

On computing temporal functions for a time-dependent networks using trajectory data

Samara Martins
Nascimento
Federal University of Ceara
Ceara, Brazil
samara.nascimento@dc.ufc.br

Mirla R. R. Braga Chucre
Federal University of Ceara
Ceara, Brazil
mirla.chucre@dc.ufc.br

Jose Antônio Fernandes
de Macedo
Federal University of Ceara
Ceara, Brazil
jose.macedo@lia.ufc.br

Jose Monteiro
Federal University of Ceara
Ceara, Brazil
monteiro@dc.ufc.br

Marco Antonio Casanova
Pontifical Catholic University
of Rio de Janeiro
Rio de Janeiro, Brazil
casanova@inf.puc-rio.br

ABSTRACT

Time dependent networks are of key importance to allow computing precise travel times taking into consideration moving object's departure time. However the computation of time functions that are used to annotate time dependent networks are challenging since we must cope with noisy and incomplete traffic data. Recent related works adopt approaches that build *Piecewise* linear functions, which do not cope with aforementioned problems. In this work, we propose a new method for generating *Piecewise* linear functions by applying a map-matching technique allied to a curve smoothing approach in order to treat outliers and complete data. We performed experiments using real trajectory data and compared our results with a baseline. Preliminary results show that our approach generates time functions with better approximation than the baseline competitor.

Categories and Subject Descriptors

H.2 [Database Management]: Miscellaneous; H.3 [Information Storage and Retrieval]: Miscellaneous

Keywords

Time Dependent Network; Functions & Piecewise

1. INTRODUCTION

The increasingly growth of trajectory data allows creating time-dependent services over road networks that resort on spatial queries, such as, shortest path in time, route planning taking into consideration varying travel times, optimal sequence routing that uses time-to-service of point of interests, KNN queries, to name a few. Clearly, all these queries need

an underlying network model equipped with travel time cost functions, which allows answering such queries according to departure time.

In the literature, several works propose answering spatial queries on time-dependent network by using shortest path algorithms (KNN) as in [1, 2, 3]. However, few works describe how to create a time-dependent network using trajectory data, which is a challenging task since trajectory datasets usually contains incomplete information and outliers. Incomplete data is due to the inherited traffic data distribution, where few road segments have a lot of traffic/trajectory data and many road segments have few traffic/trajectory data. Outliers are mainly generated by GPS device errors.

In order to create a time-dependent road network, it can be done by annotating edges with temporal cost functions. The advantages to use this model is to build up a structure that frees up disk space and minimizes the size of the time dependent network. In this way, if n edges were created between two consecutive vertices $v_i \in V$ and $v_{i+1} \in V$, we can use a temporal function, in the time dependent network, and obtain just one edge between v_i and v_{i+1} . Therefore, the number of edges is minimized as well as the running time of shortest path algorithms.

The main objective of this paper is to compute temporal cost functions to time-dependent networks. We state our problem as *"Considering a street segment, with two intersections, we would like to find one temporal function, whose can represent the duration of all objects that travels in this road, at any time of day"*. Our approach determines how the temporal function would behave for each part of network, observing the moving objects that crossing in each road.

The first contribution of this paper is a stochastic method by combining two predict functions: (i) *Piecewise*; and (ii) *LOESS*, which is a method for smooth curve [4], that allows observing the different breakpoints existing in a trajectory data distribution. Another contribution of this work is in the *Map Matching* processing, which associates each trajectory position to a correct road segment. Moreover, our *Map Matching* processing returns a matrix structure composed by the road segment, the timestamp and the total travel time for this road segment. This matrix is given as input to

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

IDEAS '16, July 11-13, 2016, Montreal, QC, Canada

© 2016 ACM. ISBN 978-1-4503-4118-9/16/07... \$15.00

DOI: <http://dx.doi.org/10.1145/2938503.2938542>

create our regression method.

This paper is structured as follows. In Section 2, we introduce some important definitions for understanding our proposed solutions and explain the road network model used. In Section 3, we explain our approach and show the correctness of our solution. We show an experimental evaluation and results in Section 4 and in Section 5, we present a brief discussion of related works. Finally, we present the conclusions of the paper in Section 6.

