1 Introduction

Growing importance of application domains such as location-based services and evacuation planning 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 frequent operations posed on such networks. A frequent query that is posed on such networks is to find 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 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 the frequent computations.

Often dynamic networks have been modeled as time expanded networks, where the entire network is replicated for every time instant. The changes in the network, especially the travel time variations, can be very frequent and for modeling such frequent changes, the time expanded networks would require a large number of copies of the original network, thus leading to network sizes that are too memory expensive. For example, traffic sensors on highway networks send measurement data every 30 seconds. A one-year dataset may need over one million copies of the road network, which itself may have a million nodes and edges for each time instant. Such large sized networks would also result in computationally expensive algorithms.

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.

1.1 An Illustrative Application Domain

Location based services find the geographical location of a mobile device and then provide services based on that location [21]. Most of these services rely heavily on road maps, spatial networks that can change with time. For example, the travel times associated with road segments can change over time. One of the most frequent computations performed on a road network is to identify the shortest route from one point in the network to another. The result of this query will depend on the availability of road segments and the time taken to traverse them; these parameters are time-dependent and hence are the results of the shortest route queries. The results to another frequent query, the nearest neighbor query, can also be time dependent, if computed on a road network; the accessibility of various points in a road network can vary with time, depending on the connectivity of the network at different instants of time. The need to answer such queries in location based services on a spatial network that varies with time makes a simple, efficient model for spatio-temporal networks a necessity.

Such a model is even more critical to applications related to evacuation planning. Route finding here involves
identifying paths in a transportation network to minimize the time needed to move people from disaster-impacted areas to safe locations [16]. One key step in this operation is finding the fastest possible evacuation routes in the network. In computing these routes, it is critical to honor the time dependence of the parameters like travel time (which would change with congestion on roads) and the road capacities. Failure to do so could affect the quality of the solution and even create chaos in an emergency situation.

1.2 Problem Formulation

Spatial networks that show time-dependence serve as the underlying networks for most location based services. 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. We formulate this as the following problem:

Given: The set of frequent queries posed by an application on a spatial network, the pattern of variations of the spatial network with time.

Output: A model which supports efficient and correct algorithms for computing the query results.

Objective: Minimize the storage and computational cost of computation.

Constraints: (1) Edge travel times are positive integers. (2) Edge travel time preserves the FIFO (First-In First-Out) property.

Example: The figures 1 show a network at three instants of time. The travel times on the edges (the number shown on the edges) change with time. For example the edge N2-N1 has a travel time of 1 at the instant \( t = 1 \) and 5 at \( t = 2 \). It can be seen that the topology of the network is also time-dependent. Though the edge N2-N1 is present at \( t = 1 \) and at \( t = 2 \), it is absent at \( t = 3 \). The task is to develop a model that captures the network across time.

1.3 Related Work and Our Contribution

Graph databases [6, 22, 23] have been used widely for the representation of spatial networks. But these models do not address the time-variance of the networks. While efficient and effective techniques for spatial graph analysis, including shortest path computation and route evaluation [4, 7, 9, 10, 11, 12, 13, 18] have been studied, they do not consider the time-dependence of the network. Though research reported in [5, 8, 20] allowed changes in availability of nodes and edges over time, they perform computations over
a snapshot of the network and do not consider the interplay between the edge travel times and existence of edges. Traditionally, time-expanded networks \[1, 3, 15, 19, 14, 17\] have been used to model spatio-temporal networks where the entire graph is replicated for every time instant as shown in Figure 2(a). The

![Time Expanded Graph](image1)

![Time-aggregated Graph](image2)

Figure 2: Time-expanded Graph and Time-aggregated Graph

time expanded graph in this case needed 7 copies of the entire graph to depict the travel time of 4 units from node N3 to the node N4 at time instant 3. Time-expanded graphs, though suitable for optimization algorithms that use linear programming, are not computationally efficient. The changes in the network, especially the travel time variations, can be very frequent and for modeling such changes, the time-expanded networks could require a large number of copies of the original network. Such large sized networks may result in very large storage and computation overheads. Ding [5] proposed a model that addresses the 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.

