GTI: A Scalable Graph-based Trajectory Imputation

Keivin Isufaj[1] Mohamed M. Elshrif[1] Sofiane Abbar[1] Mohamed F. Mokbel[2]
[1] Qatar Computing Research Institute, Hamad Bin Khalifa University, Qatar [2] University of Minnesota, USA
keisufaj,melshrif,sabbar@hbku.edu.qa,mokbel@umn.edu

ABSTRACT

GPS-enabled devices, including vehicles, smartphones, wearable and tracking devices, as well as various check-in and social network data are continuously producing tremendous amounts of trajectory data, which are used consistently in many applications such as urban planning and map inference. Existing techniques for trajectory data imputation rely heavily on the existing maps to perform map-matching operations. However, modern applications such as map construction and map update assume no map exists. In this paper, we propose GTI - a scalable graph-based trajectory imputation approach for trajectory data completion. GTI relies on cross-trajectory imputation, as it exploits "mutual information" of the aggregated knowledge of all input sparse trajectories to impute the missing data for each single one of them. GTI can act as a pre-processing step for any trajectory data management system or trajectory-based application, as it takes raw sparse trajectory data as its input and outputs dense imputed trajectory data that significantly increase the accuracy of different systems that consume trajectory data. We evaluate GTI on junction-scale as well as city-scale real datasets. In addition, GTI is used as a pre-processing step in multiple trajectory-based applications and it boosts the accuracy across these applications compared with the state-of-the-art work.

CCS CONCEPTS

• **Information systems → Spatial-temporal systems**.

KEYWORDS

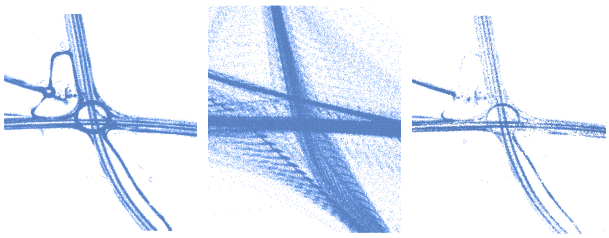
trajectory imputation, GPS, spatial data, road network

ACM Reference Format:

Keivin Isufaj[1] Mohamed M. Elshrif[1] Sofiane Abbar[1] Mohamed F. Mokbel[2] . 2023. GTI: A Scalable Graph-based Trajectory Imputation. In *SIGSPATIAL '23, November 13-16, 2023, Hamburg, Germany, © 2023 Association for Computing Machinery.*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3589132.3625620

1 INTRODUCTION

GPS-enabled devices, including vehicles, smart phones, wearable and tracking devices, as well as various check-in and social network data are continuously producing tremendous amounts of trajectory data [40]. In general, a trajectory is represented by a sequence of GPS samples where each sample has spatial and temporal coordinates, but due to the power cost of these GPS-enabled devices, most recorded trajectory data are stored sparsely [39]. Trajectory imputation solves this issue by reconstructing missing GPS samples and enriching sparse trajectories with additional dense samples, resulting in the restoration of the original recording frequency.



(a) Raw "Dense" (b) Linear Interpolation (c) GTI

Figure 1: Effect of GTI on improving quality of trajectories.

The recent explosion of such trajectory data has empowered countless of principal trajectory operations that are heavily employed in highly important services used by millions of people on a daily basis. These principal operations include trajectory data management [13] trajectory similarity [34] and trajectory clustering [37]. In addition, many services that are based on such trajectory operations, including map inference [8, 18, 33], map matching [21, 24], and travel time estimation [20, 30] necessitate dense trajectories in terms of spatial and temporal attributes. It has been practically shown that many of these services and applications suffer from a low accuracy [9, 19, 36] that not only hinders its wide spread use, but it also causes severe accidents [6, 19, 29]. The main reason behind such sub par performance is the sparsity and the inaccuracy of the input trajectory data. As a recorded trajectory comes from GPS-enabled devices of inherent inaccuracy due to battery savings, it is common to have sparse trajectory data where there are large spatio-temporal gaps between consecutive samples as well as off-road inaccurate samples [15, 19].

As a means of providing high quality services, there have been many attempts to increase the accuracy of collected trajectory data by filling the gaps between consecutive trajectory samples; a process that had various names, including trajectory imputation [10, 14], trajectory completion [22], trajectory data cleaning [38], or trajectory interpolation [23]. Unfortunately, one major drawback of such attempts is that they rely on road network data. Therefore, they get invalidated in cases when the road network is unavailable or changed (as in the case of bicycle traffic). Wang et al. [36], highlights that ~15% of the roads around the world undergo changes each year. The challenge here is that traditional techniques for increasing the accuracy of trajectory data are inapplicable as

The work of this author is partially supported by the National Science Foundation, USA, under Grants IIS-1907855 and IIS-2203553.

SIGSPATIAL '23, November 13–16, 2023, Hamburg, Germany

ACM ISBN 979-8-4007-0168-9/23/11. . . $15.00
https://doi.org/10.1145/3589132.3625620

Figure 2: Example of applying a range query at three intersections. Background map is shown only for reference.

