Facilitating Eco-Routing via Spatial Big Data: A Case-Study on Temporally-Detailed Roadmaps

Michael R. Evans
Dev Oliver
Venkata M.V. Gunturi
Shashi Shekhar

Department of Computer Science
University of Minnesota
Minneapolis, MN

1 Introduction

Routing and navigation services are a set of ideas and technologies that transform lives by understanding the physical world, knowing and communicating relations to places in that world, and navigating through those places. From Google Maps [1] to consumer Global Positioning System (GPS) devices, society is benefiting immensely from routing services. Scientists use GPS to track endangered species to better understand animal behavior, and farmers use GPS for precision agriculture to increase crop yields while reducing costs. We've reached the point where a hiker in Yellowstone, a biker in Minneapolis, and a taxi driver in Manhattan know precisely where they are, their nearby points of interest, and how to reach their destinations.

Increasingly, however, the size, variety, and update rate of spatial datasets exceed the capacity of commonly used spatial computing and database technologies to learn, manage, and process the data with reasonable effort. We believe that this data, which we call Spatial Big Data (SBD), represents the next frontier in routing services. Examples of emerging SBD datasets include temporally detailed (TD) roadmaps that provide speeds every minute for every road-segment, GPS track data from cell-phones, and engine measurements of fuel consumption, greenhouse gas (GHG) emissions, etc. Harnessing SBD has transformative potential. For example, a 2011 McKinsey Global Institute report estimates savings of “about $600 billion annually by 2020” in terms of fuel and time saved [2] by helping vehicles avoid congestion and reduce idling at red lights or left turns. Preliminary evidence for the transformative potential includes the experience of UPS, which saves millions of gallons of fuel by simply avoiding left turns (Figure 1(a)) and associated engine-idling when selecting routes [3]. Immense savings in fuel-cost and GHG emission are possible if other fleet owners and consumers avoided hot spots of idling, low fuel-efficiency, and congestion. In this chapter we discuss ideas likely to facilitate ‘eco-routing’ to help identify routes which reduce fuel consumption and GHG emissions, as compared to traditional route services reducing distance traveled or travel-time. It has the potential to significantly reduce US consumption of petroleum, the dominant source of energy for transportation (Figure 1(b)). It may even reduce the gap between domestic petroleum consumption and production (Figure 1(c)), helping bring the nation closer to the goal of energy independence [4].

However, SBD raises new challenges for the state of the art in spatial computing for routing services. First, it requires a change in frame of reference, from a global snapshot perspective to the perspective of the individual object traveling through a road network. Second, SBD increases the impact of the partial nature of traditional route query specification. It significantly increases computation cost due to the tremendous growth in the set of preference functions beyond travel-distance and travel-time to include fuel consumption, GHG emissions, travel-times for thousands of possible start-times, etc. Third, the growing diversity of SBD sources makes it less likely that single algorithms, working on specific spatial datasets, will be sufficient to discover answers appropriate for all situations.

The rest of this chapter is organized as follows: Section 2 discusses traditional routing services. Section 3
2 Traditional Routing Services

Traditional routing systems utilize digital road maps [5–8]. Figure 2(a) shows a physical road map and Figure 2(b) shows its digital, i.e., adjacency graph-based, representation. Road intersections are often modeled as vertices and the road segments connecting adjacent intersections are represented as edges in the graph. For example, the intersection of SE 5th Ave and SE University Ave is modeled as node N1. The segment SE 5th Ave between SE University Ave and SE 4th Street is represented by the edge N1-N4. The directions on the edges indicate the permitted traffic directions on the road segments. Digital roadmaps also include attributes for road-segments (e.g., center-lines, road-classification, speed-limit, historic speed, historic travel time, address-ranges, etc.) Figure 2(c) shows a tabular representation of nodes and edges in the digital road map. Additional attributes are shown in the edge table. For example, the entry for edge E1 (N1-N2) shows its speed and distance. Such datasets include roughly 100 million (10^8) edges for the roads in the U.S.A. [6]. Turn restrictions are difficult to model in either the node or edge table, as they represent relationships among edges. Commercial approaches [8] annotate nodes with turn information for procedural interpretation during graph traversals. Other limitations include lack of modeling of synchronized traffic lights and differences in delays across left turns and right turns.