2. PRELIMINARIES

We assume that the structure of a time-dependent road network is modeled by a graph where the vertices represent starting and ending points of road segments or intersections. Those are connected by edges and the cost to traverse these edges vary with time. More formally, we formalize TDG as follows:

Definition 1. (Time-Dependent Graph (TDG)) A **time dependent graph** (TDG) $G = (V, E, C)$ is a graph where: (i) $V = \{v_1, \dots, v_n\}$ is a set of vertices; (ii) $E = \{(v_i, v_j) \mid v_i, v_j \in V, i \neq j\}$ is a set of edges; (iii) $C = \{c_{(v_i, v_j)}(\cdot) \mid (v_i, v_j) \in E\}$, where $c_{(v_i, v_j)} : [0, T] \rightarrow R^+$ is a function which attributes a positive weight for (v_i, v_j) depending on a time instant $t \in [0, T]$ and where T is a domain-dependent time length.

For each edge (u, v) , a function $c_{(u, v)}(t)$ gives the cost of traversing (u, v) at the departure time $t \in [0, T]$. The formalization about the function cost is defined follows:

Definition 2. The function $c_{(u, v)}(t)$ is an *Piecewise* linear function where: each edge (u, v) is associated with a street segment s_i , observing the timestamp TS_i , the total time TT_i . The *Piecewise* is the $\sum s_{ce_i}(TT_i, TS_i)$, where TT_i is the travel duration and TS_i is the time of beginning of service on edge (u, v) , for all objects.

We assume that the travel times of the edges in the network follow the FIFO property, i.e., an object that starts traversing an edge first has to finish traversing this edge first as well. The general time-dependent shortest path problem in which the departure is immediate, i.e. the user departs exactly at the time t , and in which waiting is disallowed everywhere along the path through the network is NP-hard [5], but it has a polynomial time solution in FIFO networks. Since the travel times satisfy the FIFO property, waiting in a intermediary vertex in a path is not beneficial.

Note that the definition given above does not require the graph to be bi-directed. More specifically, the existence of an edge (u, v) does not imply in the existence of the edge (v, u) . Furthermore, there may be opposing edges (u, v) and (v, u) such that $c_{(u, v)}(t) \neq c_{(v, u)}(t)$.

The time cost to traverse a path from a specific starting time, or departure time, is called *travel-time*. The *travel-time* is calculated assuming that stops are not allowed because, as discussed before, we consider that the network is FIFO waiting in a vertex do not anticipate the arrival time of a vehicle. The *travel-time* of a path is calculated considering the *arrival time* at each vertex belonging to it. These concepts are formally defined below.

Definition 3. Given a TDG $G = (V, E, C)$, the *arrival time* at the vertex v_j of an edge $(v_i, v_j) \in E$ at departure $t \in [0, T]$ is given by $AT(v_i, v_j, t) = t + c_{(v_i, v_j)}(t) \bmod T$.

Given that a vehicle starts a path at a vertex of the graph and this path starts at a determined departure time, the *arrival-time* calculates the time instant when the vehicle arrives at the other end of the edge.

Definition 4. Given a TDG $G = (V, E, C)$, a path $p = \langle v_{p_1}, \dots, v_{p_k} \rangle$ in G and a departure time $t \in [0, T]$, the *travel-time* of p is the time-dependent cost to traverse this path, given by $TT(p, t) = \sum_{i=1}^{k-1} c_{(v_{p_i}, v_{p_{i+1}})}(t_i)$ where $t_1 = t$ and $t_{i+1} = AT(v_{p_i}, v_{p_{i+1}}, t_i)$.

The above definition shows how the cost of a path, called *travel-time*, is calculated. Given the sequence of vertices that compose a path and the time instant when one starts to traverse this path, the *travel-time* is the sum of the costs to go from one vertex to the next one in the sequence. The cost to go from the first to the second vertex is calculated considering the departure time t . The cost to reach the next vertices depends on the *arrival-time* at the previous vertex. It is important to notice that this definition does not take into consideration stops at the nodes of the graph, that is, the way to the next vertex in the sequence begins at the same moment when the previous vertex was reached.