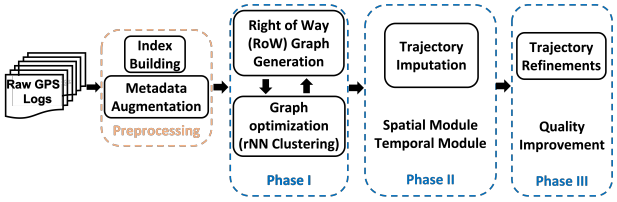


Figure 3: Overview of the GTI framework.

these techniques rely on the map itself, which is not always available. Recent techniques that are agnostic of the underlying road networks have been presented [14, 22]. These techniques work well in small regions in the absence of road network. However, they are not scalable enough to work in large road networks, time consuming, or unable to produce reliable temporal data.

In this paper, we propose GTI a Graph-based approach for Trajectory data imputation. GTI relies on the "mutual information" of aggregated knowledge of all input sparse trajectories to impute the data for each single one of them. The main idea of GTI is to create a connected directed graph from the sparse trajectories and find the shortest-path between every two successive GPS samples by traversing the graph nodes. This means that GTI exploits existing knowledge of neighboring trajectories to densify the target trajectory - without any prior knowledge of the underlying road network. GTI complements spatial imputation with temporal imputation. Hence, GTI would act as a pre-processing step for any trajectory data management system or trajectory-based application.

Figure 1 (a), shows a road network that has been inferred using the raw "dense" GPS samples, whereas Figure 1 (b), depicts the inferred road network of sparse trajectories using linear interpolation. It is clear that the main problem of trajectory imputation solutions based on linear interpolation method is that GPS samples get generated in areas where there are no roads. This obviously penalizes many applications, especially when the underlying maps are outdated, or not available to them. Therefore, GTI solves this issue by imputing samples only in areas where traffic was previously reported. Clearly, as shown in Figure 1 (c), GTI imputed samples are very similar to the dense dataset representation. To show the importance of GTI, here is one trajectory-related application:

Traffic flow management: Since the raw trajectory data is sparse, using simple imputation techniques such as linear interpolation is far from approximating the actual route traversed by a moving vehicle. In addition, estimating capacities and the resulting vehicle delays at intersections can not be assessed correctly. For example, Figure (2), shows an example of applying a range query at three intersections with the goal of monitoring vehicles passing through these intersections using linear interpolation and GTI framework. As visually implied, linear interpolation misses all the intersection range queries, while GTI is able to capture that a vehicle passes through all three intersections.

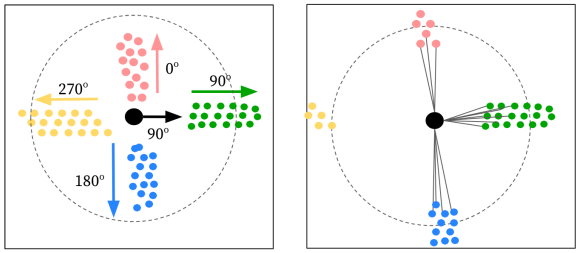
The main idea of GTI relies on cross-trajectory information, where any GPS sample can consult its neighboring samples, in spite of their belonging trajectories, to guide the imputation process without any knowledge of the underlying road network. Hence, GTI exploits all input trajectory dataset to build the graph and finds the route between every two consecutive sparse GPS samples. While individual trajectories can be noisy, when plotting a large number of them together, it becomes easy to visually spot the approximate position of roads (as you will see in the Experiments Section 7). Thus, GTI borrows this interesting concept of *Right-of-Way (*RoW*)* that is widely used in civil engineering literature to refer to the piece of land that is planned to host future road constructions. That is, if a road is to be built in the future, it should be built within the RoW boundaries. We argue that when enough sparse trajectories are available, their corresponding GPS samples can be used to create some sort of a RoW graph that GTI can utilize to figure out the right position of missing GPS samples.

The GTI framework accepts a sparse GPS trajectory data, where as a pre-processing step, it associates metadata (i.e., angle and speed) to every GPS sample. Then, it builds a directed connected graph from the input sparse trajectory data. Next, it utilizes the RoW graph to impute sparse trajectories by inferring missing GPS samples between consecutive distant samples. GTI enriches the spatial imputation with temporal imputation. Finally, it refines imputed trajectories to improve their quality. In case the input sparse trajectory data is huge, for instance, in large-scale cities, GTI has an optimization module that can be exploited to increase the efficiency and effectiveness of the imputation process.

We perform extensive experimental results based on real trajectory datasets, two public [2, 28] and one private that was collected from a real deployment of GTI within the QARTA system [3, 27], that show: (a) GTI is highly scalable to a city-wide trajectory imputation, (b) GTI is very efficient in imputing large amount of sparse trajectories, (c) GTI imputed trajectories resemble the originally raw dense trajectories that were made sparse for experimental evaluation purposes, (d) GTI imputed GPS samples have high accuracy when compared to a ground truth map, obtained from OpenStreetMap [1], and (e) GTI was able to significantly boost the performance of four different trajectory-based applications: map inference, trajectory similarity and search engine, and estimated arrival time, when it was used as a preprocessing step in these applications.