**Our Contribution:** We propose a model called time-aggregated graphs that represents the time dependence of the network and its parameters. We aggregate the travel times of each edge over the time instants into a time series and keep track of the edges that are present at every instant. 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 identified some challenges that are unique to time-varying graphs, in formulating algorithms such as the shortest path algorithm. We also formulated algorithms for finding the shortest route from one node to another for a given start time and for the best start time. We assume that the travel times of the edges vary with time in a predictable fashion.

**1.4 Scope and Outline of the Paper**

The main focus of the paper is our proposed use of time-aggregated graphs to represent spatial networks and account for the changes that can take place in the network over a period of time. The model requires that the spatial network be a discrete time dynamic network. The rest of the paper is organized as follows. Section 2 discusses the basic concepts of the proposed model and Section 3 proposes algorithms shortest path computations. It also proposes the cost models for these algorithms. In section 4, we summarize the results from our experimental analysis of the model, in comparison with existing models.
2 Basic Concepts

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 a day which leads to a variation in travel time. Thus in spatial routing problems, the underlying spatial network is time-dependent and hence time-variant graphs may be needed to represent them. 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.

**Conceptual Model:** The conceptual model for the time-varying graph will be a graph time-series, which will comprise the set of snapshots of the graphs at successive, discrete instants of time. For example, figure 1 shows a time series for three instants, each graph representing the network at an instant of time.

**Logical Model:** A set of logical operators is listed below.

```plaintext
get(node n1, time t)
- returns the label of the node if it exists at time instant t.

gEdge(node n1, node n2, time t)
- returns the edge from node n1 to node n2 to node n2 at time t.

g_node_earliest_Presence(node n, time t)
- returns the label of the node n and the earliest time after t, at which it is present.

g_edge_earliest_Presence(node n1, node n2, time t)
- returns the edge properties of n1-n2 at the earliest time instant after t, at which it is present.

g_Graph(time t)
- returns the snapshot graph at time t.
```

Table 1 also shows the difference in the behavior of the logical operators for snapshot (when time is specified) and aggregated (time is not specified) views. In snapshot view, the operators return the results with respect to the graph at the given instant of time and aggregated-view-operator results are evaluated on the time-aggregated graphs. For example, the snapshot operator `getEdge(node1,node2,time)` returns the edge properties for the edge from node n1 to node n2, such as edge identifier (if any) and the edge parameters at the given time. While the operator `getEdge(node1,node2,time)` returns the edge properties at the given time, the corresponding time-aggregate operator `get_edge_presence_series(node1,node2)` extracts the time series of the edge properties in addition to the edge identifier (if any).

**Physical Model:** Time-variant graphs raise many challenges. New concepts need to be investigated to represent new 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. Another challenge is the design of efficient and correct algorithms since some of the commonly assumed graph-properties may not hold for spatio-temporal graphs. For example, consider the optimal substructure (a requirement for dynamic programming principle, [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 potential wait at the intermediate node.
Table 1: Examples of logical operators with and without ‘time’ dimension

<table>
<thead>
<tr>
<th>Operator</th>
<th>Snapshot</th>
<th>Time-aggregate</th>
</tr>
</thead>
<tbody>
<tr>
<td>get</td>
<td>get(node, time)</td>
<td>get_node_Presence_series(node)</td>
</tr>
<tr>
<td>getEdge</td>
<td>getEdge(node1, node2, time)</td>
<td>get_edge_Presence_series(node1, node2)</td>
</tr>
<tr>
<td>get_node_Presence</td>
<td>get_node_earliest_Presence(node, time)</td>
<td>get_node_Presence_series(node)</td>
</tr>
<tr>
<td>get_edge_Presence</td>
<td>get_edge_earliest_Presence(node1, node2, time)</td>
<td>get_edge_Presence_series(node, node2)</td>
</tr>
<tr>
<td>get_Graph</td>
<td>get_Graph(time)</td>
<td>get_Graph()</td>
</tr>
</tbody>
</table>