Figure 2: Current representation of road maps as directed graphs with scalar travel time values.

Route services [9, 10] provide two basic types of capabilities [11]. The first deals with determination of a best route given a start location, end location, optional waypoints, and a preference function. Here, the choice of preference function could be: fastest, shortest, easiest, pedestrian, public transportation, avoid locations/areas, avoid highways, avoid tollways, avoid U-turns, and avoid ferries. Route finding is often based on classic shortest path algorithms such as Dijkstra’s [12], A* [13–17], hierarchical [18–40], materialization [36,
Spatial Big Data for Next-Generation Routing Services

SBD raises significant new challenges for next-generation routing services. First, it requires a change in frame of reference from a snapshot perspective to the perspective of the individual traveling through a transportation network. For instance, new temporally detailed (TD) roadmaps provide historical travel-time (or speed) for every road-segment for every distinct minute of a week. A traveler moving along a chosen path in a TD roadmap would experience a different road-segment and its historical speed as well as traversal-time at different time-intervals, which may be distinct from the start-time. In addition, the traveler may experience synchronized traffic lights and different delays for left turns, right turns, and going straight, which are difficult to represent and compute with traditional road intersection (node table, edge table) models of roadmaps (Figure 2), and their temporal generalizations such as snapshots, TEG, and time aggregated graphs. In this subsection we describe examples of spatial big data.

**Gps Track Data:** Gps trajectories are becoming available for a larger collection of vehicles due to rapid proliferation of cell-phones, in-vehicle navigation devices, and other GPS data-logging devices [106] such as those distributed by insurance companies [107]. Such GPS tracks allow indirect estimation of fuel efficiency and GHG emissions via estimation of vehicle-speed, idling, congestion, synchronized traffic lights, and turn delays. They also make it possible to offer personalized route suggestions to users to reduce fuel consumption and emissions. For example, Figure 3 shows 3 months of GPS track data from a commuter with each point representing a GPS record taken at 1 minute intervals, 24 hours a day, 7 days a week. As can be seen, 4 alternative commute routes are identified between home and work from this dataset. These routes may be compared for idling which are represented by darker (red) circles. Assuming the availability of a model to estimate fuel consumption from speed profile, one may even rank alternative routes for fuel efficiency.

In recent years, consumer GPS products [106, 108] have been evaluating the potential of this approach. In addition GPS track mining [109–132] has also been used to create a subset of TD roadmaps connecting

![Figure 3: A commuter’s GPS tracks over three months reveal preferred routes. (Best viewed in color)](image)
landmark locations and improve the quality of recommendation from web based routing services leveraging traditional roadmaps. A key hurdle is the dataset size, which can reach $10^{13}$ items per year given constant minute-by-minute resolution measurements for all 100 million US vehicles.

**Temporally Detailed (TD) Roadmaps:** New datasets from companies such as NAVTEQ [6] use probe vehicles (e.g., GPS tracks) and highway sensors (e.g., loop detectors) to compile travel time information across road segments for all times of the day and week at fine temporal resolutions (seconds or minutes). While, traditional roadmaps (Figure 2(a)) have only one scalar value of speed for a given road segment (e.g., EID 1), TD roadmaps may potentially list speed/travel time for a road segment (e.g., EID 1) for thousands of time points (Figure 4(a)) in a typical week. This level of detail allows a commuter to compare alternate start-times in addition to alternate routes. It may even allow comparison of (start-time, route) combinations to select distinct preferred routes and distinct start-times. For example, route ranking may differ across rush hour and non-rush hour periods and in general across different start times. However, TD roadmaps are big and their size may exceed $10^{13}$ items per year for the 100 million road-segments in the US when associated with per-minute values for speed or travel-time. Thus, industry is using speed-profiles, a lossy compression based on the idea of a typical day of a week, as illustrated in Figure 4(b), where each (road-segment, day of the week) pair is associated with a time-series of speed values for each hour of the day.

![Figure 4: Temporally Detailed Roadmaps using Historical Speed Profiles. (Best viewed in color)](image)

In the near future, values for the travel time of a given edge and start time will be a distribution instead of scalar. We also envision richer temporal detail on many preference functions such as fuel cost, pot-holes [133], crime reports [134], and social media reports of events on road networks [135].