2 GTI FRAMEWORK

This section describes the GTI framework for imputing GPS trajectory data without any knowledge of the underlying map (i.e., road network). The input to GTI is a collection of *sparse* GPS trajectory data, each of which is a sequence of: trajectory_ID, latitude, longitude, and timestamp. The output is a *dense* imputed GPS trajectory data with additional contextual information represented in bearing angle (i.e., vehicle moving direction) and speed. As shown in Figure (3), after preprocessing the input data, there are three phases that comprise the GTI framework, namely, RoW graph generation with an optimization step, trajectory imputation, and trajectory refinements, which can be highlighted as follows:



(a) Edge distance without angle penalty (b) Edge distance with angle penalty

Figure 4: Effect of the angle penalty θ on edge distance. Black dot is the sample being considered for imputation. Other dots are neighbors colored by their movement direction.

Preprocessing: GTI preprocesses input sparse trajectories by associating it with metadata (i.e., contextual information). Hence, GTI takes the raw trajectory data which consists of <*vehicle_id, latitude, longitude, timestamp*> and appends to each GPS sample the bearing angle and its speed [14]. The estimated metadata will be utilized by the next phases of GTI framework.

RoW graph generation: [Section 3], is responsible for creating a directed connected graph that GTI uses as a right-of-way to find a plausible path that connects two consecutive trajectory GPS samples (s_i, s_{i+1}). The graph vertices represent GPS samples and the graph edges represent the plausible path between two vertices. Here, the estimated angles are exploited during the RoW graph generation, to penalize abrupt changes in direction of a moving vehicle.

Graph Generation Optimization: [Section 4], is responsible for extracting representative GPS samples from the input trajectory dataset. Based on the size of input trajectory data and its nature, this step may used to boost the efficiency of GTI framework.

Trajectory imputation: [Section 5], uses the RoW generated graph to perform the spatial data imputation by performing a path search. Based on the distance of the imputed samples and the approximated speed at that geolocation, which is computed by utilizing the mutual information of qualified neighbors, a timestamp is also attached as a temporal coordinate.

Trajectory refinements: [Section 6], refines the imputed trajectories by removing different types of noise to produce cleaned dense trajectories. The input here is the noisy spatio-temporal imputed trajectories and it outputs more elegant imputed trajectories.

3 ROW GRAPH GENERATION

Objective: Build a directed connected graph from sparse GPS trajectory data.

Main Idea: The main idea of this phase is to construct a connected directed graph $G = (V, E)$ out of the sparse GPS samples included in the input trajectories. The set of vertices (V) is formed by all unique GPS samples that are provided in the input sparse trajectories. We create the set of edges (E) by linking all pairs of vertices, i.e., GPS samples, whose distance is equal or below a predefined distance threshold (d). However, not all such edges are optimal to build a path. Therefore, we penalize edges that have a large impact on the change of bearing angle of the current sample. As shown in Figure (4), if a sample is going east (black dot), we do not want to suggest candidate edges that change the direction to west (yellow dots). To achieve that, we replace the distance function by a modified distance metric

Algorithm 1 RoW Graph Generation Algorithm

```
1:  Procedure: GraphGeneration(𝒯, d, θ)
2:  V ← uniquePoints(𝒯)
3:  E ← {}
4:  for s_i ∈ 𝒱 do
5:      rNN(s_i) ← RangeQuery(s_i, V, r)
6:      if rNN(s_i).isempty() then
7:          d = d(s_i, s_{i+1})
8:          rNN(s_i) ← RangeQuery(s_i, V, r)
9:      end if
10:     for s_j ∈ rNN(s_i) do
11:         if d_θ(s_i, s_j) ≤ d then
12:             E ← E ∪ newEdge(s_i, s_j)
13:         end if
14:     end for
15: end for
16: Return G =< V, E >
```

for a candidate edge (s, t) [33]. Consequently, the new distance between two consecutive GPS samples can be calculated as follows:

$$d_\theta(s_i, s_t) = \sqrt{d(s_i, s_t)^2 + (\theta \frac{d_0(\alpha_i, \alpha_t)}{180^o})^2} \quad (1)$$

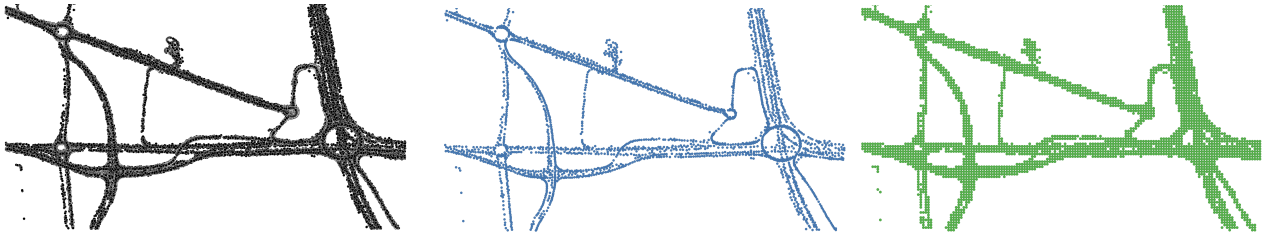
Where $d(s_i,s_{i+1})$ is the regular distance of an edge, $d_0(\alpha_i,\alpha_t)=min(|\alpha_i-\alpha_t|, 360^o\text{-}|\alpha_i\text{-}\alpha_t|)$ is the unit circle distance between bearing angles of two consecutive GPS samples, and θ is the angle penalty.

