series (node cost and edge weight) of length T. Hence the computational complexity of the algorithm is $O(n^2mT)$.

Lemma 12: The algorithm computes the least duartion journey paths from the source node to the destination.

The proof follows from the correcteness of EA-tTAG algorithm (Lemma 8). The algorithm TI_EA-tTAG iterates EA-tTAG over the entire time period. At each start time (an iteration), the algorithm computes the earliest arrival, which is the shortest duration journey for the given start time. Since there is a minimum for every start time and the algorithm picks the minimum of these durations, the algorithm computes the least duration journey shortest path.