

# Time-aggregated Graphs for Modeling Spatio-Temporal Networks\*

Betsy George\*\* and Shashi Shekhar

Department of Computer Science and Engineering,  
University of Minnesota,  
200 Union St SE, Minneapolis, MN 55455, USA,  
E-mail: [bgeorge, shekhar]@cs.umn.edu  
WWW home page: <http://www.spatial.cs.umn.edu/>

**Abstract.** Given applications such as location based services and the spatio-temporal queries they may pose on a spatial network (e.g., road networks), the goal is to develop a simple and expressive model that honors the time dependence of the road network. The model must support the design of efficient algorithms for computing the frequent queries on the network. This problem is challenging due to 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 networks across time instants, resulting in high storage overhead and algorithms that are computationally expensive. In contrast, the proposed time-aggregated graphs do not replicate nodes and edges across time; rather they allow 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 are computationally more efficient than for time expanded networks. One important query on spatio-temporal networks is the computation of shortest paths. Shortest paths can be computed either for a given start time or to find the start time and the path that lead to least travel time journeys (best start time journeys). Developing efficient algorithms for computing shortest paths in a time variant spatial network is challenging because these journeys do not always display optimal prefix property, making techniques like dynamic programming inapplicable. In this paper, we propose algorithms for shortest path computation for a fixed start time. We present the analytical cost model for the algorithm and compare with the performance of existing algorithms.

**Keywords:** time-aggregated graphs, shortest paths, spatio-temporal databases, location based services

---

\* This work was supported by NSF/SEI grant 0431141. The content does not necessarily reflect the position or policy of the government and no official endorsement should be inferred.

\*\* Corresponding author: Betsy George, E-mail: [bgeorge@cs.umn.edu](mailto:bgeorge@cs.umn.edu)

## 1 Introduction

The growing importance of application domains such as transportation networks, emergency planning and location based services highlights the need for efficient modeling of spatio-temporal networks (e.g. road networks) that takes into account changes to the network over time. The model should provide the necessary framework for developing efficient algorithms that implement the frequent operations posed on such networks. Frequent queries on such networks might include finding the shortest route from one place to another or a search for the nearest neighbor. The shortest route would depend on the time dependent properties of the network such as congestion on certain road segments, which would increase the travel time on that segment. The result of a nearest neighbor search could also be time sensitive if it is based on a road network.

Modeling such a network poses many challenges. Not only should the model be able to accommodate changes and compute the results consistent with the existing conditions, it should do so accurately and simply. In addition, the need to answer frequent queries quickly means fast algorithms are required for computing the query results. The model should thus provide sufficient support for the design of correct and efficient algorithms for frequent computations.

Related work in the field of databases fall 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 computation [15]. 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.) [7]. However, these products do not address the time variance of spatial networks, which is crucial in applications such as route computations and emergency planning. 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 [13], studied various aspects of spatio-temporal databases including ontology, modeling, and implementation. However, researchers have yet to study spatio-temporal networks in this framework.

Graph databases [5–7, 19, 23, 24] also primarily deal with spatial networks that do not vary with time. Research in graph databases that accounts for temporal variations perform computations over a snapshot of the network [4, 10, 18], and do not consider the interplay between the edge travel times and the existence of edges. Ding [4] 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. It does not propose an algorithm for the least travel time paths.

Research in Operations Research is based on the time expanded network [11, 12, 14, 16, 17, 21]. This model duplicates the original network for each discrete time unit  $t = 0, 1, \dots, T$  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. The approach 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 also become quite expensive.

As the first step towards the study of spatio-temporal network databases, we proposed a spatio-temporal network model named time aggregated graph [8]. The proposed model, a time-aggregated graph, models the changes in a spatio-temporal network by collecting the node/edge attributes into a set of time series. The model can also account for the changes in the topology of the network. The edges and nodes can disappear from the network during certain instants of time and new nodes and edges can be added. The time-aggregated graph keeps track of these changes through a time series attached to each node and edge that indicates their presence at various instants of time. Our analysis shows that this model is less memory expensive and leads to algorithms that are computationally more efficient than those for the time expanded networks. Here, we build on this work by presenting a case study of this model using a routing algorithm (SP-TAG) that computes the shortest path in the given network for a given start time.

### 1.1 An Illustrative Application Domain

An important application domain for spatio-temporal network databases is transportation science [9], a multi-disciplinary field that requires expertise from different domains. The difficulty, but also fascination, of this professional practice derives from the intrinsic complexity of transportation systems, which have both physical and behavioral elements. The physical elements in the systems (e.g., vehicles, infrastructure, etc.) are governed by the laws of physics. On the other hand, the mechanisms underlying the functionality and performance of these physical elements are often connected to travelers' behavioral choices. Traditionally the center of behavioral choice modeling [22] has been user equilibrium [25], the idea that all travelers use the least inconvenient routes and no individual can unilaterally improve his/her travel. A key assumption of user equilibrium is that travelers have perfect information about road conditions, and indeed this is generally true for commuters, who learn recurrent congestion patterns from their day-to-day travels. However, the assumption does not hold when the congestion is non-recurrent, in particular, when an extreme event occurs, and transportation network conditions become dynamic and uncertain. Thus one of the greatest challenges in transportation science is how to manage traffic in time-varying transportation networks, especially in disaster situations. This challenge cannot be met without the development of spatio-temporal databases. Currently, transportation management generates tremendous volumes of data and a large semantic gap exists between transportation science concepts and the concepts supported by current database systems. Emergency traffic management requires research in computer science to develop appropriate spatio-temporal database representations and query processing algorithms to make decisions in a timely

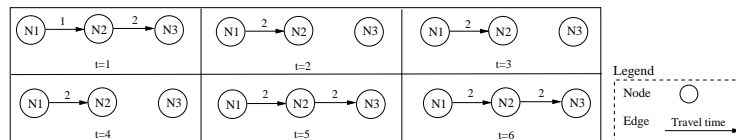
manner. Popular models of emergency traffic use time-variant flow-network [1] operations like min-cut and max-flow [2]. The queries typically encountered in emergency traffic management would involve time-variant properties, as illustrated by Table 1.