In addition, it is worth noting that the output graph needs to satisfy the connectivity property. To ensure that, we check if the set of neighbors derived from the range query is not empty. In case there are no neighbors within a radius d, we change the radius to $d = d(s_i, s_{i+1})$, to make sure that in the worst case scenario, we connect the sample to the next one in the trajectory.

Algorithm: As indicated in Algorithm (1), the data input to the graph construction algorithm is a collection of n sparse GPS trajectories ($\mathcal{T} = \{\Gamma_1, \ldots \Gamma_n\}$), a radius distance d and an angle penalty θ. The GPS samples of each trajectory $\Gamma = \{s_1, \ldots s_m\}$ are chronologically sorted in the ascendant order of their associated time stamps. First, we computed the bearing angle of each GPS sample. Then, the graph is constructed in two steps. First, we union all GPS samples in $\mathcal{T}$ and unique them to create the set of vertices V. We can optionally optimize the generated graph to reduce the size of V. We will show in the next section, how different clustering algorithms can easily be plugged here to reduce the size of the graph, and hence improve the empirical running time of subsequent operations. Second, we generate edges between vertices as follows: For each sample $s_i \in V$, we run a range query centered on s_i with a radius d to retrieve all neighboring samples $(rNN(s_i))$. We then calculate the new distance (see Equation 1), for each of the neighboring samples, considering the difference between the bearing angles. The neighbors that have a distance larger than our threshold d get discarded, while the rest form an edge with s_i.

4 GRAPH GENERATION OPTIMIZATIONS

Objective: The main objective of this module is to improve efficiency of the RoW graph generation phase.



(a) Dense high precision GPS samples (#196,525) (b) r-NN based clustering (#2,510) (c) Four digit precision GPS samples (#3,167)

Figure 5: Impact of different optimization techniques.

Main Idea: When the input trajectory data is huge, which could represent a large city, generating the corresponding graph becomes a bottleneck. Hence, graph generation could be optimized. So, depending on the size of trajectory data and its nature, some optimizations may help boost the efficiency of GTI framework.

Indeed, depending on the number of trajectories and the precision of GPS readings (number of floating digits), one can quickly number millions of candidate GPS samples to be vertices in the RoW graph. Therefore, with millions of vertices and probably one order of magnitude more edges (i.e. tens of millions), one can quickly hits some memory issues. Besides this, searching for paths that connect consecutive samples in sparse trajectories can turn to be extremely slow. Figure (5), shows the impact of different GPS sample clustering techniques on the size of RoW graph. Figure 5 (a) is an example of a high precision input to GTI framework for a small area of interest. After applying **r-NN based clustering**, even though the number of GPS samples drops $\sim$ 80 times, the data quality is not highly impacted, as visually described in Figure 5 (b). Contrary, naively reducing the precision of the data to less significant digits tends to be worse than clustering, resulting in a lower quality dataset and loss of road network attributes, such as roundabout structures as shown in Figure 5 (c). Here's how r-NN based clustering could be used to optimize graph generation.

Algorithm: r-NN based clustering: Given the raw input GPS trajectories, we start with an empty centroids list (i.e., representative GPS samples) and go over all input GPS samples sequentially. Representative samples were chosen randomly and serve as cluster centroids. A GPS representative sample "seed" is added to the list of centroids if there are no other centroids within a specific radius. In case that there are two adjacent seeds, we take only one. This process ensures that the resulting centroid list is uniformly distributed throughout the search space, i.e., the space that covers all input GPS trajectories. Then, for each cluster centroid, a query point is issued to report all neighboring GPS samples within a specific threshold radius (r), e.g., 20 meters. After that, the seed and its neighboring GPS samples are removed from the search space. Since the number of GPS samples is huge, we accelerate the searching process by using a $kd-Tree$ structure for quick nearest-neighbor lookup [26]. By doing this sampling/clustering step, the total number of original GPS samples, which should be considered during the imputation process, is dramatically decreased $\sim$%90, from order of millions (10^6) to order of thousands (10^3), or even order of hundreds (10^2). This means that the clustering step optimizes the usage of memory and accelerates the running time, but it sacrifices the accuracy, as you will see in the Experimental Section 7.

5 TRAJECTORY IMPUTATION

Objective: The main objective of this phase is to exploit the RoW graph created in the previous phase in order to impute sparse trajectories by inferring missing samples between consecutive distant samples.

Main Idea: The main idea of the imputation is to map each pair of consecutive samples into their corresponding vertices in the RoW graph. Then, use graph traversal techniques to find a possible path that links the two vertices. This process is quite similar to the traditional map-matching technique, where people use the existing map (represented as a graph) to link discrete trajectory samples. However, in GTI, we assume that the map is not available.

