Generating Time-Varying Road Network Data Using Sparse Trajectories

Elif Eser[1], Furkan Kocayusufoğlu[2], Bahaeddin Eravcı[3], Hakan Ferhatosmanoğlu[4], Josep L. Larriba-Pey[5]

[1, 3, 4]Dept. of Computer Engineering, Bilkent University, Ankara, Turkey
[2]Dept. of Computer Science University of California, Santa Barbara, California, USA
[5]DAMA-UPC, Universitat Politecnica de Catalunya, BarcelonaTech, Barcelona, Spain
E-mail: [1]elif.eser@bilkent.edu.tr, [2]furkankyo@gmail.com, [3]beravci@gmail.com, [4]hakan@cs.bilkent.edu.tr, [5]larri@ac.upc.edu

***Abstract*—While research on time-varying graphs has attracted recent attention, the research community has limited or no access to real datasets to develop effective algorithms and systems. Using noisy and sparse GPS traces from vehicles, we develop a time-varying road network data set where edge weights differ over time. We present our methodology and share this dataset, along with a graph manipulation tool. We estimate the traffic conditions using the sparse GPS data available by characterizing the sparsity issues and assessing the properties of travel sequence data frequency domain. We develop interpolation methods to complete the sparse data into a complete graph dataset with realistic time-varying edge values. We evaluate the performance of time-varying and static shortest path solutions over the generated dynamic road network. The shortest paths using the dynamic graph produce very different results than the static version. We provide an independent Java API and a graph database to analyze and manipulate the generated time-varying graph data easily, not requiring any knowledge about the inners of the graph database system. We expect our solution to support researchers to pursue problems of time-varying graphs in terms of theoretical, algorithmic, and systems aspects. The data and Java API are available at: http://elif.eser.bilkent.edu.tr/roadnetwork.**

***Index Terms*—time-varying graphs, data generation, dynamic road networks, time dependent shortest paths, graph databases**

I. Introduction

Dynamic management and analysis of road networks is an essential task for smart city applications such as traffic management and location based services (LBSs). Road networks are spatial graphs with vertices representing the geo-locational points and edges representing the streets/roads between the vertices. Most traditional graph algorithms on road networks assume that the edges have static attributes, such as the length of the road and the speed limit. A more enhanced representation is a time-varying road network that models changing traffic information via a graph with edges having a time-series of values, rather than a single aggregate value.

Computing shortest paths over time-dependent road networks is shown to be polynomially solvable with Dijkstra based solutions adapted for these graphs [1] [2]. Route planning over time-dependent graphs has also appeared in the literature aiming at reducing traffic jams [3]. Beyond, personalized route planning[4] arises by considering driver's preferences. Research on time-varying graphs has a significant projection in shortest path and route planning algorithms. To develop and compare algorithms, systems, and analysis of traffic for dynamic road networks, the research community needs publicly available time-varying graph datasets, with edge weights varying over time based on realistic patterns, and associated APIs to manage and analyze such datasets. In this paper, we present our methodology to generate a realistic time-varying graph for road networks and share our dataset and an associated graph management tool to help research on time-varying graphs and dynamic road networks.

Major service providers such as Google, Yandex, and TomTom have the ability to observe real time traffic in certain regions and can update their underlying road network's edge weights accordingly. However, most users and researchers do not have access to such dynamic updates. The datasets used in the literature are usually combinations of real maps with synthetically generated travel time-series [1] or real data collected over a limited amount of time [3] [4]. This makes the comparison of algorithms difficult because they usually have data dependencies or these datasets are not publicly available via an API or web service.

Employing GPS traces to build or exploit road networks has also been studied in the literature [5] [6]. In [5], GPS data are utilized to generate a road network without any prior information about the network topology. In [6], GPS data are employed with a road network for traffic and travel time estimation of the paths by using probabilistic model based approaches.

