Time-Dependent Reachability Analysis: A Data-Driven Approach

Chrysovalantis Anastasiou*, Chao Huang†, Seon Ho Kim*, Cyrus Shahabi*
* *Integrated Media Systems Center*
USC Viterbi School of Engineering
Los Angeles, California, USA
{canastas, seonkim, shahabi}@usc.edu
† *Department of Electrical Engineering*
Tsinghua University
Beijing, China
c-huang15@mails.tsinghua.edu.cn

***Abstract*—An *isochrone* is generally defined as a curve drawn on a map connecting points at which moving objects (e.g., cars) arrive at the same time. Their construction is an important task in many application domains. As an example, in urban planning, isochrones are essential when assessing the placement of public services like hospitals and fire departments. In this study we formally define the isochrone and reverse isochrone problems, describe our approach to solving them and provide a fully functional system that is capable of visualizing the reachability in various ways. Unlike other studies, our approach is purely data-driven and does not depend on the underlying road network for computing the isochrone. Instead, we focus on directly processing trajectory data. Our system processes two real-world taxi datasets to visualize the reachability of the cities of Seoul and Xi'an. As our experiments show, our approach outperforms the traditional graph-theory techniques while eliminating the expensive need of preprocessing the data.**

***Keywords*-Isochrone Maps, Reachability, Urban Planning**

I. Introduction

The actions taken in the initial minutes of an emergency are critical. Several studies recommend that first responders must be at the scene within the first 8 minutes [1], however, the optimal time of their arrival highly depends on the road network and its traffic conditions. In urban planning there is usually the task of assessing the placement of public facilities, such as hospitals and fire stations, to determine whether every city block can be reached from at least one of these facilities within a certain time span.This assessment is typically referred to as *reachability analysis* and aims in detecting blind spots early so that new public facilities can be constructed for their coverage. First introduced in the 19th century and recently re-introduced as a new spatial query type in [2], *isochrone maps* are the most efficient tools used in addressing the reachability analysis problem. An isochrone is defined as a curve drawn on a map connecting points at which objects arrive at the same time. In isochrones there is usually a single source point and a given time span and we are interested in finding the area that can be reached from that point within the given time span.

Due to the ubiquitous availability of geographical and transit-related data, isochrone maps have experienced a renaissance in recent years and have given rise to many data-driven applications of practical value. Most of the previous work in this field was focused on shortest paths, nearest neighbor, and range queries [3] [4]. In graph theory, an isochrone is defined as the set of all vertices that can be reached from a source vertex given a limited path cost equivalent to travel time. An average speed is computed for each edge in the graph and the estimated time of traversal is based on that speed. However, with a real-world trajectory dataset collected by GPS-equipped sensors, the isochrones can be calculated directly from the data instead of estimating the average speed of each edge in a graph, a task that can be largely time consuming. Therefore, the dynamics of daily transportation can be integrated in the calculation of isochrones, providing higher confidence and accuracy for real-world applications.

The accurate and efficient construction of reachability maps is a challenging task. First, we want to analyze the reachability at different times of the day and for that we need to construct time-dependent isochrone maps. Graph theory techniques attempt to solve this by assigning dynamic time-varying weights on edges [5], however, these techniques do not scale well to large and complex graphs such as road networks. Second, normal isochrones do not always convey the information one needs. For example, when analysing the transit system of a city, it is more critical to assess the average commute time in order to get to work and, similarly, in emergency response, we need to understand whether an emergency scene can be reached in time from emergency centers. Approaching this problem with normal isochrones does not yield correct insights because we want to analyze how long it takes for a person to travel towards a single target point instead of away from it, facing, therefore, different traffic dynamics. We refer to this as *reverse reachability analysis* and we use *reverse isochrone maps* as a tool to examine it. Different traffic directions carry different traffic dynamics and therefore normal and reverse isochrone maps with the same parameters will most likely lead to different results.

In this paper, we propose a data-driven grid-based approach for constructing isochrone and reverse isochrone maps. Unlike most of previous work, which relies on graph techniques, our approach directly processes trajectories. To the best of our knowledge, our study is the first that attempts to solve the isochrone and reverse isochrone problems by directly processing trajectory data. As we show in our experiments, our approach achieves much lower response times and is up to 400% faster for time limits greater than 10 minutes, than those of conventional graph approaches, because it is not affected by the complexity of large road networks. As a byproduct, our approach naturally enjoys time-dependent results because trajectories already encode travel times at different times throughout a day. Furthermore, we eliminate the expensive need of mining GPS measurements to estimate time-dependent speeds for each link in the graph; a step that is required for all graph-based techniques.