Algorithm: Trajectory imputation phase has two main components, which can be explained as follows:

- **Spatial imputation module:** The spatial imputation process is conducted by finding the shortest path between every pair of consecutive GPS samples in the sparse trajectory data by traversing the connected neighboring vertices, i.e. representative GPS samples, on the generated graph (RoW). The algorithm used to find the shortest path is the Dijkstra's shortest path first algorithm [12]. However, if we limit ourselves to distance only, most of the imputation will be positioned on the end-sides of the roads each time there is some sort of a turn. Ideally, we would like the imputed samples to be on the center-line of roads. To this end, we introduce a new weighting function to score graph edges by incorporating knowledge about the density of the GPS samples (i.e, graph vertices) forming each edge. Our intuition is that samples that lay on road center-lines achieve higher density as compared to peripheral samples on the side. Figure (6) shows an example, where we project GPS samples from a road in Doha dataset onto a 2D scatter plot. After aligning the direction of the movement to the north, we draw a histogram that shows the distribution of GPS samples that are positioned within the road. As indicated, there is a higher density on the center of the road compared to the sides of it. To signify that within our GTI framework, we define the density of a vertex s_i as follows:

$$density(s_i) = \frac{||rNN(s_i)||}{avg(||rNN(s_k)_{s_k \in rNN(s_i)}||)} \quad (2)$$

This means that the density of each vertex is evaluated based on the density of its neighbors. We use the density score of each vertex

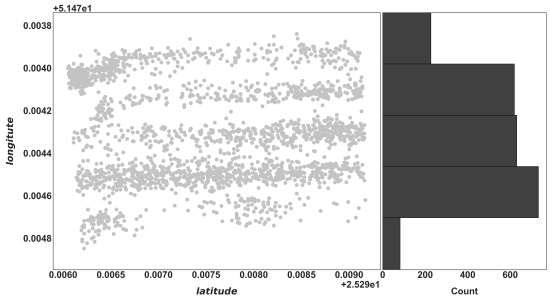


Figure 6: Histogram of GPS samples positioning along a road in Doha dataset.

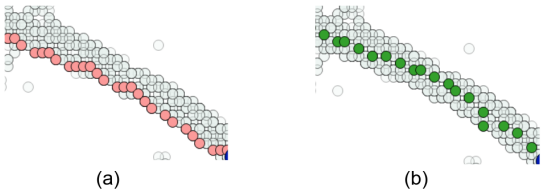


Figure 7: Distance based (red dots) vs. Density-based shortest path (green dots) GTI imputation. Grey dots represent GPS samples across all neighboring sparse trajectory data.

to re-evaluate the weight of edges, $e = (s_i, s_j)$, as follows:

$$weight(s_i, s_j) = length(s_i, s_j) + \lambda \times \frac{1}{density(s_j)} \quad (3)$$

That is, the higher the density, the lower the penalty applied to the edge e; λ is a constant factor we used to regulate the amplitude of the density-induced penalty. Figure (7), shows the differences between distance-based (a) and our density sensitive (b) trajectory imputations. It is clear that accounting for vertex density does have an impact on moving the imputation to the road center-line.

- Temporal imputation module: To enrich the spatial imputation with time stamps, for each imputed GPS sample, we need to know what the speed of that sample is. In the pre-processing step, we already added speed as part of the metadata to each GPS sample. However, the process of estimating the actual speed of a moving vehicle is not straight forward like the bearing angle computation process. This is because other associated factors, such as the travel time and road conditions, vary over time and space. For example, the vehicle speed during the working day is different from the speed if the vehicle passes the same street, but during the weekend. Similarly, the road conditions vary when the vehicle is moving during rush hours (e.g., morning, or afternoon), or at night. To support that, Figure (8) illustrates the distribution of the number of vehicles over the week days derived from all GPS samples (~ 23 million) from Doha dataset. The pattern of the distribution has a similar trend during the weekdays (Sunday - Thursday). However, it has a very different representation during the weekend days. In addition, the metadata differs across moving vehicles which traverse the same path even at similar times. This means that it is not straight forward to use the information of the input GPS samples only. Instead, we rely on using the idea of having dynamic travel cost distributions. We start by inferring this contextual information on every imputed sample by using the following scheme:

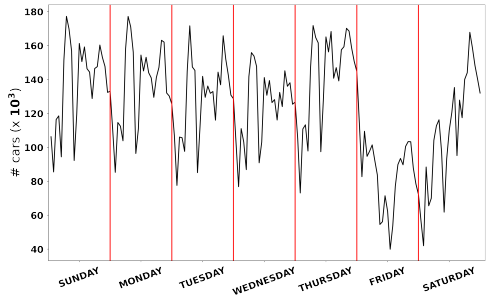


Figure 8: Distribution of the number of vehicles during week days.

• Calculate the speed of every sample in the sparse trajectory given the spatial geolocation, i.e., latitude and longitude, of itself and the successor sample as well as both timestamps.
• Discretize the week into *weekdays*, *weekend_1*, *weekend_2* and each one of them into 24 time slot intervals (one for each hour).
• For each sample s_i define the *speed* of that sample to be the average speed of the neighboring samples within a radius r that fall in the same time slot interval.