The time-variant graphs are implemented using a time-aggregated graph. It collects the node/edge attributes into a set of time series as illustrated in Figure 2(b) for the snapshots shown in Figure 1. For example, consider the edge N1-N2. Time aggregated graph attaches time-series \((1, 1, \infty)\) to edge N1-N2 to represent travel times of 1 unit during time-instants 1 as well as 2 and absence of edge N1-N2 during time instant 3. A simple example is shown in figure 3 to illustrate the lack of optimal sub-structure. The figure shows a network with three nodes at six time instants. Edges are marked with the travel times. A journey that starts at t=1 at node N1 reaches the node N2 at t=2. Since the edge N2-N3 is not present at t=2, there is a wait at node N2 until t=5. The total travel time from N1 to N3 takes 6 time units. However, if the start time is moved to t=3, the travel time is 4 time units, this being the shortest travel time. But, the prefix path (N1-N2) of this shortest travel time path is not optimal since it takes 2 time units. The shortest travel time from N1 to N2 is 1 time unit starting at t=1. The lack of an optimal substructure in the actual shortest paths may make techniques such as dynamic programming and greedy strategies less effective.

3 Algorithms for Network Computations

The critical step in most queries on a spatio-temporal network is the shortest path computation. Thus the proposed model must include an efficient shortest path algorithm. Here, the shortest path is the route that can be traversed in the shortest time, given the start time at the start node. We assume that there is no cost for waiting at the nodes other than the wait time, and that edge presence is closed for \([t, t + \sigma]\) where \(\sigma\) is the travel time of the edge at the time instant \(t\).

3.1 Algorithm for Shortest Path Computation

As noted earlier, any route computation in a spatio-temporal network must be consistent with the edge presence. Here, the application of a greedy strategy (which is a popular choice in most of the optimization problems) faces a challenge. Not all shortest paths display the optimal sub-structure, which is an essential condition for greedy algorithms to generate an optimal solution. This is clearly illustrated in Figure 4.
Although it can be verified that the route, \( s - N1 - N2 - N4 - u - d \) is an optimal path from \( s \) to \( d \), the route does not display optimal sub-structure since the route from \( s \) to \( u \) following the above path is not optimal (shortest path being, \( s - N1 - N3 - N5 - u \)). Though 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.

**Lemma 1**: If there is an optimal route from \( s \) to \( d \), then there is at least one optimal route from \( s \) to \( d \) that shows optimal sub-structure.

**Proof**: As Figure 4 illustrates, the failure of optimal structure of the shortest path occurs due to a potential wait at the 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 wait at the intermediate node \( u \)) from the optimal path. This would be still the shortest path from \( s \) to \( d \). Otherwise, it would contradict the optimality of the original shortest path.

Lemma 1 enables us to use a greedy approach to compute the shortest path. The algorithm stores the current shortest path travel times to reach every node from the source node. It closes the node with the minimum travel time and updates the travel times of its neighbors. At this step, it uses the operator \( \text{get}_\text{edge}_\text{earliest}_\text{Presence} \) to find the earliest time instant after the arrival at which this edge can be traversed. The updated travel time thus checks for the edge presence and adds the wait time to the travel time, if necessary. The algorithm is similar to Dijkstra’s shortest path algorithm, the key difference being the step that looks for the earliest availability of the edges to the adjacent nodes.

**Computational Complexity**: 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 \( O(deg(v)(\log f_E + \log n)) \). The asymptotic complexity of the algorithm would be \( O(\Sigma_{v \in V}[\text{degree}(v)](\log f_E + \log n)) = O(m(\log f_E + \log n)). \)

The time complexity of the shortest path algorithm based on a time expanded network is \( O(nT \log T + mT) \) [3]. It can be seen that the algorithm based on a time-aggregated graph is faster if \( \log n < T \log T \).