3. A METHOD FOR COMPUTING TEMPORAL FUNCTION

In this section, we describe a method for computing temporal functions used to build time-dependent networks. The objective of this method is to compute temporal functions that captures correct behavior of traffic/trajectory data distribution, considering the changes of day time and treating outliers.

The TDG is built from a road network model where each edge contains a non-negative weights, and each edge connects two different nodes of the graph. Likewise, the main goal to use functions between two nodes is minimizing the number of edges. We present two solutions to process the cost of the edge: the *Piecewise* function, namely Naive and our proposal. They are based on *Piecewise function*, which is a defined on a sequence of intervals expansion [6]. The first solution uses *Piecewise* predictor, and the second solution use two predict functions, the smooth curve, which is the *LOESS* function, to discover the breakpoints values and after this, we use the *Piecewise* function.

The *Map Matching* is the first step to build the temporal function. It shall ensure the information wiping and the completeness of these information, obtained from GPS. After the cleaning and completeness process, the own *Map Matching* organizes the information structure provided on crossing time. That is to say, for each road segment, the *Map Matching* service returns a matrix structure with information about the timestamp (i.e. TS) and total travel (i.e. TT). The structure obtained is similar to that shown in Figure 1.

The *Piecewise* function concerns the second step and corresponds to a model widely used in temporal series. This function is an general example of *Spline* interpolation functions, whose divides the interval of interest in sub-intervals, that are the breakpoints values. Each segment obtained from sub-intervals corresponds to one linear segment, that can differ according with changes of the slope curve. The breakpoints value may be estimated before the analysis beginning, which does not make this process trivial.

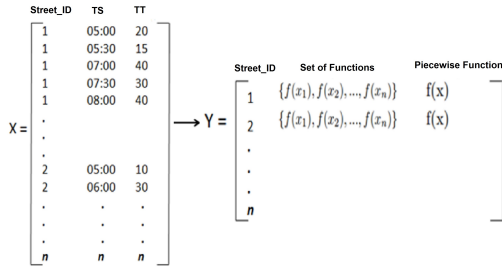


Figure 1: Structure of receipt and return the information.

3.1 Naive Solution (Baseline)

The naive approach is basically an extension of segmented package [7], which the relationships between the response and one or more explanatory variables are *Piecewise* linear. The breakpoints are important values, although may be of some concern, due to difficulties in estimating and testing. This naive approach adopts the following criteria: (i) The estimation might have more than one breakpoint and/or with large datasets; (ii) The uncertainty of breakpoint value is not taken into account.

3.2 Proposed Solution

The implementation of our strategy needs an estimate of breakpoints and needs to know how to represent breakpoints according to data distribution. To satisfy these requirements we use the smoother *LOESS* [4] to make adjustments and treat the outliers. Additionally, using the *LOEES* function, we could determine where the breakpoints will manifest, from the observations made in the changes of smooth curve, whose is possible to obtain the peaks from changes of smooth, which will define the breakpoints.

After computing *LOESS* function, we obtain, both the data set with the correct adjust of outliers and the breakpoints. These values are used in the second step, to build the temporal function. These functions try to estimate the conditional value (i.e. expected value), like the time of day, from other value, which is independent, such as the total travel.

The temporal function built up from each segment is defined between two consecutive breakpoints, which correspond to the values of x . Analyzing, individually, each breakpoint we should have the *Piecewise* function using the model below. To the first point breakpoint (i.e. c_1), we are able to write: $y = a_1 + b_1 * x$; to $x \leq c_1$ and $y = a_2 + b_2 * x$; to $x > c_1$. When $x = c_1$, both equations of y also must be equal. So: $a_1 + b_1 * c_1 = a_2 + b_2 * c_1$. So $a_2 = a_1 + c_1 (b_1 - b_2)$. Replacing a_2 in $y = a_2 + b_2 * x$, we can find the first part of the segmented regression: $y = \{a_1 + c_1 (b_1 - b_2)\} + b_2 * x$.