Now that we have a better estimation of what speed the vehicle should move at that geolocation, we can easily estimate the time of arrival from s_i to a neighbor sample s_j by dividing their spatial distance by the acquired speed as follows:

$$s_j.timestamp = s_i.timestamp + \frac{d(s_i, s_j)}{s_i.speed} \quad (4)$$

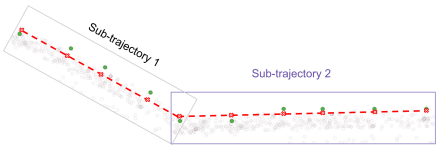
6 TRAJECTORY REFINEMENTS

Objective: Improve the quality of imputed trajectories.
Main Idea: The main idea of this phase is to segment imputed trajectories into sub-trajectories and re-align the imputed GPS samples, so they produce more natural shapes.
Algorithm: During the previous phase (i.e., trajectory imputation), we occasionally face a zig-zag problem in which the imputed samples are not correctly aligned. While this minor problem has very insignificant impact on several applications built on top of trajectory data, such as map inference, trajectory clustering, and trajectory similarity. Our goal is for the imputed trajectories to mimic realistic scenarios. Figure (9), shows the two refinement cases we would like to fix. Figure 9 (a), shows a case where imputed GPS samples (in green) are within a reasonably small distance from the linear line that joins the two consecutive samples in the sparse trajectory (dotted-red line). Ideally, we should proceed to moving all imputed GPS samples towards the dotted-red line. This is particularly relevant when segments of the imputed trajectory lay on highways. However, there are more complicated cases, as shown in Figure 9 (b), where we need to pay attention to different sub-trajectories in order to perform the most optimal refinement. As indicated in Algorithm (2), we adapt the concept of a Sliding Window to find the best representation of multiple best-fit lines. If there are only two samples, then we just connect them with a straight line. Otherwise, we start with the first three samples and try to find the line of best-fit among them, solving a linear matrix equation and getting the least-squares solution to it. If the residuals of such an equation are

Algorithm 2 Trajectory Refinement

```
Procedure: Refine(Γ_i, ϵ)
anchor ← 0
window_size ← 3
refined ← {}
prev_refined_segment ← Γ_i[0 : 2]
for i ← 3 to length(Γ_i) do
    segment ← Γ_i[anchor : anchor + window_size]
    refined_segment, residual ← best_fit(segment)
    if residual > ϵ then
        refined ← refined ∪ prev_refined_segment
        anchor ← anchor + window_size
        window_size ← 3
        prev_refined_segment ← Γ_i[anchor : anchor + 2]
    else
        prev_refined_segment ← refined_segment
        window_size ← window_size + 1
    end if
end for
return refined
```


(a) Straight-line refinement



(b) Best-line fit refinement

Figure 9: Trajectory refinement use cases.

below a small enough threshold ϵ; then we know that these three samples can be represented by a straight line, so we try to add one more sample to this buffer. Otherwise, this will be the *breaking sample*. We map the previous samples to their best-fit representation, and we continue building the next window. An example of such a buffer of samples and the breaking sample is shown in Figure 9 (b). We repeat this process, until we cover all the imputed samples. Proceeding this way ensures that a large trajectory is segmented into smaller chunks that can be relatively approximated with straight lines, mimicking real-world moving patterns.

7 EXPERIMENTS

In this section, we extensively evaluate the accuracy and efficiency of GTI framework[1] in imputing real sparse trajectories in the absence of the road network "map". We compare the performance of GTI with existing methods including linear interpolation, knowledge-based trajectory completion framework [22], and network-less trajectory imputation [14], which is the state-of-the-art work. In this section, we will refer to [22] as *Knowledge Based* and [14] as *TrImpute*. Also, we assess our results by comparing the imputed trajectories to the OpenStreetMap (OSM) [1]. Furthermore, we demonstrate the applicability of GTI framework to serve as a preprocessing step on four different trajectory-based applications: map inference, estimated arrival time, trajectory similarity and search engine.

[1]The GitHub repository of our GTI framework can be found at: https://github.com/qcri/QCAI-TransportaionGroup-GTI.

7.1 Real Trajectory Datasets

We utilize three real datasets from *UIC, Chicago, NYC, New York City* and *Doha, Qatar* for spatio-temporal imputation. The UIC, Chicago dataset has ~900 trajectories, which covers an area of approximately 2km×3km in downtown of Chicago. The total trajectory length is ~3,000Km and density 13K samples per Km^2. It is generated by a fleet of 13 University buses for approximately one month with relatively regular routes. The NYC, New York City dataset has 1.4 million trajectories, which cover an area of 14km×4km in Manhattan and released by the New York City Taxi and Limousine Commission [28]. The total trajectory length is 6.8×10^6 Km and density 50K samples per Km^2. Since this dataset is sparse and there is no ground truth of it, we exploit it only for estimated arrival time experiments. The Doha dataset contains ~200K trajectories, which covers almost the whole city of Doha with total trajectory length 750,000 Km and density ~15K samples per Km^2. It is generated by a fleet of ~4K taxis. We processed all datasets to have four attributes for each trajectory: trajectory_ID, latitude, longitude and timestamp. In addition, GTI estimates the bearing angle (i.e., heading of the moving direction) and speed.