### 3.2 Computation of Best Start Time Shortest Path

The time dependency of the network parameters affects the connectivity and the shortest paths between nodes in a spatial network. As illustrated in figures 5 and 6, the travel time from the node N2 to the node N6 changes with the start time. A path that takes the smallest travel time for source-destination traversal over...
Algorithm 1 Computation of Shortest Path

Input:
1) $G(N,E)$: a graph $G$ with a set of nodes $N$ and a set of edges $E$; define type $p$ positive integer
   Each node $n \in N$ has two properties:
   - $NodePresenceTimeSeries$: series of $p$
   Each edge $e \in E$ has two properties:
   - $EdgePresenceTimeSeries$, $Travel_time(e)$ series: series of $p$
   $\sigma_{u,v}(t)$ is the travel time of edge $(u,v)$ at time $t$
2) $s$: Source node, $s \subseteq N_G$
3) $d$: Destination node, $d \subseteq N_G$

Output: Shortest Route from $s$ to $d$

Method:
$\begin{align*}
  c[s] &= 0; \forall v(\neq s), c[v] = \infty;
  &\text{Insert } s \text{ in priority queue } Q,
  \text{ while } Q \text{ is not empty do } \\
  &u = \text{extract min}(Q); \quad v = \text{get all Successor nodes}(u,c[u]) \quad t = \text{get earliest Presence}(u,v,c[u]);
  \quad \text{if } t + \sigma_{u,v}(t) < c[v] \quad \{ \\
  &\quad \text{update } c[v]. \quad \text{parent}[v] = u;
  &\quad \text{if } v \text{ is not in } Q \quad \text{insert } v \text{ in } Q;
  \}
  \text{update } Q;
\end{align*}$

Output the route from $s$ to $d$.

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 travel time for the same source-destination pair if the travel starts at the “right” time instant. As illustrated in figure 3 that the prefix journeys of a best start time shortest path are not always optimal since some optimal prefix journeys can lead to longer waits at intermediate nodes. The lack of optimal substructure in patient shortest paths rules out the possibility of using greedy strategy in algorithm design. The proposed algorithm attempts to form classes of paths from a node $u$ to a node $w$ based on the observation that a path that start at node $u$ and end at node $w$ at consecutive time instants (subject to the condition of edge presence) will take the same time to traverse the path if there is no wait at the intermediate nodes. We can define a path class as $P = (R, I(S, \delta))$ where $R$ lists the edges on the path, $S$ is the list of start times on the edges and $\delta$ is the interval during which the path is valid. $\delta$ would indicate the time interval during which the edges in the path are available and the path travel time is the same. It also uses the fact that the length of any shortest path is bounded by the number of nodes, $n$, in the time aggregated graph.

Definition: Irrelevant Path: If $P1$ and $P2$ are two journeys that start at node $u$ at time instants $l_{dep1}$ and $l_{dep2}$ respectively and end at node $v$ at $t_{arr1}$ and $t_{arr2}$, the path $P2$ is irrelevant if $l_{dep1} \geq l_{dep2}$ and $t_{arr1} \leq t_{arr2}$. Any path that is not irrelevant is relevant.

The algorithm 2 finds the shortest paths from a given source node $s$ to all other $r$ nodes. At every step, the algorithm adds an edge $uv$ to the list of classes of paths from $s$ to $v$ thereby constructing path classes to $v$ and merges the resulting classes to an existing list while eliminating irrelevant paths. While appending an edge to a class, the algorithm tries to minimize the waits at the intermediate nodes by postponing the start by the amount determined by the presence series of the edges and the potential wait times at the intermediate nodes.

Computational Complexity
Algorithm 2 Computation of Best Start Time

Input:
1) \( G(N,E) \): a graph \( G \) with a set of nodes \( N \) and a set of edges \( E \); define type \( n \) positive integer
   Each node \( n \in N \) has two properties:
   NodePresenceTimeSeries : \( n \)
   Each edge \( e \in E \) has two properties:
   EdgePresenceTimeSeries,
   Travel\_time(e)\_series : series of \( n \)
2) \( s \): Source node, \( s \subseteq G \)
3) \( d \): Destination node, \( d \subseteq G \)
4) \( P(v) \ v \in N \): Set of paths from \( s \) to \( v \);