The contribution of our work can be summarized as follows:

- We address the isochrone problem, for both single-source and multi-source scenarios, which is essential in many urban planning applications.
- We address the reverse isochrone problem, for both single-target and multi-target scenarios, which is critical in many domains.
- We utilize two real-world trajectory datasets to calculate and construct time-dependent isochrone maps and conduct comprehensive experiments to prove that our data-driven approach can achieve much better response times without the expensive need of preparing data for graph processing.
- We provide a comparison between different visualization methods and present the advantages and disadvantages of each method.

II. Reachability Maps

In this section, we describe our data-driven approach to construct time-dependent isochrone maps and how we use the proposed index to answer reachability queries.

A. Reachability Index

The main idea is to find all those trajectories that pass by the given query point and retrieve the segment of each trajectory that falls within the given time limit. However, doing this in an efficient way is not an easy task. A straightforward approach is to build an R-tree [6] index on top of the trajectories. Then, a radius search can retrieve all trajectories that pass by a given query point. A last pass over these trajectories will be required to filter out all those that do not pass by the given source at the given departure time. Therefore, this approach would not scale well for large and dense trajectory datasets simply because the filtering step is an expensive linear search. Hence, in order to efficiently process reachability queries, we first need to quickly identify trajectories that pass close by the query location, and, subsequently, we need to efficiently decide which of those trajectories satisfy the time limit conditions. We propose a grid-based index that can be utilized to efficiently filter and process only those trajectories that are guaranteed to meet the query criteria. First, an empty grid $G(n)$ over the general area of interest, e.g., a metropolitan city, is generated in which all cells have an equal predefined size of n meters. Then, each GPS measurement of the trajectories is assigned to the cell that it falls into. Each trajectory, is essentially a temporal sequence $(gps_1, gps_2, \ldots, gps_k)$ of tuples $gps_i = (loc, ts)$ where $loc = (lat, lng)$ is the location of the measurement in latitude and longitude degrees and ts is the timestamp at which the measurement was recorded. Additionally, in order

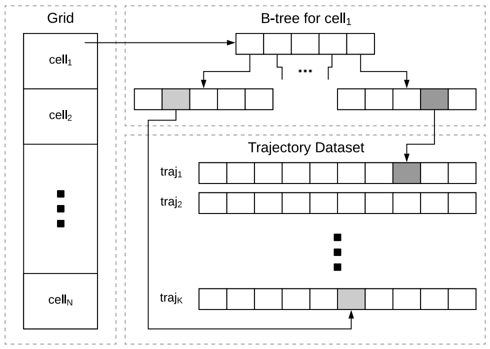


Fig. 1. Indexing scheme of trajectories in our grid-based approach. Each cell is associated with a B-tree whose leaves point to GPS measurements.

to efficiently retrieve only those measurements within a cell that satisfy the given departure (or arrival) time, a B-tree is simultaneously constructed on top of the timestamps. Figure 1 illustrates the aforementioned approach.

B. Query Answering

Our proposed index can be used to process both isochrone and reverse isochrone queries. Traffic conditions can vary highly at different times of a day and, therefore, so do isochrone maps. Our system integrates those traffic dynamics by generating time-dependent isochrone maps and can support four types of time-dependent queries; the Single-Source Reachability Query (SSRQ), the Multi-Source Reachability Query (MSRQ), the Single-Target Reverse Reachability Query (STRQ), and the Multi-Target Reverse Reachability Query (MTRQ). The first two types of query will generate normal isochrones, whereas, the last two will generate reverse isochrones. The following sections explain in detail the steps taken to process each query type.

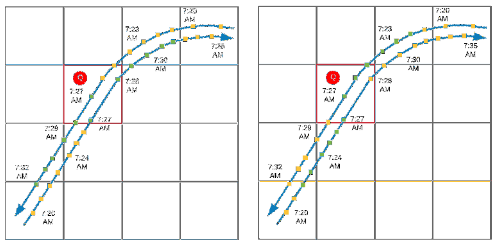
1) Reachability Queries: Formally, in graph theory, an *isochrone* is the minimal, possibly disconnected, sub-graph that covers all the vertices that are within a given time span (or weight budget) from the query source vertex. However, our approach does not consider the underlying road network graph. Instead, our methodology only takes into consideration the trajectories and consists of retrieving all the reachable GPS measurements from them to construct isochrone maps.