**Spatio-Temporal Engine Measurement Data:** Many modern fleet vehicles include rich instrumentation such as GPS receivers, sensors to periodically measure sub-system properties [136–141], and auxiliary computing, storage and communication devices to log and transfer accumulated datasets. Engine measurement datasets may be used to study the impacts of the environment (e.g., elevation changes, weather), vehicles (e.g., weight, engine size, energy-source), traffic management systems (e.g., traffic light timing policies), and driver behaviors (e.g., gentle acceleration/braking) on fuel savings and GHG emissions. These datasets may include a time-series of attributes such as vehicle location, fuel levels, vehicle speed, odometer values, engine speed in revolutions per minute (RPM), engine load, emissions of greenhouse gases (e.g., CO2 and NOX), etc. Fuel efficiency can be estimated from fuel levels and distance traveled as well as engine idling from engine RPM. These attributes may be compared with geographic contexts such as elevation changes and traffic signal patterns to improve understanding of fuel efficiency and GHG emission. For example, Figure 5 shows heavy truck fuel consumption as a function of elevation from a recent study at Oak Ridge National Laboratory [142]. Notice how fuel consumption changes drastically with elevation slope changes. Fleet owners have studied such datasets to fine-tune routes to reduce unnecessary idling [143, 144]. It is tantalizing to explore the potential of this dataset to help consumers gain similar fuel savings and GHG emission reduction. However, these datasets can grow very large. For example, measurements of 10 engine variables, once a minute, over the 100 million US vehicles in existence [145,146], may have $10^{14}$ data-items per year.
4 Modeling the Traveler’s Perspective

An important aspect of navigation and routing services is to model the traveler’s frame of reference. In other words, candidate routes should be evaluated from the perspective of a person moving through a transportation network. In other fields, such a frame of reference is defined as a Lagrangian frame of reference [147]. Here we discuss conceptual, logical, and physical models for database systems based on a Lagrangian frame of reference to facilitate time-dependent routing queries.

**Conceptual Model:** We present a Lagrangian representation of a roadmap through a model called a time-expanded graph (TEG) [148]. To illustrate, Figure 6(a) provides a snapshot graph representation of a simple roadmap with four consecutive time-instants. In the figure, the travel-time for edge A to C is 2 for time = 1 but decreases to 1 for time = 2, 3, and 4. An equivalent TEG is shown in Figure 6(b). Note that all nodes are replicated across time points. For example, node A at time 1, 2, 3, and 4 in Figure 6(a) is represented as a set of nodes (A1, A2, A3, and A4, respectively) in Figure 6(b). Edge AC in Figure 6(a) is represented as a set of edges (i.e., A1 - C3, A2 - C3, A3 - C4, ...) in Figure 6(b), encoding the time-series of travel-times. A1 - C3 indicates that the travel time for edge AC is 2 at start-time 1. Similarly, A2 - C3 indicates a travel-time of 1 for start-time 2.

**Table 1: Snapshot and Lagrangian Travel-Time**

<table>
<thead>
<tr>
<th>Route</th>
<th>Start t = 1</th>
<th>Start t = 2</th>
<th>Start t = 3</th>
<th>Start t = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A - B - D</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>A - C - D</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time Step</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A1</td>
<td>A2</td>
<td>A3</td>
<td>A4</td>
</tr>
<tr>
<td>B</td>
<td>B1</td>
<td>B2</td>
<td>B3</td>
<td>B4</td>
</tr>
<tr>
<td>C</td>
<td>C1</td>
<td>C2</td>
<td>C3</td>
<td>C4</td>
</tr>
<tr>
<td>D</td>
<td>D1</td>
<td>D2</td>
<td>D3</td>
<td>D4</td>
</tr>
</tbody>
</table>

Unlike snapshot graphs, time-expanded graphs model the actual travel time experience of a moving object. For example, when traveling from node A to node D via node B starting at time = 1, the object will reach node B at time = 2, since edge A - B has a travel-time of 1 at start-time = 1. The object will reach node D at time = 4, since travel-time for edge B - D is 2 at time = 2. Thus, the total travel time is 3, which is different than the total travel-time of 2 based on the snapshot graph for time = 1. It also models
wait, i.e., lack of motion. For example, edge A1 - A2, A2 - A3, etc., indicate staying at the same location over time. Table 1 displays the travel time for various routes via both Lagrangian and Snapshot models for Figure 6.