In this piece of work, by using real yet sparse and noisy GPS trajectories, we build a realistic time-varying graph dataset with different travel times for each time slot for its edges. We use the sparse trajectories found in the Floating Car Dataset of Telecom Italia [7]. After mapping the trajectories to the road network, we use statistical inference methods to estimate the missing values to generate a time-varying graph. We analyze the frequency content of nearly-complete edges using frequency analysis tools. We observe that the time series of interest are band-limited and most of the signal power is in the low-pass region, i.e., the time series are slow varying signals. We complete the missing data using minimum travel time for the respective edge and random sample drawn from a normal distribution, with parameters equal to the signal itself. We make the preference of these two values based on Nyquist-Shannon sampling theorem to counter aliasing. After padding the signal we use the fact that the signals are expected to be band-limited to smooth the padded time series such that the resultant signal is of the same model as the expected signals. We store the time-varying road network dataset on the Sparksee [8] graph database . We provide a Java API for easy manipulation of the graph, such that there is no need for a prior knowledge about Sparksee. The API is available at http://elif.eser.bilkent.edu.tr/roadnetwork.

II. Data Generation

In this section, we explain our process to generate the time-varying graph dataset and store it in a graph database. We first build the road network and match the trajectory GPS traces with the underlying network. We estimate the incomplete parts of the dataset by a variety of statistical approaches.

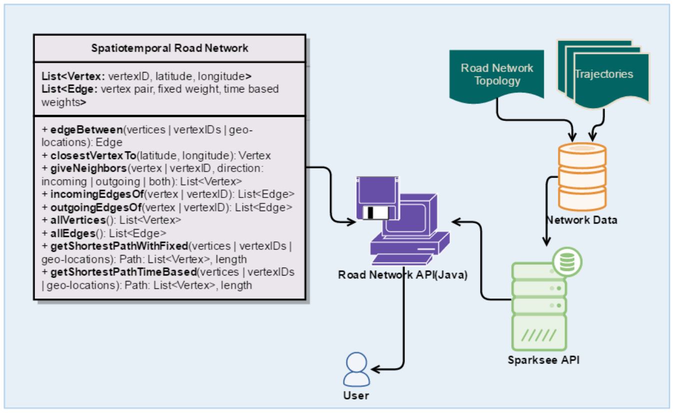

Spatiotemporal Road Network
List<Vertex: vertexID, latitude, longitude**>**
List<Edge: vertex pair, fixed weight, time based weights**>**
+ **edgeBetween**(vertices | vertexIDs | geo-locations): Edge
+ **closestVertexTo**(latitude, longitude): Vertex
+ **giveNeighbors**(vertex | vertexID, direction: incoming | outgoing | both): List<Vertex>
+ **incomingEdgesOf**(vertex | vertexID): List<Edge>
+ **outgoingEdgesOf**(vertex | vertexID): List<Edge>
+ **allVertices**(): List<Vertex>
+ **allEdges**(): List<Edge>
+ **getShortestPathWithFixed**(vertices | vertexIDs | geo-locations): Path: List<Vertex>, length
+ **getShortestPathTimeBased**(vertices | vertexIDs | geo-locations): Path: List<Vertex>, length
Road Network Topology
Trajectories
Network Data
Road Network API(Java)
Sparksee API
User

Fig. 1: System Structure

A. Building and Managing the Road Network Topology

We generate the road network as a directed graph with vertices as latitude-longitude pairs and edges with both fixed and time-dependent weights. For time-dependent weights, we seek to identify accurate travel time values for 288 time slots in a day starting from 00:00 to 23:55. The fixed weight represents the average travel time of the edge.

We use OpenStreetMap (OSM) [9] to gather the topology of the underlying road network. OSM gives an XML file based on given geo-location boundaries with latitude and longitude values along with sequences of roads. We build the network by representing the road/street intersections and end points as vertices and the road/street segments as edges. We treat each direction of double roads as different edges.

For storage and manipulation of the data, we employ Sparksee, formerly known as DEX, which is a scalable graph database [8]. Figure 1 shows the layers of the system structure. We provide a new Java API with easy to use features and functionalities. Apart from the access to the network, e.g. retrieving edges and vertices or finding the closest vertex to a given geo-location inside the boundary, we provide methods for computing the shortest paths both for time-varying and fixed travel times of the edges. For details, please refer to web-based documentation on http://elif.eser.bilkent.edu.tr/roadnetwork/documentation.