A *time-dependent single-source reachability query (SSRQ)* is a tuple $Q = (s, t, d)$, where Q_s is the query source point, Q_t is the departure time, and $Q_d > 0$ is the maximum acceptable time span. The result consists of all location points that are reachable from the source point Q_s within Q_d minutes if one was to depart at time Q_t.

The algorithm in Figure 2 details the steps taken to process a SSRQ query. The procedure receives as input the grid index $G(n)$ and a query Q. First step is to calculate the grid cell c that contains the source point $Q.s$. Line 1 uses a utility procedure, $findCell$, to achieve this goal and due to the simplicity of the grid, this procedure is just a mathematical formula that can be evaluated in constant time. Next, we traverse the B-tree index structure of the cell c to get all

Fig. 2. (Algorithm) Time-Dependent Single-Source Reachability Query

```
Input: $G, Q$
1: $c \leftarrow findCell(G, Q.s)$
2: $r \leftarrow \{\}$ // Initialize result to empty set
3: for $(traj, i) \in c.gpsInWindow(Q.t, Q.t + Q.d)$ do
4:     while $i < traj.length$ and $traj[i].ts \leq Q.t + Q.d$ do
5:         $r \leftarrow r \cup \{traj[i].loc\}$
6:         $i \leftarrow i + 1$
7:     end while
8: end for
9: return $r$ // Return the set of all reachable points
```



(a) Normal Query (b) Reverse Query

Fig. 3. Processing a reachability query with departure/arrival time of 7:27 AM and a time-limit of 5 minutes. Reachable and non-reachable GPS measurements are shown as green and yellow squares, respectively.

trajectory pointers that are within the query's time range, i.e. from $Q.t$ until $Q.t + Q.d$. Subsequently, for each pointer, we load the trajectory it points to, and investigate all the GPS measurements that where recorded from that pointer and after until we find the first measurement that exceeds the time limit. All those measurements that satisfy the conditions are appended to the final result set. Figure 3a provides a simple example of the algorithm for a SSRQ with departure time at 7:27 AM and a time limit of 5 minutes, illustrated using two trajectories with opposite directions. In this example, GPS measurements, shown as small squares, are sampled in 1-minute intervals. Trajectories are being processed from the points that fall within the cell and have a time stamp no later than 5 minutes after 7:27 AM, hence giving a time-dependent aspect to our approach.

However, a SSRQ is not sufficient for the reachability analysis of a city as it is more critical to assess the collective reachability performance of many source points, e.g., transit stations and emergency centers. Therefore, we propose a straightforward solution in which for each point of interest in a multi-source reachability query (MSRQ), a single-source reachability query is executed. The result of a MSRQ is effectively the union of all the SSRQ that are constructed for each source point.

2) Reverse Reachability Queries: As discussed in Section I, in many applications reverse isochrones are more critical. As an example, in the emergency response setting, we are interested in finding those emergency centers that can reach an emergency scene in time. Simply picking the one that is geographically closest to the scene does not imply that the travel time is minimized. By constructing reverse isochrones, we can detect which centers are more suitable for responding to an emergency that takes place at a specific location.

A *time-dependent single-target reverse reachability query (STRQ)* is a tuple $Q = (q, t, d)$, where Q_q is the query target point, Q_t is the latest arrival time, and $Q_d > 0$ is the maximum acceptable time span. The query result consists of all location points from which the target location Q_q is reachable within Q_d minutes and by arrival time Q_t.

Fig. 4. (Algorithm) Time-Dependent Single-Target Reverse Reachability Query

```
Input: $G, Q$
1: $c \leftarrow findCell(G, Q.q)$
2: $r \leftarrow \{\}$ // Initialize result to empty set
3: for $(traj, i) \in c.gpsInWindow(Q.t - Q.d, Q.t)$ do
4:     while $i \geq 0$ and $Q.t - Q.d \leq traj[i].ts$ do
5:         $r \leftarrow r \cup \{traj[i].loc\}$
6:         $i \leftarrow i - 1$
7:     end while
8: end for
9: return $r$ // Return the set of all reachable points
```