In addition to emergency management, many important applications, including

**Table 1.** Example Queries with Time-variance and Flow Networks

	Static	Time-Variant
Graph (No capacity constraints)	Which is the shortest travel time path from Minneapolis downtown to airport?	Which is the shortest travel time path from Minneapolis downtown to airport at different times of a work day?
Flow Network	What is the capacity of Twin-Cities freeway network to evacuate Minneapolis downtown?	What is the capacity of Twin-Cities freeway network to evacuate Minneapolis downtown at different times in a work day?

travelers' trip planning, and consumer business logistics need to be built upon spatio-temporal network databases. Commuters often try to find a suitable time to start their commute so that they spend the least time in the traffic. There are many factors affecting the start time and the shortest route such as congestion levels, incident location, and construction zone. This is illustrated by the simple time-variant network shown in Figure 1. It can be seen that the travel time from node N1 to node N2 changes with the start time. If the travel starts at  $t = 1$ , the commute time would be 6 units; travel on the same route would take 4 units if the start time is moved to  $t = 3$ . This shows that the shortest paths in a time-dependent network vary with time which adds a new dimension to shortest path computation which cannot be ignored. Figure 2 illustrates traffic sensor

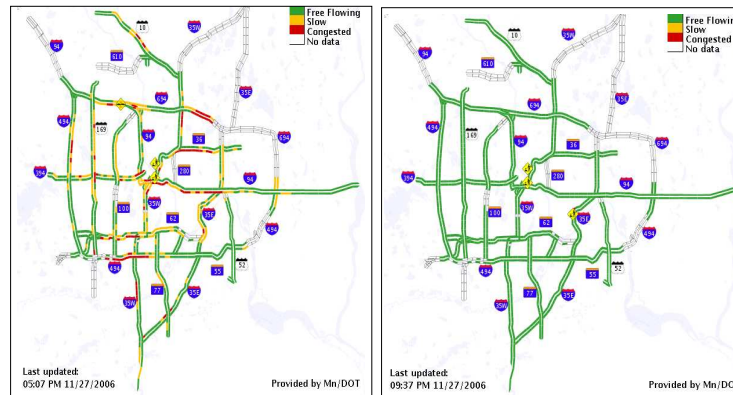


**Fig. 1.** Network at various instants

networks on urban highways which measure congestion levels at two different times (e.g. 5:07pm and 9:37pm). With the increasing use of sensor networks that monitor traffic data on spatial networks and the consequent availability

of time-varying traffic data, it becomes important to incorporate this data into the models and algorithms related to transportation networks. However, existing spatio-temporal databases do not provide adequate support for spatio-temporal networks.

The problem of time variant networks finds similar applications in business op-



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

erations such as freight delivery services, one of whose main concerns is to reduce logistic costs such as fuel consumption, which is influenced by road congestion levels that vary over time.

## 1.2 Broad Computer Science Challenges

A time-variant graph is a graph whose edge and node properties and topological structure are time dependent. For example, traffic volume on urban highways varies over the time of day which leads to variation in travel time. In addition to network parameter values, the network topology can also change with time due to the unavailability of certain road segments during some periods of time due to repair or natural calamities. There are also cases where the road segments are unavailable periodically due to traffic management strategies such as using all lanes of a street in the same direction to handle peak time congestion. Conventional graph algorithms cannot easily be applied to the snapshots at discrete time instants to evaluate frequent queries without accounting for relationships among snapshots.

Time-variant graphs 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 graph-properties may not hold for spatio-temporal graphs. For example, consider the optimal prefix property (a requirement for the greedy approaches [2]) for 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 the potential wait at the intermediate node. In the network shown in Figure 1, the best time to start 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 optimal prefix property in best start time shortest paths rules out the possibility of using a greedy strategy in algorithm design.

**Our Contributions:** Our approach to spatio-temporal databases has the following components:

*Graph Aggregation:* The temporal variation in the topology and parameter values can be represented using aggregates as edge/node attributes in the graph used to represent the spatial network. The edges and nodes can disappear from the network during certain instants of time and new nodes and edges can be added. The time-aggregated graph keeps track of these changes through a time series attached to each node and edge that indicates their presence at various instants of time.

*Query Language:* A query language needs to represent common queries. A key challenge is to define a complete set of logical operators for the time-aggregated graph.

*Query Processing:* The time aggregated graph with the proposed query operators will be used to process queries pertaining to the domain applications. A frequent query that arises in spatio-temporal networks is the shortest path computation. The algorithm needs to consider the availability of the required edges and nodes at the appropriate time instants. If the shortest route and the shortest route travel time are time-dependent, shortest path computation can be performed for a given start or it can find the least travel time path over the entire time period of interest.

In this paper we describe a model for spatio-temporal networks called the time aggregated graph based on graph aggregation.. The time-aggregated graph keeps track of the time-dependence of a graph through a time series attached to each node and edge that indicates their presence at various instants of time. We show that this model has less storage requirements than time expanded networks since it does not rely on replication of the entire network across time instants. We define a set of logical operators based on the time aggregated graph. We also

propose an algorithm for computing the shortest route from one node to another based on this model.

### 1.3 Scope and Outline of the Paper

The paper presents a model for spatio-temporal networks called time aggregated graphs.