B. Sample Dataset: Milan City Traffic

To generate the time-varying graph dataset, we apply our process over Telecom Italia Data Challenge Floating Car data for Milan which contains 65,956,914 different trajectories for 61 days between March and April, 2015 [7]. The trajectory traces include latitude-longitude pairs, time and speed information covering the rectangular area between the minimum and maximum latitude-longitude pairs within 45.3335945°N, 8.9415892°E and 45.5725183°N, 9.376938°E (Figure 2).

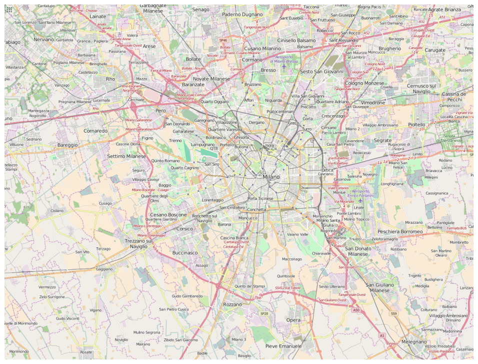


Fig. 2: Road Map of Milan

Figure 3 shows the distribution of the collected Telecom Italia trajectories over time. It depicts the percentage of all the trajectories as y-axis and the time slots in a day as x-axis. The figure shows that the density of the trajectories increases during the rush hours, i.e., between 07:30-10:00 and 17:00-19:30, and reaches to the maximum level at 18:15. The trajectories captured during rush hours cover approximately 38% of the total. The time slots having the least number of trajectories belong to the night hours, from 00:00 to around 05:00, and reaches the minimum level at 03:55.

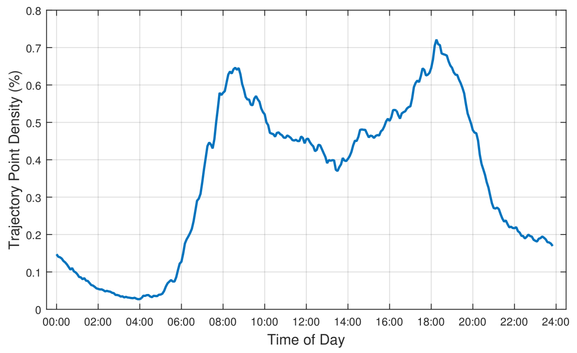


Fig. 3: Trajectory Density over Time

C. Data Preparation and Cleaning

We seek to assign accurate time-varying travel times to the edges of the underlying graph. The GPS trajectories are typically noisy and sparse, thus, careful examination and cleansing are needed. For example, we observe that 33% of the trajectories (21,706,508) have a speed of less than 10 km/h in our dataset. Around 27% of this subset is indeed noise: it corresponds to the starting of the engine and ending of the trip; and they do not represent the actual traffic. The remaining data are captured during an active driving session and represent real traffic conditions such as waiting at the traffic lights or being stoped due to an obstruction on the road. We distinguish these two cases via interpolation based on the location and speed information of two consecutive traces of the same trajectory, x' and x. We omit the data for the starting and ending traces of the trip. As a final step, we apply the following modifications in order to detect unusual decelerations or stops, which also do not represent real traffic conditions, and might be caused by instantaneous circumstances:

$$\begin{aligned} &\forall\ trajectory\ traces\ x\ with\ v < 10\ km/h:\\ &\quad v' = \Delta distance(x', x)/\Delta time(x', x)\\ &\quad v = \begin{cases} v' & v' > v \\ v & otherwise \end{cases} \end{aligned} \tag{1}$$

where $\Delta distance(x', x)$ is the physical distance between two traces, x', and x. $\Delta time(x', x)$ is the time spent to arrive from the location of trace x' to that of trace x. v is the recorded speed provided by the trace, and v' is the speed computed according to the movement and time difference between x' and x. If the calculated speed value indicates that the vehicle is faster, we replace the recorded speed v with movement-based speed v'.

D. Matching Trajectories with the Road Network

To efficiently match the trajectory points with the edges, we partition the graph into spatial subgraphs by exploiting the planarity of the road network. We compute the geographically closest edge for each trajectory point in the corresponding subgraph. The double-ways are represented by two edges with almost the same distance to the trajectory points. The direction is disambiguated using the previous road matches for that trajectory, if available. Else, we make the direction assignment based on the geographic position of the point.