The algorithm in Figure 4 details the steps taken to process a STRQ query. The procedure receives as input the grid index $G(n)$ and a query Q and is very similar to Algorithm 2 with one key difference. The condition formula at Line 4 is reversed in the sense that it treats $Q.t$ as a latest arrival time instead of a departure time and the GPS measurements of a trajectory are iterated over in a backwards manner, i.e., from more recent timestamps to older ones. This modification essentially backtracks the steps taken in order to reach the target point. Figure 3b provides a simple example of the algorithm for a STRQ with a latest arrival time of 7:27 AM and a time limit of 5 minutes, illustrated using two trajectories with opposite directions. In this example, GPS measurements, shown as small squares, are sampled in 1-minute intervals. Trajectories are being processed from the points that fall within the cell and have a time stamp no earlier than 5 minutes before 7:27 AM, hence giving a time-dependent aspect to our approach.

Similar to the normal isochrone maps, a STRQ might not always be enough to assess the reachability of a city. During a disaster, for instance, all residents of a city might be instructed to head towards their closest predefined shelter. In such cases, it is critical to assess whether every family in residential areas can travel within a certain time to one of the shelters. This is a case of multi-target reverse reachability query (MTRQ) and we propose a straightforward solution in which for each target location, a STRQ is computed and the result sets are unioned in the end to construct the final result set.

III. Performance Evaluation

On top of our proposed approach, we developed a prototype web application that can be accessed and used by other researchers and city officials. In this section, we describe the functionality as well as the responsiveness of the system through various queries.

A. *Experimental Setup*

All experiments were performed on an Ubuntu 14.04 server equipped with an Intel(R) Xeon(R) CPU E5-2603 at 1.70GHz and 64GB of RAM. The processor has 6 cores (12 threads) with private L1 (32KB for data and 32KB for instructions) and L2 (256KB) caches and a shared L3 (15MB) cache. Two real-world taxi trajectory datasets were utilized to benchmark our data-driven approach. The first dataset contains approximately 600 million GPS measurements generated by taxis in Seoul over a 3 month period and sampled in 60-second intervals (SPARSE). The second dataset contains approximately 750 million GPS measurements generated by taxis in Xi'an over a 1 month period and sampled in 3-second intervals (DENSE). Furthermore, a grid $G(n)$ with $n = 100$ meters was used.

B. *Web Application*

For constructing reachability maps in our web application, we utilize our real-world taxi trajectories. The user can select the day of week, the time span, the departure or arrival time, and one of three visualization methods as input parameters. Then, a time-dependent reachability map is constructed on-the-fly. Figure 5 illustrates the different visualization types that are supported by the prototype.

C. *Visualization*

The result of a reachability query is generally the set of all locations that satisfy the given conditions but, in our system, is instead the set of all discrete GPS data points that satisfy the conditions. This result alone does not provide a lot of insight unless visualized properly on a map. Reachability maps can be presented in a variety of ways with each representation offering a different view of the result. Hence, we implement and compare three methods, i.e., the reachable area by convex hull, the reachable cells, and the reachable buffer around the trajectories that satisfy the query. In an emergency response situation, very often the first responders, after they arrive with a vehicle at the scene, they also have to walk for some distance before arriving to their final destination. Reachable cells and reachable buffers attempt to facilitate this bimodality.

1) Reachable Area Convex Hull (CH): One of the most natural and conventional ways of visualizing the reachable area is by calculating the convex hull. The result is a polygon that spans the reachable area and it is simple enough to be rendered in negligible time. Plotting this polygon on a map can offer quick insights for single-source queries. Figures 5a and 5d illustrate the convex hull visualization of normal and reverse isochrones in our system.

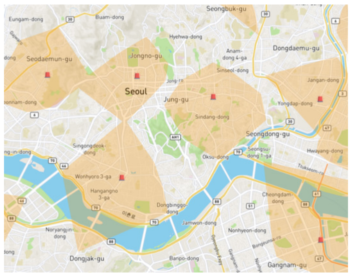
2) Reachable Cells (RC): The second visualization technique aims at providing tighter bounds to the reachability polygons. In both single-source and multi-source variants, after the reachable data points are retrieved, a list of distinct cells and their bounds is computed by using the grid index $G(n)$. In this technique, rendering is not as cheap as in the case of convex hulls, however, the shapes are still regular enough to be rendered quickly. In practice, we find that a grid cell size of 100 meters is ideal because, even if it includes a part that is not reachable within the required time span, it can be covered with no more than 2 extra minutes of walking, assuming an average walking speed of 1.6 m/s. This is of paramount importance in the context of emergency response as first responders usually have to unload when the reach a scene and walk to their final destination from there. Figures 5b and 5e illustrate the reachable cells visualization of normal and reverse isochrones in our system.