To compute the second part of the regression, we will use the next breakpoint (i.e. c_2), and we will use the concepts defined previously. Furthermore, for this case, we will use the results of first regression (i.e. $y = a_3 + b_3 * x$). Thus, using $x = c_2$, we have: $y = \{a_1 + c_1 * (b_1 - b_2)\} + b_2 * x$; to $x \leq c_2$. So $y = a_3 + b_3 * x$; to $x > c_2$.

When $x = c_2$, the both equations of y also need to be

equals. Therefore, the calculus previously made shall be remade, with different functions. So, we have: $\{a_1 + c_1 * (b_1 - b_2)\} + b_2 * c_2 = a_3 + b_3 * c_2$. So $a_3 = \{a_1 + c_1 * (b_1 - b_2)\} + c_2 * (b_2 - b_3)$. Replacing a_3 in $y = a_3 + b_3 * x$, we find the second part of the segmented regression: $y = \{a_1 + c_1 * (b_1 - b_2) + c_2 * (b_2 - b_3)\} + b_3 * x$.

The computation must be repeated for all the n breakpoints and $n+1$ linear regressions. In our example, we have three breakpoints and four linear regressions. When $x = c_3$, for the last breakpoint, we should be write the last part of segmented regression based on the last calculus realized and the linear regression $y = a_4 + b_4 * x$. So, we have: $y = \{a_1 + c_1 * (b_1 - b_2) + c_2 * (b_2 - b_3)\} + b_3 * x$; to $x \leq c_3$ and $y = a_4 + b_4 * x$; to $x > c_3$.

When $x = c_3$, the y equations also same, considering the breakpoint value. So, we can calculate these equalities as follows: $\{a_1 + c_1 * (b_1 - b_2) + c_2 * (b_2 - b_3)\} + b_3 * c_3 = a_4 + b_4 * c_3$. So $a_4 = a_1 + c_1 * (b_1 - b_2) + c_2 * (b_2 - b_3) + c_3 * (b_3 - b_4)$. Replacing a_4 in $y = a_4 + b_4 * x$, we obtain the segmented regression for our representation: $y = \{a_1 + c_1 * (b_1 - b_2) + c_2 * (b_2 - b_3) + c_3 * (b_3 - b_4)\} + b_4 * x$

4. EXPERIMENTS

This section describes performed experiments conducted in order to evaluate our proposed method and compare it to the described Naive solution. Furthermore, we show the new *Map Matching* algorithm in order to cope with GPS data errors. Our *Map Matching* extends the *Graphhopper* [8] algorithm, which have two objectives: complete data and computes a matrix structure to build the *Piecewise* function. The process to complete the data aims at investigating if there exists some missing point within trajectory data.

In order to generate time-dependent road networks, we have to use two different real datasets. The first data set, available from *Open Street Map* (OSM) [9]. The second data set, available from a Brazilian cab fleet monitoring company called *Taxi Simples* [10], provides data about the distribution and moving of cabs in Fortaleza city. Since, each taxi driver at *Taxi Simples* company has a mobile phone equipped with a GPS device, this data set stores information about the location of taxi drivers. We use the data collected since November 16, 2015 until 31th of December 2015, summing up 99,562 trajectories referring to travels cab around the city. The Figure 2 shows the average amount of data collected by day, for each network edge.

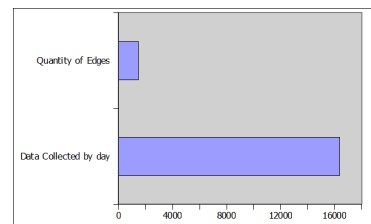


Figure 2: Average amount of data collected.

To have an idea of how effective our temporal function is, we compare it to a Naive solution used as a baseline. Besides, both the proposed approach and the baseline were implemented in Java programming language and R programming language using the *Graphast* framework [11]. All

experiments were conducted on a Intel Core 2 Quad CPU Q6600 server, with 8GB RAM and 2.40GHz, using Ubuntu 11.10 of 64 bits as operating system.

4.1 The Map Matching Process

The *Map Matching* proposed in this work share many characteristics with the algorithm *Graphhopper*, whose contains services for routing optimization with various constraints like time window and capacity restrictions [8]. We performed the map matching between the taxi positions and the road network, which is necessary for mapping trajectory data points to a correct street segment. This data set contains a real world GPS recorded data from taxis of Fortaleza/Brazil [10].