The rest of the paper is organized as follows. Section 2 discusses the basic concepts of the proposed model. This section provides an explanation of the model based on graph aggregation and the logical data model. It also explores design choices for the physical representation of the model and provides a comparison of the choices in the context of various logical operations. Section 3 proposes an algorithm for the shortest path computation based on this model. It also proposes the cost model for this algorithm. In section 4, we conclude and describe the direction of future work.

## 2 Basic Concepts

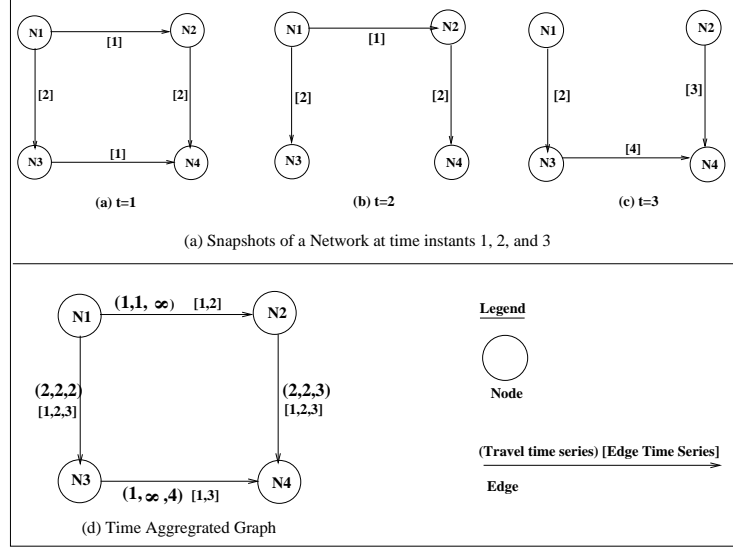
Spatial networks that show time-dependence serve as the underlying networks for most location based services. Traditionally graphs have been extensively used to model spatial networks (e.g. road networks) [19]; weights assigned to nodes and edges are used to encode additional information. In a real world scenario, it is not uncommon for these network parameters to be time-dependent. Formulation of computationally efficient and correct algorithms for the shortest path computation that takes into account the dynamic nature of the networks is important. 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 the set of frequent queries posed by an application on a spatial network and the patterns 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 spatial networks called "time aggregated networks" [8]. The algorithm presented in this paper is 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.

### 2.1 The Conceptual Model

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 determine 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 the edges. In this section, we discuss

how the time dependence of these edge/node parameters are handled in the proposed model, the time-aggregated graph.

We define the time-aggregated graph as follows.



**Fig. 3.** Network At Various Time Instants and the Time Aggregated Graph

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

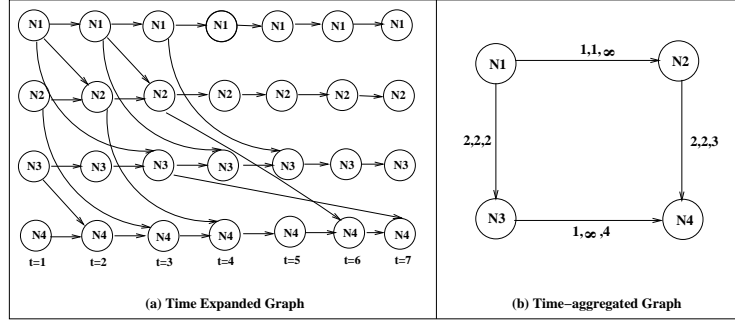
$g_1 \dots g_l$  are mappings from edges to the time series associated with the edges, and

$w_1 \dots w_p$  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 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) shows a network at three time instants. The network topology and parameters change over time. For example, edge N3-N4 is present at time instants  $t = 1, 3$ , and disappears at  $t = 2$  and its weight changes from 1 at  $t = 1$  to 4 at  $t = 3$ . The time aggregated graph that represents this dynamic network is shown in Figure 3(d). In this figure, edge N3-N4 has two attributes, both being

a series. The attribute  $(1, 3)$  represents the time instants at which the edge is present and  $[1, \infty, 4]$  is the weight time series, indicating the weights at various instants of time. Figure 4(a) shows the time aggregated graph (corresponding



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

to Figure 3(a),(b),(c)) and the time expanded graph that represent 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 N1 at  $t = 2$ . The time expansion for the example network needs to go through 7 steps since the latest time instant would end in the network is at  $t = 7$ . For example, the traversal of 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 very large in a spatial network, it would result in enormously large time expanded networks and consequently slow computations.

## 2.2 A Logical Data Model

### Basic Graph Operations

We extend the logical data model described in [19] to incorporate the time dependence of the graph model. The framework of the model consists of two dimensions (1) graph elements, namely node, edge, route and graph and (2) operator categories that consist of accessors, modifiers and predicates. A representative set of operators for each operator category is provided in Tables 2, 3 and 4. Table 2 lists a representative set of ‘access’ operators. For example, the operator  $getEdge(node1, node2, time)$  returns the edge properties of the edge from node  $node1$  to node  $node2$ , such as the edge identifier (if any) and associated parameters at the specified time instant. For example operator  $getEdge(N1, N2, 1)$  on