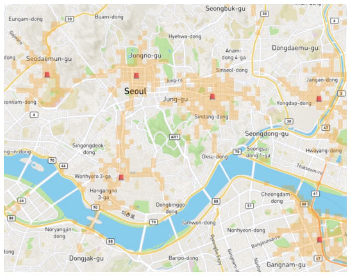
3) Reachable Trajectory Buffers (TB): The above two methods of visualization are based on 2D points to construct isochrones. However, since each trajectory in our database is a 2D link, breaking each link to a set of discrete location points loosens the edge information and reduces the accuracy of the isochrone computation. Therefore, we follow the work in [7] and process the trajectories to create a buffer around them that includes those areas within immediate walking distance. For simplicity, we assign the radius of the buffer to 100 meters, a distance that can be walked by an average human in a minute or so. The advantage of computing isochrone maps using trajectory buffers is two-fold. First, it can model the hybrid modality of both walking and driving, and second, the area it creates has better real-world interpretation and generalization, such as multi-floor buildings, since it preserves the trajectory information. This, however, comes with the cost of additional computation and rendering times. Figures 5c and 5f illustrate the trajectory buffers visualization of normal and reverse isochrones in our system.

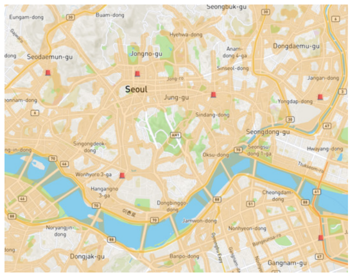
All three visualization techniques have their advantages and disadvantages. The CH technique seems to be more suitable for large and dense trajectory datasets because it reduces the amount of information that need to be communicated while requiring less computation time. On the other hand, TB are ideal for small and sparse trajectory datasets because due to the small number of trajectories the required computation is low. Furthermore, the TB will provide a more accurate representation whereas, in this case, a CH might be covering a lot of areas that are unreachable.

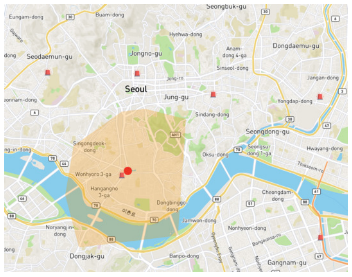
D. *Query Response Time*

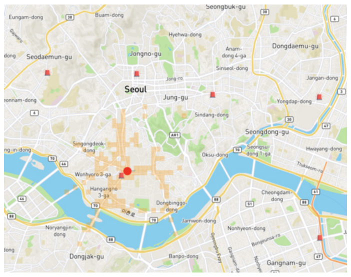
First, we compare our grid-based Data-Driven Approach (DDA) with a traditional Graph-Based Approach (GBA) and show that the former can achieve much better response times without the expensive need of a preprocessing step to compute time-dependent edge weights. For this experiment, we process the real-world taxi dataset of Seoul, to estimate hourly time-dependent speeds on the road network edges of the city using the techniques described in [8]. Subsequently, the GBA using Dreyfus's [9] algorithm is employed to answer reachability queries. During the evaluation of both the GBA and DDA solutions, all the data were loaded into memory and the actual CPU processing time was measured. For normal isochrone queries we measured the CPU time of a MSRQ originating from 23 fire stations across the city of Seoul, whereas, for


(a) Convex Hull


(b) Cells


(c) Buffers


(d) Convex Hull Reverse


(e) Cells Reverse

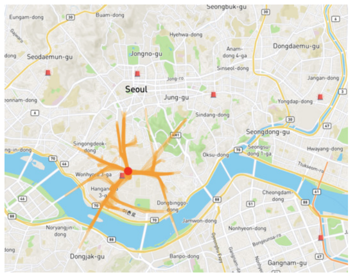

(f) Buffers Reverse

Fig. 5. Visualization of reachability in Seoul on a Monday morning 7AM - 8AM for a 5 minute time span.

reverse isochrone queries we measured the CPU time of a STRQ targeting the busiest location in the city, i.e., the grid cell with the highest number of GPS measurements. To further stress the performance, data from all the days of the week were taken into consideration, i.e., the day of the week parameter was not provided. Figure 6 plots the response time as a function of the time limit parameter d and compares GBA and DDA. The result shows that as we increase d, Dreyfus provided a longer latency simply because exponentially more edges of the graph need to be explored, whereas, DDA maintains a nearly constant response time due to the fact that there is always a constant number of trajectories intersecting the query grid cell. The same observation holds for both MSRQ and STRQ isochrones.