The *Map Matching* algorithm is performed in two different steps. The first step is responsible to obtain the cab trajectories data [12], where each trajectory point is represented by latitude, longitude and timestamp. The information about the taxi positions follows the *GEOLLOCATION API* specification [13] and have an average error of 13.60 meters. The second step associates each trajectory position to a correct road segment.

Algorithm 1 below describes our proposed map-matching algorithm [14]. This algorithm performs interpolation of the taxi position, in the MultiLineString segment. Furthermore, when some information about taxi trajectories are missing, we need to interpolate missing positions. The *Map Matching* also returns a matrix structure, which contains the street segment, initial travel time and the total travel time.

Algorithm 1: Time Total Travel

Input: ML : set of MultiLineString obtained from Fortaleza OSM; CT : set of Cab Trajectory

Output: Map Matching Total Travel

```

1 begin
2   Let  $CT_i$ : Initial point of cab trajectory  $\leftarrow$  false;
3   Let  $CT_f$ : Final point of cab trajectory  $\leftarrow$  false;
4    $MM_{Graphhopper} \leftarrow ML + CT$ ;
5    $V_{Line}[1..k] \leftarrow MM_{Graphhopper}$ ;
6    $\Delta I \leftarrow \Delta F \leftarrow 0$ ;
7   foreach  $link \in V_{Line}[1..k]$  do
8     if  $CT_i \neq false$  then
9       if  $CT_i$  lies  $link_j$  then
10         $\Delta I \leftarrow \Delta I + time(distance(ML_j, CT_i))$ ;
11         $TR \leftarrow TR + time(distance(ML_{j+1}, CT_i))$ ;
12         $foundStart \leftarrow true$ ;
13      else
14         $\Delta I \leftarrow \Delta I + time(distance(ML_j, ML_{j+1}))$ ;
15      else if  $CT_f \neq false$  then
16        if  $CT_f$  lies  $link_j$  then
17           $TR \leftarrow TR + time(distance(ML_j, CT_f))$ ;
18           $\Delta F \leftarrow \Delta F + time(distance(ML_{j+1}, CT_f))$ ;
19        else
20           $TR \leftarrow TR + time(distance(ML_j, ML_{j+1}))$ ;
21      else
22         $\Delta F \leftarrow \Delta F + time(distance(ML_j, ML_{j+1}))$ ;
23       $link.time_{entry} \leftarrow point_0.timestamp$ ;
24       $link.time_{total} \leftarrow point_j.timestamp - point_0.timestamp$ ;

```

The input of Algorithm 1 contains two sets: the MultiLineStrings (obtained from OSM) and the cab trajectories, that may be interpolated in the street segment, when it is necessary. In parallel with this interpolation, the strategy of Algorithm is estimate how long is the travel between the parts of segment, until obtain the total time travel.

Lines 2-3 of Algorithm 1 gets the begin and end of cab

trajectory. At Line 4 creates a begin and end delta time, which is assigned to zero value. An incremental form, the delta value stores the total elapsed time in each street segment. Algorithm checks, for each part of street segment, if exists a taxi close to that segment, and if the cab does not touch this segment (Lines 5-7). At this moment, it is examined if exists some point, in part of MultiLineString, which may contain the trajectory point that will be interpolated. With this detection we can determine what would be the reliable interpolation (obtained by the set of data in this analysis). The delta time is the total time traveled, in each street segment. The total time of edge is the sum up of traversed time, which is the proportion being examined on the next collection (Lines 8-10). If in the first segment of MultiLineString is not made an interpolation, the next segment is analyzed (Line 12). At Lines 13-20 a same rationale can be applied in the last point of trajectory, that realizes the interpolation of cab in the correctly street segment and update the delta end time traveled values.

The result, in the end of processing, is the real trajectory for each object, using the proportions obtained between the point of taxi and MultiLineString. The Figure 3 shows the relationship between the cab trajectories and the road network before and after the *Map Matching* process. Furthermore, the Algorithm also returns a matrix structure, whose name is *MAP-TAXI-CAB*, that stores information about the taxi positions (i.e. the edge, the timestamp and the total travel time).