E. Dividing Data into Time Slots

After we match all trajectory points with the corresponding network, we sort the trajectory points of each edge by date and time. For each edge, we divide each day into 288 time slots, i.e., with 5-minute periods in a day from 00:00 to 23:55 and distribute each trajectory point of the corresponding edge to the corresponding time slot. The different dates show similar patterns for all the days of the week. Figure 4 depicts the autocorrelation function for a sample edge. The figure clearly shows an autocorrelation of the signal within a day lag with peaks at one day and at multiples of a day. This strengthens our intuition that the data has a daily periodicity with a relatively high level of confidence. However, the autocorrelation function does not exhibit a clear weekly period since we do not observe a significantly higher peak at lag seven (day). We observe that the patterns for weekdays and weekends is similar, which apparently seems counter-intuitive. Thus, in practice, we do not observe any significant difference in Milan. Hence, in order to avoid space inefficiency and lessen sparsity, we merge the same time slots of all days' (including weekdays and weekends) trajectory data together to form the aggregated data for each edge.

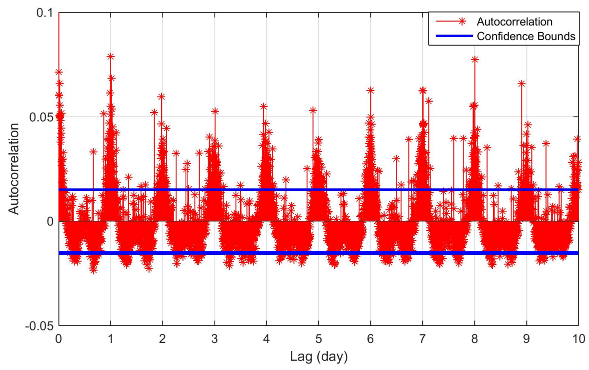


Fig. 4: Autocorrelation of An Edge Data

Note that we do not have any value for some time slots of some edges due to the sparsity of the traces. Figure 5 represents the load ratio of edges with respect to the time slots. The x-axis represents the time slots while y-axis (load ratio) shows the percentage of edges that have travel time values based on existing real data for the corresponding time slot. The time slots having the maximum, 37%, and the minimum, 3.5%, load factors correspond to 18:20 and 03:55, respectively. This figure clearly shows the level of sparsity of the data. Using commercial providers' data with larger amounts of data per time slot, like TomTom [10], would lead us to larger load ratios. In our case, the obtained ratios expose the need for further estimations as we explain in the following sections.

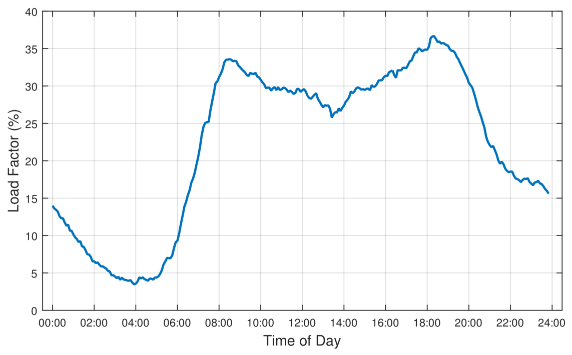


Fig. 5: Load Ratio of Edges over Time

F. *Outlier Detection*

We need to identify outlier speed values, i.e., that were recorded not due to traffic but due to various other reasons, such as cars run over the speed limits, etc. To detect and exclude those outliers we apply Generalised Extreme Studentised Deviate (ESD) Test [11], which is a generalisation of Grubb's Test for more than one outlier. Once the outliers are removed, we expect that the remaining speed values enable us to have a better understanding of the traffic condition at each time slot. Because we employ travel time as weights in the time-varying graph, we aggregate the length of the edges from OSM, then divide it by the speed values.