Second, in order to assess the performance of different visualization types, we use the metric of end-to-end query response time which is defined as the elapsed time starting from submitting the query until receiving the results, therefore including both computation, network latency, and rendering times. The elapsed time measure was averaged over the 7 days of the week, i.e., for each departure time 7 queries were executed, one for each day. For the sparse dataset of Seoul, a MSRQ originating from the 23 fire stations in the city was used to evaluate the response time of normal isochrones and a STRQ was used to evaluate the response time of reverse isochrones. Figures 7a and 7b plot the end-to-end query response time of MSRQ and STRQ, respectively, at different

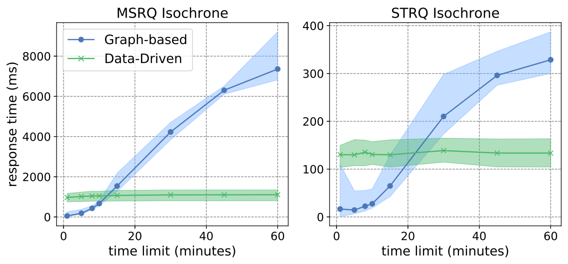
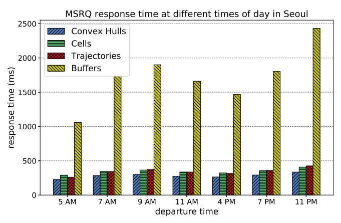
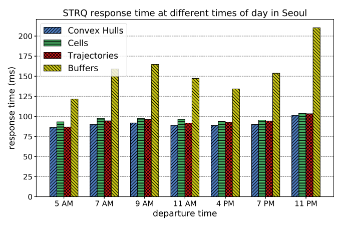


Fig. 6. Average query response time and standard deviation of GBA and DDA for different time spans across all day.

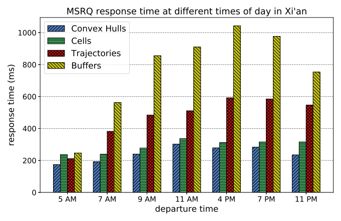


(a) MSRQ from 23 fire stations

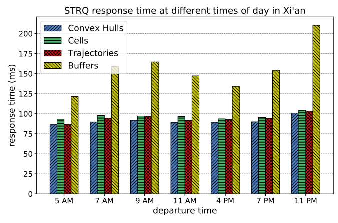


(b) STRQ for busiest grid cell

Fig. 7. End-to-end average query response time in milliseconds for all visualization methods of an isochrone query at different times of the day in Seoul.



(a) MSRQ 4 fire stations



(b) STRQ for busiest grid cell

Fig. 8. End-to-end average query response time in milliseconds for all visualization methods of an isochrone query at different times of the day in Xi'an.

times of a day for a time limit of 8 minutes using the trajectory dataset of Seoul. Similarly, Figures 8a and 8b plot the end-to-end query response times for a MSRQ originating from 4 fire stations in the city and a STRQ, respectively, using the dense trajectory dataset of Xi'an. In the dataset of Seoul, the trajectories are almost uniformly distributed over the different times of the day and we do not observe large variations in the response time of CH, RC, and TB visualizations. The TB visualization is always the slowest since generating the buffers around the selected trajectories is an expensive operation. In the dataset of Xi'an most of the trajectories fall in the 9am to 7pm window, i.e. during the time that people are more active. Figure 8a shows that even though the number of trajectories increases, the response times for the CH and RC visualizations do not vary a lot, whereas, for the TB visualization it increases. We observe that the CH representation consistently enjoys the quickest response time because its shape is fairly simple and hence not much information needs to be transmitted over the network but also because its computation is inexpensive. On the other hand, even though they offer a more accurate representation, TB are usually much slower compared to the other techniques. This shows that as the shape of an isochrone map gets more complicated and less regular, the representation covers reachable areas in a more accurate way, although with an increase in computation time and network latency.