Output: Patient Shortest Route from \( s \) to \( d \)

Method:
Initialization : \( P(s) = \text{Null} \); \( \forall v \neq s, P(v) = 0 \)
for \( i=1 \) to \( n \) do
   for \( v, \forall v \in N \) {
      \( P[i-1][v] = P[v] \)
      for each edge \( (u,v) \in E \) {
         \( P[v] = \text{append}((u,v), P[u]) \)
         merge \( (P[i-1][v], P[v]) \)
      }
      for each \( v \in N \), find the fastest path

---

Lemma 3: The number of relevant path classes from node \( u \) to node \( v \) that contains \( k \) nodes is bounded by \( 2mf_E \) where \( m \) is the number of edges and \( f_E \) is the edge frequency of the time aggregated graph.

Proof: Consider a path class \( P = (R, I(S, \delta)) \). Let \( e \) be an edge on \( R \) which is traversed at \( t \). If \( \delta \) is the largest integer \( M \), such that the path class is valid, then \( t + \delta \) would be the extremity of the presence interval of the edge \( e \). If \( \delta \) is less than \( M \), then there is an edge on \( R \) such that the lower limit of its presence interval is its traversal time in \( S \). So, corresponding to every presence interval of an edge there can be at most two path classes, one corresponding to each extremity.

Lemma 4: The computational complexity of the algorithm is \( O(nm^2f_E) \)
The append cost for every edge added is proportional to the size of the path class and hence is \( O(mf_E) \). and the cost of the operation of merging the result of the ‘append’ to the existing list is also \( O(mf_E) \). For each node, the total cost would be outdegree times \( O(mf_E) \). and for every increase in the length of the path for all nodes is \( O(m^2f_E) \) and the total cost is \( O(nm^2f_E) \) since the maximum length of a patient path is bounded by \( n \).

3.3 Algorithm for Connectivity

The existence of a valid route from one node to another in a time-aggregated graph is a non-trivial issue since a path in the time-aggregated graph does not always guarantee the existence of a path that is consistent with the edge time series and edge travel times. Figure 5 illustrates this; the node N2 is connected to node N4 for starting time instants 1, 2, 3, 4 (one route being N2 - N5 - N4), and N4 is not accessible from N2 for all time instants after \( T = 4 \).

Computational Complexity: The cost model analysis assumes an adjacency list representation of the graph with two significant modifications. For every node, the node presence time series is stored in the sorted order. Attached to every adjacent node in the linked list are the edge presence series and the travel time series.

For each node dequeued form the queue \( Q \), there is one edge series look up an enqueue operation for each of its adjacent node. The queue operations are \( O(1) \) operations. The time complexity of this step is \( O(\log f_E) \).
Algorithm 3 Procedure Append

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 two properties:
   \( \text{NodePresenceTimeSeries} : nn \)
   Each edge \( e \in E \) has two properties:
   \( \text{EdgePresenceTimeSeries}, \text{TravelTime}(e) \): series of \( nn \)
4) List of path classes from \( s \) to \( u \)

Output: List of classes from \( s \) to \( v \)

Method:
   Initialization : \( P(v) = \text{Null} \);
   while \( P(u) \) not empty
      \( p = \text{dequeue}(P(u)) \)
      \( t_{arr} = \text{Arrival time of path } p \text{ at } u \)
      \( t_{dep} = \text{min} \{ (u,v), t_{arr} \} \)
      if \( t_{arr} + \delta \leq t_{dep} \) then
         \( S = S + \delta \)
         \( \delta = 0 \)
         Add \( (P[u][u,v], S_{t_{arr}}, \delta) \) to \( P[v] \),
         break;
      if \( t_{arr} < t_{dep} \) then
         \( \delta_{incr} = t_{dep} - t_{arr} \)
         \( S = S + \delta \)
         \( \delta = \delta - \delta_{incr} \)
      if \( t_{arr} + \delta \leq t_{5} \) then
         \( \delta_{incr} = t_{dep} + t_{5} - t_{arr} + \delta \)
         Add \( (P[u][u,v], S + \delta_{incr}, \delta - \delta_{incr}) \) to \( P[v] \),
         \( \delta = \delta_{incr} \)
      Add \( (P[u][u,v], S_{t_{arr}}, \delta) \) to \( P[v] \).