**Table 2.** Examples of Operators in the Accessor Category

	<b>at_time</b>	<b>at_all_time</b>	<b>at_earliest</b>
<b>Node</b>	get(node,time)	get_node(node)	get_node_earliest_Presence (node,time)
<b>Edge</b>	getEdge(node1,node2,time)	get_edge(node1,node2)	get_edge_earliest_Presence (node1,node2,time)
<b>Route</b>	getRoute(node1,node2,time)	get_route(node1,node2)	get_route_earliest_Presence (node1,node2,time)
<b>Graph</b>	get_Graph(time)	get_Graph()	—

the time-aggregated graph shown in Figure 3 would return the travel time of the edge N1-N2 at  $t = 1$ , that is 1. Similarly,  $get\_edge(node1,node2)$  returns the edge properties for the entire time interval. In Figure 3, the operator  $get\_edge(N1,N2)$  would result in  $(1, 1, \infty)$ .  $get\_edge\_earliest(N3,N4,2)$  returns the earliest time instant at which the edge N3-N4 is present after  $t = 2$  (that is  $t = 3$ ). Table 3

**Table 3.** Examples of Operators in the Modifier Category

	<b>insert</b>		<b>delete</b>		<b>modify</b>	
	<b>at_time</b>	<b>at_all_time</b>	<b>at_time</b>	<b>at_all_time</b>	<b>at_time</b>	<b>at_all_time</b>
<b>Node</b>	insert(node, time,value)	insert(node, valueseries)	delete(node, time)	delete(node) delete(node)	update(node, time,value)	update(node, valueseries)
<b>Edge</b>	insert(node1, node2, time,value)	insert(node1, node2, valueseries)	delete(node1, node2, ,time)	delete(node1, ,node2) ,node2)	update(node1, node2,time value)	update(edge, valueseries)
<b>Route</b>	insert(node1, node2,time)	insert(node1, ,node2)	delete(node1, ,node2,time)	delete(node1, node2)		
<b>Graph</b>	insert(graph time)	insert(graph) insert(graph)	delete(graph, time)	delete(graph) delete(graph)	update(graph, ,time)	update(graph)

shows a set of modifier operators that can be applied to the time aggregated graphs. For example, Figure 5(a) and (b) show a time aggregated graph before and after the  $insert(N1,N4,3,4)$  operation. this operation inserts edge N1-N4 at time instant  $t = 3$  and the edge cost is 4. We also define two predicates on the time-aggregated graph.

**exists\_at\_time\_t:** This predicate checks whether the entity exists at the start time instant  $t$ .

**exists\_after\_time\_t:** This predicate checks whether the entity exists at a time instant after  $t$ .

Table 4 illustrates these operators. For example, node  $v$  is adjacent to node  $u$  at any time  $t$  if and only if the edge  $(u, v)$  exists at time  $t$  as shown in the table.  $exists(N1,N2,1)$  on the time aggregated graph in Figure 3 returns a "true" since the edge N1-N2 exists at  $t = 1$ .

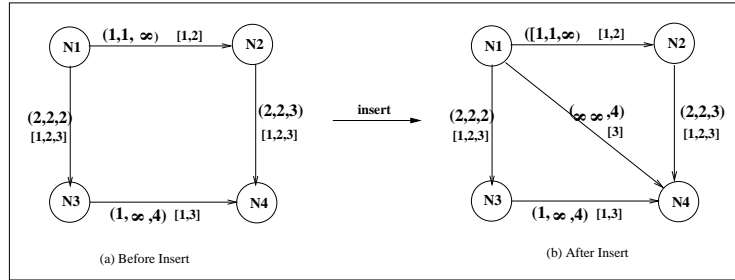


Fig. 5. A Time Aggregated Graph before and after an Insert Operation

Table 4. Predicate Operators in Time-aggregated Graphs

	exists_at_time_t	exists_after_time_t
Node	exists(node u,at_time_t)	exists(node u,after_time_t)
Edge	exists(node u,node v, at_time_t)	exists(node u,node v, after_time_t)
Route	exists(node u,node v,a_route r at_time_t)	exists(node u,node v,a_route r, after_time_t)

We list below, the fundamental entities in graphs, namely, *Graph*, *Node*, and *Edge* and a series of common operations that are associated with each class.

```

public class Graph {
    public void add(Object label, timestamp t);
    // node with the given label is added at the time instant t.

    public void addEdge(Object n1, Object n2, Object label,
                        timestamp t, timestamp t_time)
    // an edge is added with start node n1 and end node n2 at
    // time instant t and travel time, t_time.

    public Object delete(Object label, timestamp t)
    // removes a node at time t and returns its label.

    public Object deleteEdge(Object n1, Object n2, timestamp t)
    // deletes the edge from node n1 to node n2 at t.

    public Object get(Object label, timestamp t)
    // returns the label of the node if it exists at time t.

    public Iterator get_node_Presence_Series(Object n1)
    // the presence series of node n1 is returned.

    public Object getEdge(Object n1, Object n2, timestamp t)

```

```

// returns the edge from node n1 to node 2 at time instant t.

public Iterator get_edge_Presence_Series(Object n1, Object n2)
// the presence series of edge from node n1 to node n2
// is returned.

public Object get_a_Successor_node(Object label, timestamp t)
// an adjacent node of the vertex is returned if an edge exists
// to this node at a time instant at or after t.

public Iterator get_all_Successor_nodes(Object label, timestamp t)
// all adjacent nodes are returned if edges exist to them
// at time instants at or after t.

public Object get_an_earliest_Successor_node(Object label,timestamp t)
// the adjacent node which is connected to the given node with
// the earliest time stamp after t is returned.

public timestamp get_node_earliest_Presence(Object n1,
                                           timestamp t)
// the earliest time stamp after t at which the node n1
// is available is returned.

public timestamp get_node_Presence_after_t(Object n1,
                                           timestamp t)
// Part of the presence time series of node n1 after time t
// is returned.

public timestamp get_edge_earliest_Presence(Object n1, Object n2,
                                           timestamp t)
// the earliest time stamp after t at which the edge from
// node n1 to node n2 is available is returned.

public timestamp get_edge_Presence_after_t(Object n1, Object n2,
                                           timestamp t)
// Part of the presence time series of edge(n1-n2) after time t
// is returned.
}