In order to facilitate GTI framework using these datasets, we perform the following processing steps. First, we partition all trajectories based on trajectory_ID. Next, for each trajectory, we split it if we find the sampling interval exceeds 300 *seconds*, which represents either a long stop, or another trip by the same vehicle. Then, we retain all trajectories that have more than two GPS samples. Finally, we artificially sparsify these trajectories based on the dataset size. 1) For small areas like Chicago, we sparsify trajectories every 250 meters. 2) For city-wide scale like Doha: we sparsify trajectories every 1000 meters. These prepared datasets are used as input to all imputation frameworks. Since the NYC dataset is already sparsified at around 2.5 kilometers, we used it as it is.

7.2 Experimental Evaluation

We extensively evaluate GTI using the following metrics:
1) **Accuracy:** The average Euclidean distance between imputed trajectory segments to its corresponding dense trajectory segments. 2) **Completion Rate:** The ratio of imputed trajectory segments to all segments. 3) **OSM Accuracy:** Measures the accuracy, using Euclidean distance with 50 meters threshold, of imputed segments compared to the OpenStreetMap (OSM) [1]. 4) **Trajectory Similarity Accuracy:** We used four metrics, two of them as complete matches (Dynamic Time Warping (DTW) [5] and Euclidean Distance) and two as partial matches (Edit Distance with Real Penalty (ERP) [11] and Frechet Distance (FRECH) [4]). 5) **Range Query Accuracy:** It counts the number of vehicles passing through a junction in the sparse, or imputed trajectory compared to the dense trajectory. 6) **Map Inference Accuracy:** It utilizes the holes and marbles method for measuring the accuracy of the inferred map [7]. The proposed procedure relies on two metrics: a. *GEO metric:* it evaluates how well the inferred map geometrically matches the ground truth map. In our work, we treat the raw "dense" dataset as a ground truth map. First, we take sample points every 5 meters from both maps. Then, we put a hole on each sampled point of

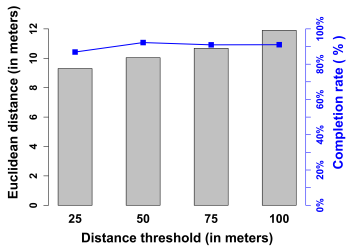


Figure 10: Effect of the distance threshold on Doha dataset.

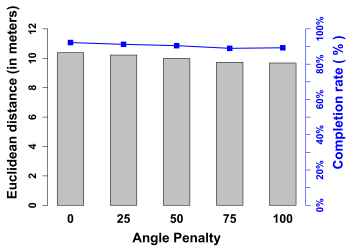


Figure 11: Effect of the angle penalty on Doha dataset.

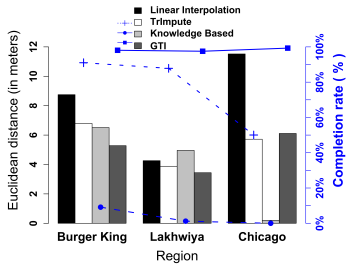


Figure 12: Comparison of imputation algorithms in different regions.

the ground truth map. Similarly, we put a marble on each sampled point of the inferred map. After that, we say a marble is matched if there is a hole within a matching tolerance. In our case, we vary the matching tolerance in a range between 10 and 50 meters with an increasing step size 10 meters. Finally, we compute the precision, recall and F_{score} as directed in [7].

b. *TOPO metric:* it assesses the quality of the topological characteristics of the inferred map compared with the ground truth map through measuring the connectivity structure of the inferred roads. The TOPO procedure starts by selecting a random marble (or hole). Then, finds all reachable marbles (or holes) from the starting marble that are within a radius distance. For the neighborhood of each selected marble, we calculate the precision, recall, and F_{score}, as above. We repeat this process n times, and report the mean values.

7.3 GTI Sensitivity Analysis

Here, we study the impact of varying GTI main two parameters d and θ on the accuracy and completion rate. The impact of both parameters is studied on small as well as on large regions as follows:

- **Distance threshold (d):** Figure (10), depicts the impact of varying the distance threshold (d) from 25 meters to 100 meters on both the accuracy and completion rate. As the distance threshold increases, the accuracy decreases because GTI creates an edge between two GPS samples that belong to different roads. However, with small distance threshold, the completion rate decreases because the graph has less number of edges. Therefore, we pick the 50 meters as the best threshold value and use it for all GTI experiments.
- **Angle penalty (θ):** Figure (11), depicts the impact of varying the angle penalty (θ) from 0 to 100. A small value of θ would reduce the accuracy because it will create an edge between two vertices that have large angle difference (i.e., the two vehicles are moving in the opposite directions). Using a large value of θ would increase the accuracy, as it will create an edge between vertices that share similar direction. In other words, this means that GTI is penalizing significantly the vehicles that are moving in opposite directions. Contrary, the completion rate decreases as the angle penalty increases because the graph has less number of edges. We choose 50 as the default value for the angle penalty threshold.

7.4 GTI Spatial Evaluation

We perform two different levels of evaluations based on region size:

- **Junction-wise scale:** Figure (12), shows that GTI is more accurate, in terms of both accuracy and completion rate, than all baselines for Burger King roundabout and Lakhwiya intersection. However, for Chicago downtown dataset, TrImpute is more accurate (5.72m).