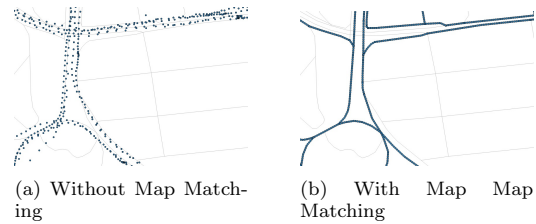


Figure 3: Cab trajectories after and before the *Map Matching* process.

4.2 Predict Temporal Function

The temporal function, proposed in this work, is used to annotate time dependent network edges. As discussed this function is built after applying a *Map Matching* technique allied to a curve smoothing approach in order to treat outliers and incomplete data. Our main goal is to generate time functions with better approximation with the roads traffic, considering the different hours of the day. The new *Piecewise* function needs our *Map Matching* algorithm necessary for building the matrix structure. The second step corresponds to the creation of a *Piecewise* function by using *LOESS* predict function. This curve smoothing, besides treat the outliers, also estimates the changing of travels duration.

The *LOESS* is used in temporal series and corresponds to an iterative process, whose each step applies the previous local regression, calculates the outliers and applies again the local regression. The procedure is repeated until reaching convergence. In this work, the *LOESS* also indicates where we can estimate the breakpoint values, as showed in Algorithm 2. This means that each point inflection of the

curve defines new breakpoint values defined to calculate the function.

Algorithm 2: Inflection Points

```

Input: Matrix structure, composed of timestamp  $TS$  and total
travel  $TT$  of the travel in each street segment.
Output: Inflection Points obtained from the prediction
1 begin
2    $plot \leftarrow M(TS, TT);$ 
3    $data.loess \leftarrow loess(TT TS);$ 
4    $seq \leftarrow \{f_{data.loess}\}_n \in TS;$ 
5    $y.predict \leftarrow predict(data.loess, seq);$ 
6    $derived \leftarrow \{d \div d(y.predict)\}f(y.predict);$ 

```

The Algorithm 2 defines inflection points of the curve, using a prediction strategy of *LOESS*. The line 2 indicates how data is distributed. In line 3 *LOESS* function processes the data received and in line 4, we have a sequence of all timestamps between minimal and maximal data. In line 5 the predict values are computed, for each timestamp value. Finally, in line 6 the algorithm computes the inflection points of the curve, from the calculating of derived, which compute the minimum and maximum values of predict function, observing the changes of curve signal. The result of Algorithm 2 is the breakpoint values, which is used to build the temporal function.

The Figures 4, 5 and 6 show two graphical representations of the behavior of the temporal function and illustrate the distribution of the cost function for one day, on December 01, 2015, in Scenario I; for one week, from December 01, 2015 to December 08, 2015, in Scenario II; and for one month, from November 31, 2015, to December 30, 2015, in Scenario III. To calculate the functions, we need to convert the timestamp and the total travel time in *Milliseconds*.

The experiments illustrates a typical problem that occurs when we need to annotate the time dependent networks with time function. The Naive solution, shown in Figures 4(a), 5(a) and 6(a), adopts the approaches of recent related works, that build *Piecewise* linear functions. The main problem is the data of traffic used, that may be noisy and skewed. Therefore, the computed function in Naive solution does not respect the real data distribution, whose structure is made up of different peaks of travel times throughout the day. The Figures 4(b), 5(b) and 6(b) shows the new method for generating *Piecewise* linear functions by applying a curve smoothing approach in order to treat outliers and represent the real distribution of data. In Table 1, we shown the Akaike information criterion (AIC) applied for both solutions. The AIC is a measure of the relative quality of statistical models, whose the preferred model is the one with the minimum AIC value [15]. The results show that our approach has the lowest value of AIC and generates a temporal function with a better approximation of the real data trajectory distribution, than the baseline.