**Logical Model:** In a logical data model, a traveler’s frame of reference may be represented as a collection of abstract data types and operators. Table 2 illustrates the abstract data types and operations for the snapshot and Lagrangian representations. Focusing on the snapshot representation first, each type (node, edge, route) has operations associated with it. `getNode(A, 1)` will return node A at time 1 and its associated attributes (e.g., turn restrictions) for the snapshot graph in Figure 6(a). `getEdge(A, B, 1)` will return the travel time associated with the edge AB in snapshot 1. The route type has two operations: the `getRoute(A, D, 1)` operation will return the shortest path from A to D on the graph shown in snapshot 1 (e.g., A-B-D). `evalRoute(Route, 1)` takes a route as input (e.g., the output from `getRoute`) and calculates the travel time. For example, the travel time for ABD in snapshot 1 is 2 time instants.

Using a traveler’s perspective requires new data types and operators. For example, an initial Lagrangian-based set is presented in Table 2. An L-Node (e.g., A1 in Figure 6(b)) represents a node in a time-expanded graph. An L-Edge represents an edge in the time-expanded graph (e.g., A1-C3 in Figure 6(b)). L-Routes (e.g., A1-C3-D4) represent Lagrangian routes in a time-expanded graph. `get – L – Node(A1, 1)` returns a Lagrangian node specified by the unique ID (A1) and the start time 1 along with its associated properties (e.g., turn restrictions) from the TEG in Figure 6(b). Similarly, `get – L – Edge(A1, C3, 1)` returns L-edge A1 - C3 at start time 1 and its associated travel time. `get – L – Route(A2, D4, 2)` returns a Lagrangian shortest path from the source node to destination node. In this case, the route returned is A2 - C3 - D4. Calling `evalRoute(A2C3D4, 2)` will calculate the Lagrangian travel time of the route, in this case, 3 time steps.

<table>
<thead>
<tr>
<th>Snapshot Abstract Data Type</th>
<th>Lagrangian Abstract Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types</td>
<td>Operations at Snapshot</td>
</tr>
<tr>
<td>Node</td>
<td><code>getNode(node, snapshot)</code></td>
</tr>
<tr>
<td>Edge</td>
<td><code>getEdge(node-1, node-2, snapshot)</code></td>
</tr>
<tr>
<td>Route</td>
<td><code>getRoute(node-1, node-2, snapshot)</code></td>
</tr>
<tr>
<td></td>
<td><code>evalRoute(route, snapshot)</code></td>
</tr>
</tbody>
</table>

**Physical Model:** The problem of storing Spatial Big Data, such as TD roadmaps, can be formalized as follows: Given a TD roadmap and a set of Lagrangian operations, find a storage scheme that minimizes the I/O costs of operations. The input to this problem is a TD roadmap (e.g., Figure 6(b)) and a set of operations, e.g., listed in Table 2. Current methods for storing TD roadmaps have focused on orthogonal partitioning, e.g., snapshot and longitudinal partitioning. Snapshot partitioning groups data by common time step; longitudinal partitioning groups data based on the object (e.g., a node and its entire time-series-valued properties). For the TEG from Figure 6(b), snapshot partitioning returns groups (A1, B1, C1, D1), (A2, B2, C2, D2), etc, and longitudinal partitioning will prefer groups of (A1, A2, A3, A4), (B1, B2, B3, B4), etc, as shown in Figure 7(a) and Figure 7(b), respectively, via distinct colors.

Figure 7: Partitioning of Spatio-Temporal Networks. (Colors encode distinct disk pages)

More recent techniques propose Lagrangian Partitioning, which groups data based upon Lagrangian traversals using a min-cut graph partitioning [149] algorithm to create partitions by minimizing the cut
Lagrangian edges. This results in groups of (A1, B2, C3, D4), (A2, B3, C2, D3), etc, as shown in Figure 7 (c). In this simple example, each group fits inside a single disk-page. In general, a group may need multiple disk-pages.

The benefit of Lagrangian Partitioning may be seen in the operator eval-L-Route(ACD,1). With Snapshot or Longitudinal partitioning, whenever an edge is traversed, (e.g., A1 to C3 and C3 to D4), a disk I/O is needed to retrieve the data page containing the record for the next node. This results in retrieval of four disk pages. However, LCP requires only one data page as all relevant sub-node records are collocated on the same data page.