The asymptotic complexity of the algorithm would be \( O(\sum_{v \in N}[\text{degree}(v).\log f_E]) = O(m \log f_E)) \).

The time dependency of the network parameters affects the connectivity and the shortest paths between nodes in the network. Figure 6 depicts the connectivity and shortest path travel times for different start time instants at the source node for the example network shown in figure 5. Figure 6(a) illustrates the connectivity of the node N2 to node N4 at instants 1, 2, 3, 4, 5, 6 (these time instants denote the starting times at the node N2). It can be seen that valid routes exist from node N2 to node N4 if the traversal starts at time instants 1, 2, 3, 4 and that the node N4 is unreachable from N2 for time instants 5, 6. It might also be interesting to note that the routes that connect the nodes also change with time. For example, at time instant 1, routes N2-N3-N4, N2-N5-N4 and N2-N3-N5-N4 connect N2 to node N4; at starting time, \( t = 4 \), only N2-N5-N4 is available.

As shown in figure 6(b), the shortest path routes and the travel times are also dependent on time. Consider the shortest path from node N1 to node N6. The shortest path from node N1 to N6, for starting time \( t = 1 \) is N1-N4-N6 and travel takes 5 units of time (reaches the destination node at \( t = 6 \)). The route remains the same for start times \( t = 2, 3 \), but the travel time changes to 4 units and 3 units respectively. At time \( t = 4 \), the route N1-N4-N6 is no longer available and the shortest route changes to N1-N2-N5-N6 with a total travel time of 6 units. This shows that the shortest paths in a time-dependent network vary with time.
Algorithm 4 Connectivity Algorithm

Input:
1) \( G(N,E) \): a graph \( G \) with a set of nodes \( N \) and a set of edges \( E \);
   define type \( p \) positive integer
   Each node \( n \in N \) has two properties:
   \( \text{NodePresenceTimeSeries} \): series of \( p \)
   Each edge \( e \in E \) has two properties:
   \( \text{EdgePresenceTimeSeries,Travel_time(e)series} \): series of \( p \)
   \( \sigma_{u,v}(t) \) is the travel time of edge \( (u,v) \) at time \( t \)
2) \( s \): Source node, \( s \subseteq N_G \);
3) \( d \): Destination node, \( d \subseteq N_G \);

Output: A route from \( s \) to \( d \), if exists; else returns FALSE.

Method:
 Initialization;
   Add \( s \) to \( Q \); \( found = \) false;
   for each node \( v \in N_G \) do {
     \( \text{arr_time}[v] = 0 \);
   }
   while \( found = \) FALSE or \( Q \) not empty do {
     \( u = \text{dequeue}(Q) \);
     \( v = \text{get_all_Successor_nodes}(u, \text{arr_time}[v]) \)
     Add \( v \) to the \( Q \);
     \( t = \text{get_earliest_presence}(u, v, c[u]) \);
     if \( t \neq \infty \) {
       \( \text{arr_time}[v] = t + \sigma_{u,v}(t) \)
       \( \text{parent}[v] = u \)
     }
     if \( v = d \), \( \text{FOUND} = \) TRUE;
   }
 Output the route from \( s \) to \( d \).

4 Experimental Design

Figure 7 illustrates the experimental design. The core part consists of comparing the two physical models to represent spatio-temporal networks: Time Expanded Graph and Time Aggregated Graph. Experiments will be performed to compare the two models in terms of computation time to perform same functions. These
functions includes finding the shortest path at a time instant, and finding the shortest path with best start time. Real dataset of Minneapolis Central Business District consisting of static road networks with weights was used. The real dataset was modified was modified to add a temporal dimension to it. The status of the elements in the road networks (e.g. path between two nodes) and their weights for a period of time were inserted randomly. All experiments will be performed on Sun SunBlade with 500 MHz and 384 MB.
References