G. *Interpolation and Filtering*

We need to address two major issues to finalize the dataset with the resultant vector with 288 weights spanning a day with 5-minute intervals:

- **Missing Data:** After forming the daily vectors, i.e., that contain 288 slots per day per edge, we observed that 77.68% of the vector slots do not contain any data.
- **Noise:** The data after the aggregation stage has noise, i.e., high frequency components apparent as jumps in the data, associated with it.

To address these issues we compute the frequency spectrum of the most populous edges to give an idea about the nature of the time series involved. The average power of the respective frequencies is plotted in Figure 6.

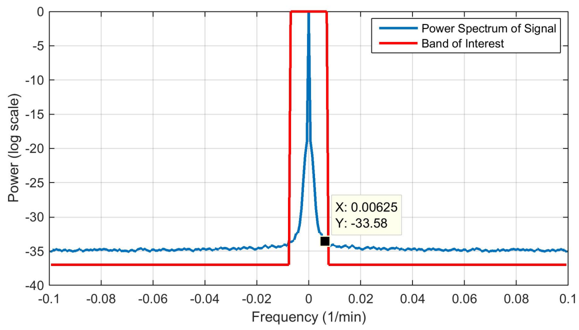


Fig. 6: Average Frequency Spectrum of The Most Populous 4000 Edges

The spectrum shows a band-limited signal with the majority of the content in the low-pass region. The other components out of the region are the noise that we observe in the aggregated signal. The signal of interest is depicted with a red band in Figure 6. This shows us that the signal is band-limited. The cutoff frequency for the signal is $0.0069/min$ which we select according to the noise level.

According to the signals and systems theory, this signal has to be sampled with at least Nyquist rate which is the twice the highest frequency (F_{max}) in the signal [12]. The minimum sampling interval for the signals of interest is given in Equation 2.

$$\begin{aligned} &F_{Nyquist} \geq 2.F_{max} \geq 0.0139 \\ &T_{sampling} \leq \frac{1}{F_{Nyquist}} \leq 72 \text{ minutes} \leq 14 \text{ samples} \end{aligned} \tag{2}$$

Equation 2 illustrates that any signal which has data with 14 consecutive missing data is undersampled and will cause aliasing. We observe from the data that this requirement does not hold for most of the edges of the data. To overcome these problems we develop two solutions for the respective cases. If the length of consecutive missing data is larger than 14 samples we use the minimum duration for that edge. This minimum duration is calculated using the length of the edge and the speed limit where available, or the value for urban areas (50 km/h). Else, we draw a sample data from a normal distribution $\mathcal{N}(\mu, \sigma^2)$ where the parameters μ and σ are calculated from the existent data for the edge. With these corrections in the time series, we use a low pass filter with cutoff frequency as depicted in Figure 6 (this filter is applied in the spectral domain) to compute the final time series. This interpolation technique is also called sinc interpolation or Whittaker-Shannon interpolation for band-limited signals. The flow of the process is outlined in Algorithm 1.

Input:
1) *S: Time series set for the edges of the given network*
2) *δ: Minimum travel time for edge n*
Output: *μ and σ: variance and standard deviation from the complete data*

```
 1: procedure DataInfo($S, \delta$)
 2: Define $I$ ▷ a set for missing data intervals
 3: for $n = 1 : size(S)$ do
 4:     $I[n] \leftarrow MissingDataIntervals(S[n])$ ▷ the time slots of $S[n]$ having no data
 5: end for
 6: $S'[n] = S[n]$
 7: for all $I[n] \in I$ do
 8:     if $Len(I[n]) \geq 14$ then
 9:         Fill $I[n]$ with $\delta$
10:     else
11:         Fill $I[n]$ with samples drawn from $\mathcal{N}(\mu, \sigma^2)$
12:     end if
13:     Update $S'[n]$ missing intervals with $I[n]$
14: end for
15: $S_{LPF}[n] = Low\ pass\ filter\ S'[n]$
16: Fill missing data of $S[n]$ with $S_{LPF}[n]$
```