5 Case Study: Routing on Temporally Detailed Roadmaps

Spatial Big Data magnifies the already partial and ambiguous nature of a traditional routing query. This is because a typical routing query specified by a start location and a destination may result in multiple answers. The apparent ambiguity in the shortest path query is clearly visible in the multiple candidate routes returned for a single start and destination pair in common web-based routing services like [1,150,151]. For example, consider Figure 8(a), which shows two candidate shortest travel time routes from the University of Minnesota to MSP International Airport. This ambiguity increases tremendously with the availability of SDB datasets, resulting in increased computational costs due to re-computation that may be necessary for thousands of possible start times. Let us consider this issue in the context of TD roadmaps via the all start-time Lagrangian shortest paths (ALSP) problem.

Given a TD roadmap, a source, a destination, and a start-time interval, an ALSP problem determines a route collection which includes the shortest path for every start time in the interval. The ALSP output includes both the shortest paths and the corresponding set of time instants when the paths are optimal. For example, consider the problem of determining the shortest travel time path between the University of Minnesota and the MSP International Airport during the interval from 7:00AM to 10:30AM. Figure 8(a) shows two different routes between the University and the Airport. The 35W route is preferred outside rush-hour, whereas the route via Hiawatha Avenue is preferred during rush-hour (i.e., 7:00AM - 9:30AM) (see Figure 8(b)). Thus, the ALSP route collection may be a set of two routes (one over 35W and one over Hiawatha Avenue) and their corresponding time intervals.

Computing ALSP is computationally challenging for three reasons. First, many links in the network may violate the property of first-in-first-out (FIFO) behavior. This is illustrated in Table 3, which shows the flight schedule for Delta airways [152] between Minneapolis and Austin, TX. Here, the travel time at 8:30 is 6 hrs 31mins, whereas waiting 40 minutes for the 9:10 flight would yield a quicker route: 2 hours and 51 minutes. This violation of first-in-first-out (FIFO) is called non-FIFO behavior. Surface transportation networks such as road networks also exhibit such behavior. For example, UPS [3,153] minimizes the number of left turns in their delivery routes during heavy traffic conditions. This leads to faster delivery and fuel savings. Second, the ranking of alternate paths between any particular source and destination pair in the network is not
stationary. In other words, the optimal path between a source and destination for one start time may not be optimal for other start times. In our previous example of shortest route between the university and airport, different routes were optimal at different times. The principle of stationarity states that, if two reward sequences $R_1, R_2, R_3, \ldots$ and $S_1, S_2, S_3, \ldots$ begin with the same reward then the sequences should be preference ordered the same way as the sequences $R_2, R_3, \ldots$ and $S_2, S_3, \ldots$ [154, 155]. This means that, in shortest route computation, if two journeys $e_1, e_2, e_3, \ldots$ and $f_1, f_2, f_3, \ldots$ start at the same node, then the preference order of the journeys should not change for another start time. This lack of stationarity in TD roadmaps eliminates the possibility of using dynamic programming based techniques. Lastly, due to the potentially large and ever growing sizes of SBD such as TD roadmaps, efficient computational methods need to developed which can scale to large datasets.

Related work in the area of route collections with minimization of travel time as the preference function [156–158] are limited due to FIFO assumptions. Other related work in the area of route collections include skyline based techniques [159] which assume independence among dimensions and do not incorporate a Lagrangian frame of reference. The challenges of ALSP may be addressed using the ideas of earliest arrival-time transformation and critical time points [160].

**Earliest arrival-time transformation (EAT):** The first challenge for solving ALSP is the non-FIFO behavior of the network. One way to address this is to converting travel information associated with an edge into earliest arrival time (at destination) information [161,162]. This is a two step process. First, the travel time information is converted into arrival time information. Second, the arrival time information is converted into earliest arrival time information. The second step captures the possibility of arriving at an end node earlier by waiting (non-FIFO behavior). For example, consider again Table 3 which shows the flight schedule between Minneapolis and Austin, TX. Here, the result of the first step, shown in the fourth column, was obtained by adding the start time with the travel time (flight time). The arrival time information (in the fourth column) is then scanned to capture any benefits associated with waiting. For example, we can observe that a quicker path for start time 8:30am can be obtained by waiting for 40 minutes. The last column of the table shows the result of the second step.