```

A few important operations associated with the classes **Nodes** and **Edges** are provided below.

```

public class Node {
    public Node(Object label, timestamp t)

```

```

        // the constructor for the class. A node with the appropriate
        // label is created at the time t.

        public Object label()
        // returns the label associated with the node if it exists at t.
    }

public class Edge {
    public Edge(Object n1, Object n2, Object label,
                timestamp t_inst, timestamp t)
        // the constructor for the class. an edge is added with start
        // node n1 and end node n2 at time instant t and
        // travel time, t_time.

    public Object start()
        // returns the start node of the edge.

    public Object end()
        // returns the end node of the edge.
}

```

### 2.3 Physical Data Model

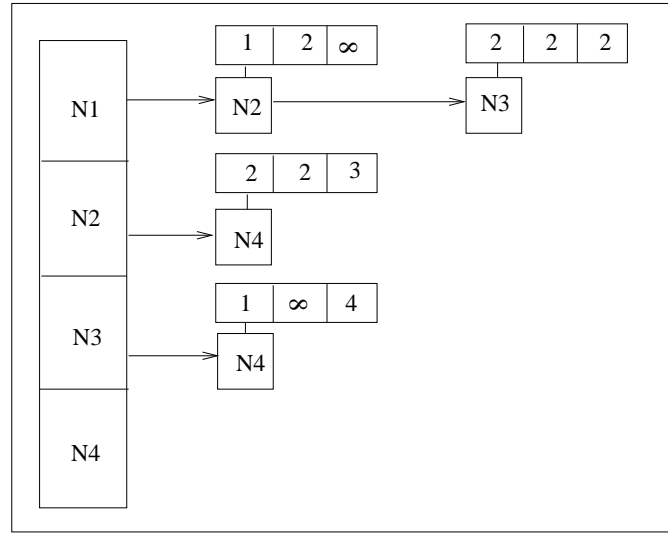
A static graph  $G = (V, E)$  can be represented using an adjacency matrix. This is a  $|V| \times |V|$  matrix,  $A$  such that the element  $a_{ij}$  is defined as  $a_{ij} = w_{ij}$  if  $ij \in E$  and  $w_{ij}$  is the weight of the edge  $ij$  and  $a_{ij} = 0$ , otherwise. This representation requires  $O(N^2)$  memory. It can be seen that the storage required for this representation is independent of the number of edges in the graph, in relation to the number of nodes. In other words, there is no saving in memory even when the graphs are sparse. One representation that can exploit such sparsity is the adjacency list representation. The adjacency list representation of a graph  $G = (V, E)$  consists of an array of lists, one for each vertex  $v \in V$ . The list corresponding to a vertex  $v$  contains all vertices that are adjacent to  $v$  in  $G$ . For a directed graph, the space requirement for the lists is  $O(m)$  where  $m = |E|$ . The total memory requirement is  $O(n+m)$  where  $n = |V|$ . The weight of each edge  $uv$  is stored with the vertex  $v$  in  $u$ 's adjacency list. This representation is especially suitable for sparse graphs.

Time aggregated graphs can be represented by either one of the representation, with the necessary modifications. These representations need to be extended to include the time series representations on edges (corresponding to time dependent edge costs) and nodes. Adjacency list representation is extended by adding a list to each vertex in the adjacency list. Adjacency list representation uses an array of pointers one pointer for each node. The pointer for each node points to a list of immediate neighbors. Stored at each neighbor node are the edge presence series and travel times for the edge starting from the first node to this neighbor. Since the length of the time series is  $T$ , where  $T$  is the length of

the time period, the adjacency list representation would require  $O((m+n)T)$ , where  $n$  is the number of nodes and  $m$  is the number of edges.

To extend the adjacency matrix to represent the time aggregated graph, a third dimension can be added. The new matrix  $A$  would be  $n \times n \times T$ , requiring  $O(n^2T)$  memory.

Figure 6 (a) and (b) show the adjacency list and adjacency matrix representa-



(a) Adjacency List Representation

	N1	N2	N3	N4
N1	∞	1	2	∞
N2	∞	∞	∞	2
N3	∞	∞	∞	1
N4	∞	∞	∞	∞

t=1

	N1	N2	N3	N4
N1	∞	1	2	∞
N2	∞	∞	∞	2
N3	∞	∞	∞	∞
N4	∞	∞	∞	∞

t=2

	N1	N2	N3	N4
N1	∞	1	2	∞
N2	∞	∞	∞	∞
N3	∞	∞	∞	4
N4	∞	∞	∞	∞

t=3

(b) Adjacency Matrix Representation

**Fig. 6.** Storage Structures for Time Aggregated Graph

tions for the time aggregated graph shown in Figure 3. For example, the edge N1-N2 in the graph at  $t = 1$  is represented by the pointer from N1 to N2 in the adjacency list. The array  $(1, 2, \infty)$  is stored at N2 to represent the travel times at  $t = 1, 2, 3$  for the edge N1N2. In the adjacency matrix the presence of edge

$N1N2$  at a time instant  $t = 1$  is represented by  $A[1, 2, 1] = 1$ , since the travel time for the edge is 1 unit at  $t = 1$ . Since the edge is absent at an instant  $t = 3$ ,  $A[1, 2, 3] = \infty$  which indicates an infinite edge cost at time instant  $t = 3$ . Note that the start node, the end node and the time instant are represented by the first, second and the third dimension of the matrix. Though the adjacency matrix has been illustrated as three snapshots in Figure 6(b) for the sake of clarity, they are represented in one, three-dimensional matrix.