Algorithm 1: Algorithm for interpolation and filtering

III. Time-Dependent Shortest Paths

In this section, we compare traditional vs time-dependent versions of shortest paths solutions over the dataset and system we built. We aim to quantify the difference in using a fixed weight graph vs. a time-varying graph in finding the shortest paths. Since the edges of the graph have different travel time values(edge weights) for different time slots within a day, in time-dependent shortest paths (TDSP) the employed travel time (edge weight) of an edge changes depending when the path employs the edge. This information has a cascading dependence on the travel times of the previous edges. The steps of the TDSP we employ is given in Algorithm 2. The algorithm spreads from the start vertex s to the destination vertex d, by iterating until there is no new vertex to be processed, i.e., the shortest paths of all vertices that can be reached by s are discovered, or d is already found (Line 8). For the edge weight between two vertices, the algorithm uses the weight belonging to arrival time of the preceding vertex as in Line 14.

For the standard shortest path problem, we utilize the same algorithm by modifying the edge weight related parts (Line 12, 14). We discard the usage of t_{start} and get the fixed edge weights, i.e., based on the average travel time on the corresponding edges.

A. Experimental Setup

We sample the dataset by using the start and destination locations of randomly selected 4,620 real trajectories. The distribution of the selected trajectories (Figure 7) is similar to the trajectory density over the time slots (Figure 3).

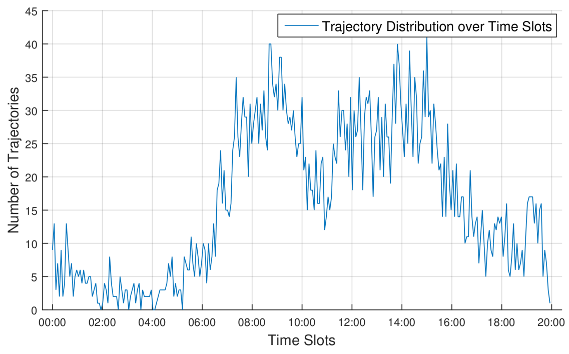


Fig. 7: Sample Trajectory Distribution over All Time Slots

We compare the time-varying vs. fixed weights shortest paths using two measures: similarity of resulting paths and gain of time-dependent shortest path regarding the travel time. The similarity of paths is based on the common edges of two given paths. Here we calculate the similarity between TDSP and traditional shortest path. We use Jaccard distance (Eq. 3). The range of similarity index is [0,1].

$$sim(p_1, p_2) = \frac{edgeSet(p_1) \cap edgeSet(p_2)}{edgeSet(p_1) \cup edgeSet(p_2)} \quad (3)$$

where p_1 and p_2 represent paths, i.e., an ordered vertex set.

The gain on the path length, or more accurately path duration is as follows:

$$\frac{\Theta(SP_f(s,d),t) - \Theta(SP_{tv}(s,d),t)}{\Theta(SP_f(s,d),t)} \quad (4)$$

where the Θ function computes the duration of the path starting at t. SP_{tv} and SP_f are the shortest paths obtained

Input:
1) *G(V,E): the spatiotemporal network whose each edge e has travel times for different time slots*
2) *s: source vertex over the network*
3) *d: destination vertex over the network*
4) *t_{start}: start time of desired path for s and t.*
Output: *p: a path including vertex list*

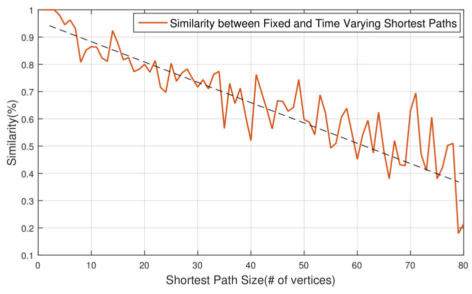
```
 1: procedure TDSP($G, s, d, t_{start}$)
 2: $p \leftarrow \emptyset$ ▷ result path
 3: Define $D$ ▷ a priority queue of vertices with travel time
    as priority index
 4: $D[s] \leftarrow 0$
 5: Define $P$ ▷ a list of preceding vertices
 6: $P[s] \leftarrow NIL$
 7: Define $S$ ▷ vertices having the shortest paths
 8: while $D.size \neq S.size \vee d \in S$ do
 9:     $v \leftarrow ExtractMin(D)$
10:     $S \leftarrow S \cup v$ ▷ add v to discovered list
11:     for all $u \in outNeighbors(v)$ do
12:         if $u \notin S \wedge$
            $D[u] > D[v] + edge(v,u).weight[t_{start} + D[v]]$
            then
13:             ▷ the index of a vertex not included in D is $+\infty$
14:             $D[u] \leftarrow D[v] + edge(v,u).weight[t_{start} + D[v]]$
                ▷ update travel time to u
15:             $P[u] \leftarrow v$ ▷ update preceding of u
16:         end if
17:     end for
18: end while
19: if $d \in D$ then
20:     $cur \leftarrow d$
21:     while $cur \neq NIL$ do
22:         $p.addHead(cur)$
23:         $cur \leftarrow P[cur]$
24:     end while
25: end if
26: return $p$
```