5. RELATED WORK

Table 1: Akaike Information Criterion

	One Day	One Week	One Month
Proposed Solution	387.1858	2155.851	3901.251
Baseline	451.4266	2525.758	4553.297

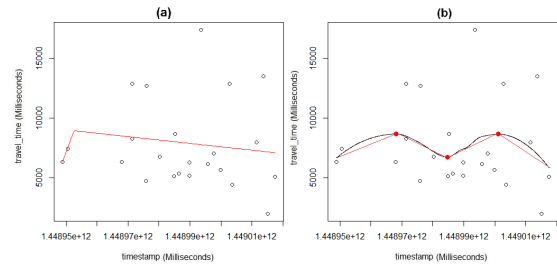


Figure 4: Effectiveness Analysis Scenario I for one Day: (a) Naive Solution and (b) Proposed *Piecewise*.

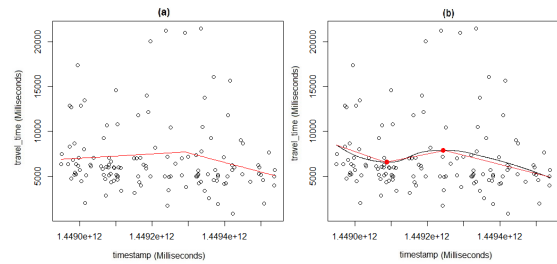


Figure 5: Effectiveness Analysis Scenario II for one Week: (a) Naive Solution and (b) Proposed *Piecewise*.

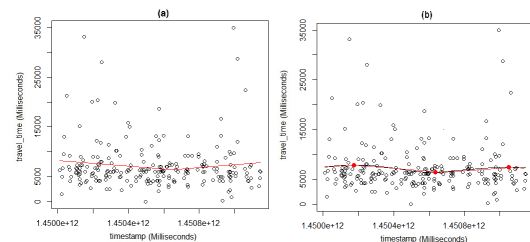


Figure 6: Effectiveness Analysis Scenario III for one Month: (a) Naive Solution and (b) Proposed *Piecewise*.

The time dependent networks may be build using different temporal functions. These networks allows the processing of different queries. Studies about shortest path algorithm can be widely found in works like [1]. Moreover, recently, [2] proposed an algorithm that is based on INE expansion [16] and use an A^* search to guide this expansion and [3] consider the problem of finding the closest point of interest in road networks. Furthermore, to guarantee the accuracy of information, that will be used to build the time dependent network, is important define a map matching process, like [17] [18].

In the literature, the works to resolve shortest path problems have main concern both the accuracy of values and costs inference. [19] propose techniques that enable the construction of a multi-cost, time-dependent, uncertain graph model of a road network. [20] use a deterministic model and assume that the speed, on road segment, is a constant *Piecewise*.

The *Piecewise* regression criterion is employed in our method. These function is extensively studied in order to

solving a nonlinear programming model and capture the salient characteristics of a signal, where linear modeling yields unsatisfactory results. Like others models, the *Piecewise* checks if two or more variables relates. It means, if it is possible define the behavior of one variable observing the changes of other. Authors like [21] apply univariate *Piecewise* linear functions to fit different experiments of data and identify breakpoints that represent critical threshold values.

In the literature, the curves smoothing extends the concepts of interpolation and approximation [22], and both can be used in temporal series. In this work, the curves smoothing are used in the decision-making process, when it is necessary to infer where the *breakpoint* of *Picewise* functions occurs.

6. CONCLUSION AND FUTURE WORK

In this paper, we presented a new method for generating *Piecewise* linear functions. In the proposed method we applied a *Map Matching* technique allied to a curve smoothing approach in order to treat outliers and incomplete data. We discussed our proposed solution and presented experimental results comparing with the Naive solution. Our results show the efficiency and effectiveness using both process: the *Map Matching* and *Piecewise* function. In future work, we aim at investigating the variations of this function considering trajectory data streams, with dynamic weights, where we have frequent updates.

Acknowledgement. Research supported by CNPq and FUNCAP Brazilian funding agencies under project numbers CNPq 451247/2016-7, CNPq 307082/2015-6, CNPq 454727/2014-3, CNPq 552578/2011-8, FUNCAP INRIA ED 06/2015 and FUNCAP scholarship.