IV. Related Work

Most of the previous work about isochrone maps relies on graph techniques for estimating the shortest path from the query point. The road network is represented as a graph with nodes typically being intersections and edges being the streets. Raw location data, gathered from GPS-equipped sensors are processed in order to estimate the average speed of each edge in the graph. Since, GPS measurements are discrete and not completely accurate, a map-matching algorithm [8] needs to be employed in order to calculate the exact path of travel and the edges that were involved. Lastly, time-dependent average speeds can be calculated by estimating the time spent on each link. After that, a shortest path algorithm is employed for detecting all the reachable vertices before an isochrone map is constructed. A recent study [10] proposed a novel indexing scheme that adds temporal awareness to traditional query processing methods. The authors utilize trajectory datasets to generate a set of indexes that speed up reachability queries, however, they still include the expensive step of map-matching. The key difference in their approach is that instead of computing a reachable area, they focus in computing reachable road segments of the graph.

Isochrone maps are also extremely helpful for measuring the accessibility of critical services where time optimization can make a difference. In [11], the authors conduct a case study about the quality of living in Slovakia, focusing on location reachability from the hospital of a city. A web application called *iso4app* is utilized for generating isochrone maps along a specific bus line using estimated velocity. However, this work still has limitations, since it does not take into consideration the entire public transport network of the city and only convex hull is considered for visualization.

The study in [7] discusses the most appropriate shape for representing isochrones as areas when querying the spatial network database. The authors propose two algorithms for transforming an isochrone network to an isochrone area. The main difference between our work and theirs is that we utilize real-world trajectories and consider a hybrid transportation model instead of only walking mode at a constant average speed of 1.6 m/s.

V. Conclusion

In this paper we presented our work in constructing and visualizing both isochrone and reverse isochrone maps. We proposed a data-driven grid-based approach that directly processes trajectories to answer four types of queries. Our work, instead of depending on the traditional road network approaches, utilizes real-world trajectory datasets to approximate the reachability of a city at different times of a day and enjoys an average speedup of 400% on queries with a time limit greater than 10 minutes. Furthermore, we provided a comparison of different visualizations and discussed the advantages and disadvantages of each type.

Acknowledgements

This work was supported in part by NSF grants IIS1320149 and CNS-1461963, the USC Integrated Media Systems Center, and unrestricted cash gifts from Google, and Oracle.

References

[1] P. T. Pons and V. J. Markovchick, "Eight minutes or less: does the ambulance response time guideline impact trauma patient outcome?" *The Journal of Emergency Medicine*, vol. 23, no. 1, pp. 43 – 48, 2002.

[2] V. Bauer, J. Gamper, R. Loperfido, S. Profanter, S. Putzer, and I. Timko, "Computing isochrones in multi-modal, schedule-based transport networks," in *GIS'08*, ser. GIS'08. New York, NY, USA: ACM, 2008, pp. 78:1–78:2.

[3] E. Kanoulas, Y. Du, T. Xia, and D. Zhang, "Finding fastest paths on a road network with speed patterns," in *ICDE'06*, April 2006, pp. 10–10.

[4] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network databases," in *VLDB'03*, ser. VLDB '03. VLDB Endowment, 2003, pp. 802–813.

[5] B. Ding, J. X. Yu, and L. Qin, "Finding time-dependent shortest paths over large graphs," in *EDBT'08*, ser. EDBT '08. New York, NY, USA: ACM, 2008, pp. 205–216.

[6] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *SIGMOD'84*, 1984.

[7] S. Marciuska and J. Gamper, "Determining objects within isochrones in spatial network databases," in *ADBIS'10*. Springer, 2010, pp. 392–405.

[8] P. Newson and J. Krumm, "Hidden markov map matching through noise and sparseness," in *GIS'09*, ser. GIS '09. New York, NY, USA: ACM, 2009, pp. 336–343.

[9] S. E. Dreyfus, "An appraisal of some shortest-path algorithms," *Oper. Res.*, vol. 17, no. 3, pp. 395–412, Jun. 1969.

[10] G. Wu, Y. Ding, Y. Li, J. Bao, Y. Zheng, and J. Luo, "Mining spatio-temporal reachable regions over massive trajectory data," in *ICDE'17*, April 2017, pp. 1283–1294.

[11] G. Lugano, M. Hudák, K. E. Ambrosch, and T. Kováčiková, "Travel time, resource reachability and quality of living in urban contexts: A case study from slovakia," in *SCSP'17*. IEEE, 2017, pp. 1–6.