Logical operations on a time-aggregated graph can be classified as

1. Topology first operators (graph dominated operations). Examples include `get_route(n1,n2)` and `get_edge(n1,n2)`.
2. Time-first operators (Time dominated queries).  
Some examples are `get_Graph(time t)` and `get_edge_at.t(n1,n2,t)`.

Both representations are equally capable of handling graph dominated queries. To compute time first operations (snapshot queries such as to find the graph at a given time instant), adjacency matrix representation is more suitable. In this representation, these queries represent the time slices of the matrix at the given time instants.

Graphs representing transportation networks are generally sparse and hence adjacency list representation is more likely to be storage efficient compared to adjacency matrix representations. The choice is hence a tradeoff between the storage cost and the frequency of time dominated queries. We expect route queries (which are topology first queries) to be more frequent and since adjacency list representation is capable of handling these, based on storage costs, we used adjacency lists in our implementations. Moreover, most databases use adjacency list representation.

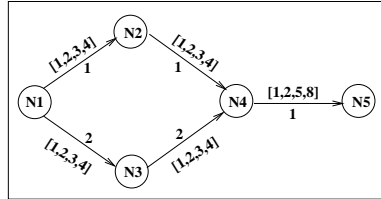
#### **Comparison of Storage Costs with Time Expanded Networks:**

According to the analysis in [20], 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 average length of the time series is  $\alpha (\leq T)$ , the memory required is  $O(\alpha m + \alpha n)$ , assuming an adjacency list representation. 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  $\alpha < T$ .

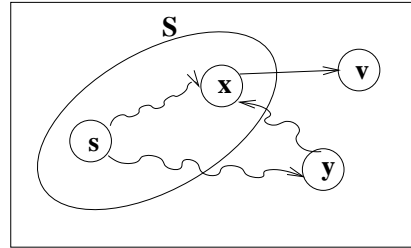
### **3 Shortest Path Computation for Time Aggregated Graphs (SP-TAG Algorithm)**

In a time dependent network, the shortest path and its traversal time are dependent on the start time at the source node. Though it is common to apply greedy

strategies in optimization problems such as shortest path computation, this approach presents a challenge in time aggregated graphs, where not all shortest paths display an optimal sub-structure. Figure 7 gives an example. For the sake of simplicity, the travel times are assumed to be constant in this example. It can



**Fig. 7.** Illustration of Shortest Paths



**Fig. 8.** Correctness of SP-TAG Algorithm

be seen that a shortest path (N1-N3-N4-N5) from N1 to N5 for the start time  $t = 1$ , which takes 5 time units does not display optimal substructure. The path from N1 to N4 following the above path is not optimal (shortest path being, N1-N2-N4). Although such 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 [8].

**Lemma 1:** There is at least one optimal path which satisfies the optimal sub-structure property.

**Proof:** As Figure 7 illustrates, a shortest path fails to display optimal structure of due to a potential wait at intermediate node ( $u$ ), after reaching this node traversing the optimal path from  $s$  to  $u$ . Consider the optimal path from  $s$  to  $u$ . Append this path to the path  $u - d$  (allowing a wait at the intermediate node  $u$ ) from the optimal path. This would still be the shortest path from  $s$  to  $d$ . Otherwise, it will contradict the optimality of the original shortest path.

This result enables us to use a greedy approach to compute the shortest path. The algorithm, called SP-TAG uses greedy 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 (*get\_edge\_earliest\_Presence* operation in the algorithm description). A trace of the algorithm is given in table 5. 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 minimum cost selected for expansion. The travel times are assumed to follow the FIFO property.

**Lemma 2:** The SP-TAG algorithm is correct.

**Proof:** The proof of correctness of the algorithm which follows a greedy strategy

---

**Algorithm 1 Shortest Path (SP-TAG) Algorithm**

---

**Input:**

- 1)  $G(N, E)$ : a graph  $G$  with a set of nodes  $N$   
and a set of edges  $E$ ;  
define type  $nn$  positive integer  
Each node  $n \in N$  has one property:  
*NodePresenceTimeSeries* : series of  $nn$   
Each edge  $e \in E$  has two properties:  
*Edge Presence TimeSeries*,  
*Travel time series* : series of  $nn$   
 $\sigma_{u,v}(t)$  - travel time of edge  $uv$  at time  $t$ .
- 2)  $s$ : Source node,  $s \subseteq N_G$ ;
- 3)  $d$ : Destination node,  $d \subseteq N_G$ ;
- 4)  $t_{start}$ : Start Time;

**Output:** Shortest Route from  $s$  to  $d$  for  $t_{start}$

**Method:**

```
 $c_s = t_{start}; \forall v \neq s, c_v = \infty;$ 
//  $c_u$  is the cost at the node  $u$ .
insert( $Q, s$ );
// $Q$  is a min-heap.
while  $Q$  not empty {
     $u = extract\_min(Q)$ ;
    for each node  $v$  adjacent to  $u$  do {
         $t = get\_edge\_earliest\_Presence(u, v, c_u)$ ;
         $\sigma = get\_edge(u, v, t)$ ;
        relax( $u, v, \sigma$ );
        insert( $Q, v$ ) if  $v$  is relaxed;
    }
    update( $Q$ );
}
}
}
Output the route from  $s$  to  $d$ .
```

---

follows the proof of correctness for Dijkstra's algorithm to find the shortest path from a source 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 8). 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

**Table 5.** Trace of the SP-TAG Algorithm for the Network shown in Figure 7

Iteration	N1	N2	N3	N4	N5
1	1 (closed)	$\infty$	$\infty$	$\infty$	$\infty$
2	1	2 (closed)	3	$\infty$	$\infty$
3	1	2	3 (closed)	3	$\infty$
4	1	2	3	3 (closed)	6
5	1	2	3	3	6 (closed)

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 8. 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 than the path  $s \cdots xv$ . The cost of  $v$  cannot be further 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 representation 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

$$O(\sum_{v \in N} [degree(v) \cdot (\log T + \log n)]) = O(m(\log T + \log n)).$$

The time complexity of the SP-TAG shortest path algorithm based on a time expanded network is  $O(nT \log T + mT)$  [3]. 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  $\log n < T$ .