<table>
<thead>
<tr>
<th>Start Time</th>
<th>Route</th>
<th>Travel Time</th>
<th>Arrival Time</th>
<th>Earliest Arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:30</td>
<td>via Detroit</td>
<td>6 hrs 31 mins</td>
<td>15:01</td>
<td>12:01</td>
</tr>
<tr>
<td>9:10</td>
<td>direct flight</td>
<td>2 hrs 51 mins</td>
<td>12:01</td>
<td>12:01</td>
</tr>
<tr>
<td>11:00</td>
<td>via Memphis</td>
<td>4 hrs 30 mins</td>
<td>15:38</td>
<td>15:38</td>
</tr>
<tr>
<td>11:30</td>
<td>via Atlanta</td>
<td>6 hrs 28 mins</td>
<td>17:58</td>
<td>17:21</td>
</tr>
<tr>
<td>14:30</td>
<td>direct flight</td>
<td>2 hrs 51 mins</td>
<td>17:21</td>
<td>17:21</td>
</tr>
</tbody>
</table>

**Critical time points:** A naive approach for solving the non-FIFO ALSP problem involves determining the shortest path for each start time in the interval using techniques [17,60,162–167] developed for single-start-time shortest paths. This leads to redundant re-computation of the shortest path across consecutive start times sharing a common solution. Some efficiencies can be gained using a time series generalization of a label-correcting algorithm [163]. However, this approach still entails large number of redundant computations, which may be reduced [160]. For instance, consider again the problem of determining the shortest path between the University of Minnesota and MSP international airport over a time interval of 7:30am through 10:30am. Figure 8(b) shows the preferred paths at some time instants during this time interval, and Figure 8(c) shows the travel-times for the candidate paths for all start-times during this interval. The dotted lines across bars are drawn for ease of understanding only.

As can be seen, the Hiawatha route is faster for times in the interval [7:30am 8:30am], whereas the 35W route is faster for times in the interval [9:30am 10:30am]. This shows that the shortest path changed at some instant inside the interval [8:31 9:30]. For sake of simplicity we assume that the shortest path changed at 9:30am. We define this time instant as a critical time point. Critical time points can be determined by computing the earliest intersection points between the functions representing the total travel time of paths. For example, the earliest intersection point of the Hiawatha route was at 9:30am (with 35W route function). Therefore, it would be redundant to recompute shortest paths for all times in interval [7:31am 9:29am] and [9:31am 10:30am] since the optimal path for times within each interval did not change. This approach is
Thus, critical time points inspire a divide and conquer based approach to handle network non-stationarity. In this approach we divide the time interval over which the network exhibits non-stationarity into smaller intervals which are guaranteed to show stationary behavior. These intervals are determined by computing the critical time points, that is, the time points at which the cost functions representing the total cost of the path as a function of time intersect. Now, within these intervals, the shortest path can be easily computed using a single run of a dynamic programming (DP) based approach [163]. In our previous example of determining the shortest paths between the university and the airport over an interval of [7:30am 10:30am], the critical time point was 9:30am. This created two discrete sub-intervals [7:30 9:29] and [9:30 10:30]. Now, we can compute the ALSP using two runs of a DP based algorithm [163] (one on [7:30 9:29] and another on [9:30 10:30]). The advantages of critical time point based approaches are two fold. First, they allow us to compress the result set of the query to a set of routes and their corresponding time intervals instead of a single route for every start time. Second, they are computationally more efficient since they avoid redundant re-computation across the start-times sharing a common solution.

A key challenge in designing algorithms based on critical time points is to minimize the amount of time needed to compute them while ensuring correctness and completeness. Critical time points can be computed using one of the two strategies; (a) Precomputing based method, (b) On-the-fly based method. In a precomputing based method all the candidate solutions (routes) are enumerated and compared to determine the best route for each start time. This approach would become a major bottleneck in the case of TD roadmaps, as there can be exponential number of candidate routes between any source-destination pair. In contrast, an on-the-fly technique computes the critical time points on the fly while exploring a small subset of candidate routes. In other words, while computing the shortest path for one start-time, it determines the next time instant when recomputation needs to start. In the University-MSP airport example, this means computing the shortest path at 7:30am finds 9:30am as the next critical time point. A brief description of the On-the-fly technique for computing critical time points is given next.