Algorithm 2: Time Dependent Shortest Path (TDSP)

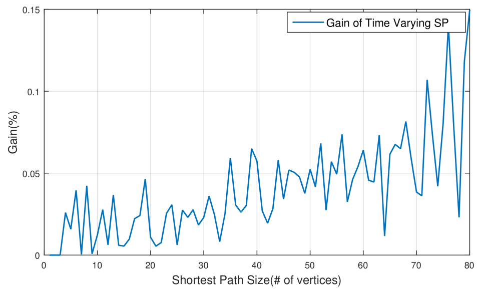
with the time-varying and fixed weights, respectively. For each query,i.e., source-destination pair (s-d), we measure how much gain one would get using a time-varying network instead of employing the traditional network. For simplicity, we refer the path retrieved with time-varying weights as SP_{tv} and the one with fixed weights as SP_f.

B. Experimental Results

Figure 8 presents the result of each measure with respect to path sizes, i.e., number of vertices of the paths, ranging from 1 to 80 for our dataset. The average similarity ranges from 0.18 to 1, for path sizes of 79 and 5, respectively. Similarity between SP_{tv}s and SP_fs decreases as the path size increases (Figure 8a). Note that the fluctuations on the figures stem from varying the number of samples we have for each path size. For example, there are 161 paths of length 23 vertices, and there are only 2 of length 76. Figure 8b illustrates that the shortest path queries over time-varying network get higher gains, i.e., SP_{tv} has much lower travel time than SP_f, as the number of vertices increases. Note that the gain cannot be lower than 0, because if SP_f had lower travel time at t, SP_{tv} would be the same as SP_f according to TDSP algorithm (Algorithm 2). In other words, if a path for a query provides more benefit, i.e., less travel time, than any other paths starting at the same time, the path would be the resultant path of TDSP. For the dataset the gain goes up to 0.14, for the path with 80 vertices.



(a) Similarity of Time-varying Shortest Paths vs Static Paths



(b) Gain of Time-varying Shortest Paths over Static Paths w.r.t. Path Length

Fig. 8: Path Size based Analysis

Figure 9 shows the results related to one query started at different times to show the changes in the result paths. We choose a representative query with an average of 37 vertices in all its resulting paths having the start times with half-hour intervals. It means there are 48 different queries with the same source-destination pair yet different start times. The x-axis of the figure represents the start times of the query. The red line illustrates the similarity index between SP_{tv} and SP_f with the corresponding start times. Similarity between SP_{tv}s and SP_f varies in time reaching exact paths in some cases, i.e., similarity index 1, and not exactly equal paths in other cases, with a worse case similarity index 0.5. Additionally, we present the similarity between the consecutive shortest paths with the green dashed line. According to the results, all result paths are at least 45% similar. In total, 15 different paths are retrieved and SP_f is observed in the time-varying shortest paths twice. On average, the consecutive paths are similar in an hour period, probably because of the similar traffic patterns, in the period. After the period, the selected path at the next start time change drastically, i.e., not smoothly any more. It may indicate the changes in the traffic conditions may not evolve gradually. The paths belonging to night hours, i.e., 22:00-6:00, do not change as much as the ones between 7:00 and 18:00.

The results confirm that time-varying shortest paths can reveal alternative paths with shorter time routes and the paths are not necessarily similar to the paths computed in static networks.