### 3.1 Summary of Experimental Evaluation:

An experimental analysis of the SP-TAG algorithm was performed to compare its run-time with an algorithm based on a time-expanded graph. Time expanded graphs make copies of the original network for every time instant under consideration. In our experiments the following were selected as the independent parameters: 1) network size represented by the number of nodes; and 2) the length of the time interval in terms of number of time instants. The networks chosen were road maps from the Minneapolis downtown area in USA with radii of .5 mile, 1 mile, 2 miles and 3miles. This was 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 an expanded graph encoding of the travel time information. An algorithm for computing the shortest path for a given start time was run on this graph. The SP-TAG algorithm 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. The experimental results reported are the average over 5 experiment runs with networks generated using the same input parameters, but with different destination nodes.

### Experimental Results and Analysis

We wanted to answer two 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?

In the experiment to evaluate the effect of network size on the performance of the algorithm, we fixed the length of the travel time series and varied the network size to observe the run times of the SP-TAG algorithm and time expanded graph based algorithm. Figure 9 shows the run-time of the fixed start time algorithm

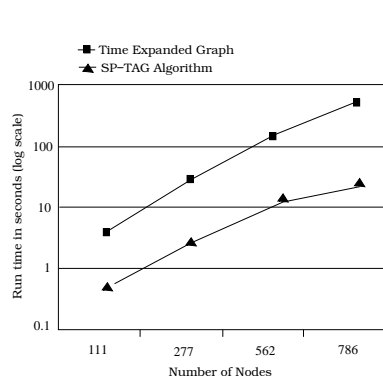


Fig. 9. SP-TAG Algorithm: Run-time With Respect to Network Size

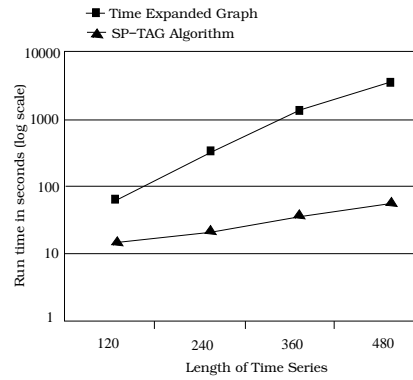
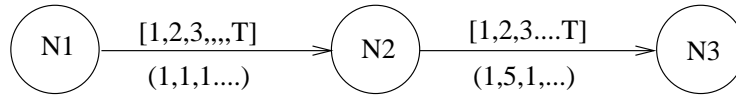


Fig. 10. SP-TAG Algorithm: Run-time With Respect to Length of Time Series

based on the time aggregated graph and the performance of the algorithm based on the time expanded graph. As can be seen, the SP-TAG algorithm runs faster than the time-expanded graph based algorithm in all cases; further, its run-time seems to increase at a slower rate.

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 and the network size parameters were fixed. As seen in Figure 10, the SP-TAG algorithm performs better.

**Discussion:** Due to the interplay between the travel times and the availability of the edges, the shortest path displays some interesting properties. Consider the time aggregated graph shown in Figure 11. Assume that edges are present



**Fig. 11.** Illustration of Effect of non-FIFO travel times

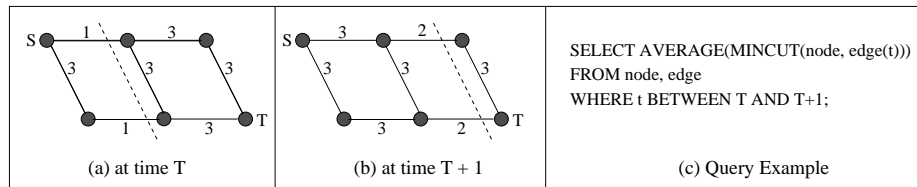
at all time instants. The travel times are time dependent as shown in the cost time series. For example, the travel time of edge N2-N3 is 5 units at  $t = 2$  and 1 unit at  $t = 3$ . Consider a journey that starts at  $t = 1$  at node N1. It reaches node N2 at  $t = 2$ . Since edge N2-N3 is available at  $t = 2$  (this being the earliest availability), the greedy algorithm would traverse the edge at  $t = 2$  reaching node N3 at  $t = 7$ , travel time being 5 time units at  $t = 2$ . If the algorithm had made a decision to wait at N2 until  $t = 3$ , N3 would be reached at  $t = 4$ . This indicates that the travel times should follow the FIFO property for the greedy algorithm to compute the optimal path. Though the FIFO property is satisfied by travel times in most situations, strategies can be explored to handle non-FIFO travel times also.

## 4 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 database support in this area. The paper describes a model based on time aggregation to represent a spatio-temporal network and proposes an algorithm for shortest path computation. It also defines a set of logical operators on the time aggregated graphs. We present an analytical evaluation of the shortest path algorithm and storage cost requirements of the proposed time aggregated graph. Evaluation shows that the algorithms based on time aggregated graphs significantly reduce the computational cost compared to similar algorithms based on time expanded networks and the model is less memory expensive than the existing models.