Before describing the details on-the-fly technique, we talk about necessary tools required for any method of computing the critical time points. In order to compute the critical time points (where path ranking changes) we need to model the total cost of the route as a function of time. This function is called Lagrangian Path Function. A lagrangian path function associated with a candidate route denotes earliest arrival time at the end point of the route. For instance, consider a candidate route <A,C,D> with an associated lagrangian path function [4 4 6 6] for times $t = 0, 1, 2, 3$. This means that, if a traveler starts at $A$ at time $t = 0$, he/she would reach $D$ at time $t = 4$. Now, the critical time points can be computed by determining the intersection points between the lagrangian path functions (see Figure 9(c)).

On-the-fly technique for Critical time points: This primarily involves incorporating lagrangian path functions into a refine and prune based search strategy of exploring the candidate route space. We now provide an example of this using Dijkstra’s algorithm. Dijkstra’s algorithm is a known refine and prune based search
Dijkstra’s algorithm initializes by associating labels (denoting the distance from source) to each of the nodes in the graph. The source node is inserted into the priority queue (ordered on the distance label). In each iteration, the node with the least distance label is expanded and distance labels to its neighbors are either inserted into the priority queue or updated (if already present). This process continues until the destination node is expanded.

Now, in order to incorporate Dijkstra’s algorithm into the critical time point framework, lagrangian path functions (instead of distance labels) are used to denote the distance from the source. Thus, the priority queue contains lagrangian path functions instead of distance labels. Recall that a lagrangian path function stores the arrival time for several start times, but a priority queue can be ordered only on one time instant. Therefore, it is necessary to process the start-times in a sequence, i.e., begin with the first start time and continue until the shortest path for each time is found. While computing the shortest path for one time instant, the information in the lagrangian path functions is used to decide the next start time. Specifically, whenever a node is expanded for one start time (as priority queue is ordered only on the one start time), its lagrangian path function is compared with the others in the queue to compute the next start time (critical time point) when this decision (of closing the node) is no longer valid. The rest of the procedure is similar to Dijkstra’s in the sense that whenever a node is expanded, the path functions to this neighbors are included in the priority queue. A trace of this procedure is given in Figure 10.

6 Summary

Increasingly, location-aware datasets are of a size, variety, and update rate that exceed the capability of spatial computing technologies. This chapter discussed some of the emerging challenges posed by such datasets, referred to as Spatial Big Data (SBD). SBD examples include trajectories of cell-phones and GPS devices, temporally detailed (TD) road maps, vehicle engine measurements, etc. SBD has the potential to transform society. A recent McKinsey Global Institute report estimates that personal location data could save consumers hundreds of billions of dollars annually by 2020 by helping vehicles avoid congestion via next-generation routing services such as eco-routing. Eco-routing may leverage various forms of SBD to compare routes by fuel consumption or greenhouse gas (GHG) emissions rather than total distance or travel-time.

However, the envisaged SBD-based next-generation routing services pose several challenges for current routing techniques. First, SBD requires a change in frame of reference, moving from a global snapshot perspective to the perspective of an individual object traveling through a transportation network. Second, SBD magnifies the impact of partial information and ambiguity of traditional routing queries specified by a start location and an end location. For example, traditional routing identifies a unique (or a small set of) route(s), given historical and current travel-times. In contrast, SBD may identify a much larger set of solutions, e.g., one route each for thousands of possible start-times in a week, significantly increasing computational costs. Third, SBD challenges the assumption that a single algorithm utilizing a specific dataset is appropriate for all situations. The tremendous diversity of SBD sources substantially increases the diversity of solution methods. For example, methods for determining fuel efficient routes leveraging engine measurement and GPS track datasets may be quite different from algorithms used to identify minimal
travel-time routes exploiting temporally detailed roadmaps. This paper discusses one possible algorithm for temporally-detailed roadmaps that aim to reduce idling and fuel consumption. Newer algorithms may emerge as new SBD becomes available, creating the need for a flexible architecture to rapidly integrate new datasets and associated algorithms.

References