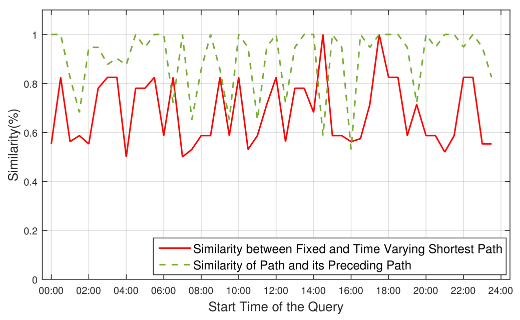


Fig. 9: Comparisons on Time-Varying Paths for the Same Query with Different Start Times

IV. Conclusion and Future Work

We presented our methodology to generate a time-varying road network dataset along with a graph manipulation API that we share to help research on time-varying graphs.

We match the traces of a sparse GPS data with a road network, aggregate the data to a single day version with 5 minutes sampling interval, and develop a time-series model that infers and estimates the missing data. We employ a scalable database, Sparksee, for the storage and manipulation of our dataset. We provide a Java API for easy utilization. One can download the database file and can use the *.jar* to manipulate the network. Besides the basic operations on graphs such as getting edges and vertices, we include shortest path methods for time-varying and fixed weights of the edges.

We employ the system in a use case evaluation of time-varying shortest path solutions. The difference between the shortest paths using time-varying vs. fixed weights increases as the number of vertices increases. The travel times of these two types of shortest paths give different results in favor of using the time-varying road network.

Our work is an early step towards generating benchmarks and systems for analysis and management of time-varying graphs. The dataset we employ is not a large-scale one. One of our future directions is building larger time-varying road networks with larger amount of GPS traces. Additionally, we plan to feed our database with more real based trajectory data and increase the ratio of the real based data against the synthetic (estimated) data; redistribute trajectories considering each day of the week independently, instead of merging all trajectories into one sample day. An accurate and scalable dataset and graph manipulation tool can be used in smart city applications, route recommendation systems and location based services.

References

[1] B. George and S. Shekhar, "Time-aggregated graphs for modeling spatio-temporal networks," in *Advances in conceptual modeling-theory and practice*. Springer, 2006, pp. 85–99.

[2] S. E. Dreyfus, "An appraisal of some shortest-path algorithms," *Operations research*, vol. 17, no. 3, pp. 395–412, 1969.

[3] H. Wang, G. Li, H. Hu, S. Chen, B. Shen, H. Wu, W.-S. Li, and K.-L. Tan, "R3: a real-time route recommendation system," *Proceedings of the VLDB Endowment*, vol. 7, no. 13, pp. 1549–1552, 2014.

[4] J. Letchner, J. Krumm, and E. Horvitz, "Trip router with individualized preferences (trip): Incorporating personalization into route planning," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 21, no. 2. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006, p. 1795.

[5] L. Cao and J. Krumm, "From gps traces to a routable road map," in *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*. ACM, 2009, pp. 3–12.

[6] T. Hunter, R. Herring, P. Abbeel, and A. Bayen, "Path and travel time inference from gps probe vehicle data," *NIPS Analyzing Networks and Learning with Graphs*, vol. 12, no. 1, 2009.

[7] "Source of the dataset: Tim big data challenge 2015," www.telecomitalia.com/bigdatachallenge, accessed: 2015-06-1.

[8] N. Martínez-Bazan, V. Muntés-Mulero, S. Gómez-Villamor, J. Nin, M.-A. Sánchez-Martínez, and J.-L. Larriba-Pey, "Dex: high-performance exploration on large graphs for information retrieval," in *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*. ACM, 2007, pp. 573–582.

[9] M. Haklay and P. Weber, "Openstreetmap: User-generated street maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, 2008.

[10] "Tomtom city," http://www.tomtom.com/en_gb/traffic-news/, accessed: 2016-07-1.

[11] "Michael thomas flanagan's java scientific library," http://www.ee.ucl.ac.uk/~mflanaga/java/.

[12] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab, *Signals & Systems*, ser. Prentice-Hall signal processing series. Upper Saddle River, N.J. Prentice Hall London: Prentice Hall international, 1997.