7. REFERENCES

- [1] Ugur Demiryurek, Farnoush Banaei Kashani, and Cyrus Shahabi. Efficient k-nearest neighbor search in time-dependent spatial networks. In *Database and Expert Systems Applications, 21st International Conference, DEXA 2010, Bilbao, Spain, August 30 - September 3, 2010, Proceedings, Part I*, pages 432–449, 2010.
- [2] Lívia A. Cruz, Mario A. Nascimento, and José Antônio Fernandes de Macêdo. k-nearest neighbors queries in time-dependent road networks. *JIDM*, 3(3):211–226, 2012.
- [3] Camila F. Costa, Mario A. Nascimento, José Antônio Fernandes de Macêdo, and Javam C. Machado. Nearest neighbor queries with service time constraints in time-dependent road networks. In *Proceedings of the Second ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems, MobiGIS 2013, November 5, 2013, Orlando, Florida, USA*, pages 22–29, 2013.
- [4] William S Cleveland and Eric Grosse. Computational methods for local regression. *Statistics and Computing*, 1(1):47–62, 1991.
- [5] Ariel Orda and Raphael Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *J. ACM*, 37(3):607–625, 1990.
- [6] Damodar N Gujarati and Dawn C Porter. *Econometria Básica-5*. AMGH Editora, 2011.
- [7] Vito MR Muggeo. Segmented: an r package to fit regression models with broken-line relationships. *R news*, 8(1):20–25, 2008.
- [8] Peter Karich. Graphhopper. <http://www.graphhopper.com>, last accessed, 4(2):15, 2014.
- [9] OpenStreetMap Wiki. Main page — openstreetmap wiki,, 2014. [Online; accessed 10-dezembro-2015].
- [10] Táxi Simples. Main page táxi simples. <https://taxisimples.com.br>, 2015. [Online; accessed 5-novembro-2015].
- [11] R. Magalhães, G. Coutinho, J. de Macãdo, C. Ferreira, L. Cruz, and M. Nascimento. Graphast: an extensible framework for building applications on time-dependent networks. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems SIGSPATIAL/GIS, Bellevue, EUA*, page 93, 2015.
- [12] Tracking Táxi Simples. Tracking táxi simples. <https://s3.amazonaws.com/txsimplestracking/index.html>, 2016. [Online; accessed 3-april-2016].
- [13] Geolocation api specification. <http://dev.w3.org/geo/api/spec-source.html>, 2015. Accessed: 2015-12-13.
- [14] Fernando F. Chucre. Mapmatching time travel. <https://github.com/chucre/time-travel>, 2016. [Online; accessed 1-april-2016].
- [15] Hirotugu Akaike. A new look at the statistical model identification. *Automatic Control, IEEE Transactions on*, 19(6):716–723, 1974.
- [16] Dimitris Papadias, Jun Zhang, Nikos Mamoulis, and Yufei Tao. Query processing in spatial network databases. In *VLDB*, pages 802–813, 2003.
- [17] Adel Javanmard, Maya Haridasan, and Li Zhang. Multi-track map matching. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, pages 394–397. ACM, 2012.
- [18] Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On map-matching vehicle tracking data. In *Proceedings of the 31st international conference on Very large data bases*, pages 853–864. VLDB Endowment, 2005.
- [19] Jian Dai, Bin Yang, Chenjuan Guo, and Christian S Jensen. Efficient and accurate path cost estimation using trajectory data. *arXiv preprint arXiv:1510.02886*, 2015.
- [20] Evangelos Kanoulas, Yang Du, Tian Xia, and Donghui Zhang. Finding fastest paths on A road network with speed patterns. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 10, 2006.
- [21] Gihan F Malash and Mohammad I El-Khaiary. Piecewise linear regression: A statistical method for the analysis of experimental adsorption data by the intraparticle-diffusion models. *Chemical Engineering Journal*, 163(3):256–263, 2010.
- [22] Ivo Babuška, Gabriel Caloz, and John E Osborn. Special finite element methods for a class of second order elliptic problems with rough coefficients. *SIAM Journal on Numerical Analysis*, 31(4):945–981, 1994.