We plan to evaluate the performance of the algorithms using real-traffic datasets shortly. We expect this evaluation to give new insights into the average case run time of the algorithms, which we expect to be significantly better than the worst case complexity. We believe that time aggregated graphs can accommodate 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. Flow networks [1] have been extensively used in evacuation planning. We plan to use time aggregated graphs to represent time-variance in flow networks. Time-variance poses novel challenges for flow network operations by introducing alternative interpretations of traditional operations. Consider a query to identify bottleneck capacity of a transportation network (modeled as a minimum cut) shown in Figure 12 at two time instants T

and  $T+1$ . The numbers associated with various edges represent their capacities. At time  $T$ , the bottleneck (i.e., minimum cut) of this network is 2 for flows starting from node  $S$  towards destination node  $T$  as shown in Figure 12(a). At time  $T+1$ , the bottleneck changes to 4 as shown in Figure 12(b). Thus, the minimum-cut of this time-variant flow-network may be a function of time. A database may allow aggregate queries over time-variant network-flow properties like min-cut. Figure 12(c) shows an example of a query to find an average among time variant min-cuts with temporal range. We also plan to incorporate the algorithms as



**Fig. 12.** Two Min-cut graphs with time variant capacity and a query example

building blocks that finds the shortest paths in the CCRP evacuation planner [14]. We will also explore other graph problems in the context of time aggregated graphs. We would explore ways to include spatial attributes at nodes and edges and incorporate necessary changes in the algorithms.

Spatial properties need to be represented in the time aggregated graph, which might add to the effectiveness of the model and may lead to the formulation of efficient algorithms. For example, the spatial location of a node can be represented as a node attribute. We plan to explore effective ways to incorporate spatial properties of nodes and edges in the model. We also plan to include operators that handle time intervals as parameters.

## Acknowledgment

We are thankful to the reviewers for their helpful comments, especially in relation to the computational complexity of the shortest path algorithm. We wish to thank Sangho Kim from the Spatial Database Research Group at the University of Minnesota for his valuable input on time-variant flow graphs. We are grateful to the members of the Spatial Database Research Group at the University of Minnesota for their helpful comments and to Kim Koffolt for her help in improving the readability of this paper. This work was supported by NSF/SEI grant 0431141, Oak Ridge National Laboratory grant and US Army Corps of Engineers (Topographic Engineering Center) grant. The content does not necessarily reflect the position or policy of the government and no official endorsement should be inferred.

## References

1. R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows - Theory, Algorithms, and Applications*. Prentice Hall, 1993.
2. 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.
3. B. C. Dean. Algorithms for minimum-cost paths in time-dependent networks. *Networks*, 44(1):41–46, August 2004.
4. Z. Ding and R.H. Güting. Modeling temporally variable transportation networks. *Proc. 16th Intl. Conf. on Database Systems for Advanced Applications*, pages 154–168, 2004.
5. M. Erwig. *Graphs in Spatial Databases*. PhD thesis, Fern Universität Hagen, 1994.
6. M. Erwig and R.H. Güting. Explicit graphs in a functional model for spatial databases. *IEEE Transactions on Knowledge and Data Engineering*, 6(5):787–804, 1994.
7. ESRI. ArcGIS Network Analyst. <http://www.esri.com/software/arcgis/extensions/>, 2006.
8. B. George and S. Shekhar. Time-aggregated Graphs for Modeling Spatio-Temporal Networks - An Extended Abstract. *Proceedings of Workshops at International Conference on Conceptual Modeling*, November 2006.
9. R.W. Hall (editor). *Handbook of Transportation Science*. Kluwer Academic Publishers, 2003.
10. 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.
11. 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.
12. 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.
13. Manolis Koubarakis, Timos K. Sellis, Andrew U. Frank, Stéphane Grumbach, Ralf Hartmut Güting, Christian S. Jensen, Nikos A. Lorentzos, Yannis Manolopoulos, Enrico Nardelli, Barbara Pernici, Hans-Jörg Schek, Michel Scholl, Babis Theodoulidis, and Nectaria Tryfona, editors. *Spatio-Temporal Databases: The CHOROCHRONOS Approach*, volume 2520 of *Lecture Notes in Computer Science*. Springer, 2003.
14. Q. Lu, B. George, and S. Shekhar. Capacity Constrained Routing Algorithms for Evacuation Planning: A Summary of Results. *Proc. of 9th International Symposium on Spatial and Temporal Databases (SSTD'05)*, August 2005.
15. Oracle. Oracle Spatial 10g, An Oracle White Paper. <http://www.oracle.com/technology/products/spatial/>, August 2005.
16. A. Orda and R. Rom. Minimum weight paths in time-dependent networks. *Networks*, 21:295–319, 1991.
17. S. Pallottino and M. G. Scutella. Shortest path algorithms in transportation models: Classical and innovative aspects. *Equilibrium and Advanced transportation Modelling (Kluwer)*, pages 245–281, 1998.
18. J. Rasinmäki. Modelling spatio-temporal environmental data. In *5th AGILE Conference on Geographic Information Science*, Palma, Balearic Islands, Spain, April 2002.
19. Shekhar S. and Chawla S. *Spatial Databases: Tour*. Prentice Hall, 2003.
20. D. Sawitzki. Implicit Maximization of Flows over Time. *Technical report, University of Dortmund*, 2004.
21. Dreyfus S.E. An appraisal of some shortest path algorithms. *Operations Research*, 17:395–412, 1969.
22. Y. Sheffi. *Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Method*. Prentice-Hall, Englewood Cliffs, NJ, 1985.

23. 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.
24. S. Stephens, J. Rung, and X. Lopez. Graph data representation in oracle database 10g: Case studies in life sciences. *IEEE Data Engineering Bulletin*, 27(4):61–66, 2004.
25. J. Wardrop. Some theoretical aspects of road traffic research. *Proceedings of the Institution of Civil Engineers*, 2(1), 1952.