Algorithm	accuracy	completion rate
Linear Interpolation	43.44 m	N/A
TrImpute	13.88 m	85.68 %
Knowledge Based	N/A	N/A
GTI	**10.04 m**	**92.11 %**

Table 1: Comparing imputation algorithms for Doha city.

Yet, it completes only 50.04% from input trajectories, whereas GTI completes almost all input trajectories (99.30%). Knowledge Based fails to give an imputation on the Chicago dataset.

- **City-wide scale:** Table (1), depicts the superiority of GTI framework across all different imputation algorithms. Imputing sparse trajectories using GTI (~ 10 meters) improves the accuracy by more than 4 times compared to the linear interpolation method (~ 43 meters). Also, it improves the imputation by almost 40% compared with TrImpute framework (~ 14 meters), while knowledge-based algorithm is not scalable enough to run on large regions. For completion rate, GTI is the best, where it imputes more than 92% of input trajectories followed by TrImpute, which imputes ~86%. Knowledge Based does not give any imputation for the city of Doha.

7.5 Impact of Input Data Size

We evaluate the optimal size and density of input trajectory data:

- **Number of input trajectories:** The approach proceeds by choosing a random sample from input trajectories, ranging from 1,000 to ~200k trajectories. Since the chosen trajectories still cover the same area, this approach mostly impacts the density of the samples per km^2. Figure (13), depicts the robustness of GTI, where inspite of the number of input trajectories, it is still very accurate (~10 meters). Except, when the number of input trajectories are very small (less than 1%), then, the linear interpolation is the most accurate one, followed by GTI and TrImpute in the third place. As per the completion rate, with the number of input trajectories increasing, GTI imputes more trajectories and it reaches the best value (~92%), when we use the whole Doha dataset.
- **Sparsification rate:** Instead of choosing trajectories randomly, in this experiment we use the entire set of trajectories, but we change the number of representative samples for each one. Thus, we extract a sample every 750, 1000, 1500 and 2000 meters respectively from each trajectory for each of the reported results in Figure (14). For accuracy, GTI being constantly 3-4 times better than linear interpolation and 4-6 meters closer to the raw "dense" dataset. TrImpute resides in the middle and more closer to GTI. In terms of completion rate, GTI has the best imputation for the 1000 meters sparsification. The completion rate only falls short to 85% on a 2000 meters

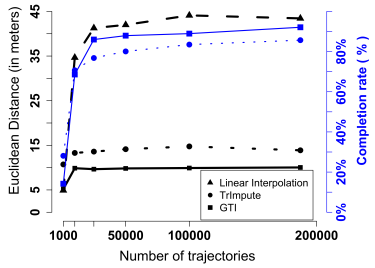


Figure 13: Euclidean distance and completion rate when we change the number of input trajectories

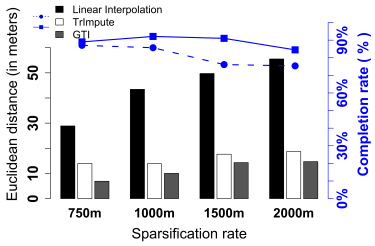


Figure 14: The impact of sparsification length on the average Euclidean distance and completion rate.

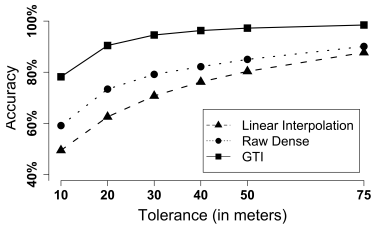


Figure 15: Matching accuracy of Sparse, Raw "Dense" and GTI compared to Doha road network from OpenStreetMap.

d (in meters)	0	2.5	5	10	25
# nodes	535 k	93 k	53 k	25 k	9 k
# edges	250 mill	7.7 mill	2 mill	300 k	25 k
accuracy (in m)	10.04	9.90	10.25	10.67	12.80
completion rate	92.11%	85.72%	84.96%	84.79%	74.81%
time/traj (sec)	3.13	0.19	0.084	0.059	0.035

Table 2: Effect of the cluster radius for Doha dataset.

sparsification, and it is more than 89% on the rest. TrImpute on the other hand, reaches ~87% on a 750 meters sparsification rate and gradually falls to ~ 75% on 2000 meters.

7.6 Graph Generation Optimization

In this section, we showcase the importance of the cluster radius parameter in the computational time of GTI. Table (2), depicts the impact of varying the cluster radius (r) from 0 to 25 meters for the Doha dataset. Initially, the dataset is represented by more than half a million vertices and ~250 million edges. However, by slowly increasing the cluster radius to 25 meters, we see a drop of around 60 times for the vertices and around 10,000 for the number of edges. Also, the imputation speed becomes faster by ~90 times. The accuracy and the completion rate are both be reversely associated with the cluster radius, impacting them negatively when the radius increases. However, we notice that when the cluster radius r is 2.5 meters, we achieve the best accuracy at 9.90 meters, and the completion rate is the second highest at 85.72%. This is still more than 16 times faster than working with the initial dataset, having minor impact on accuracy and completion rate.

7.7 OSM Accuracy

We evaluate the performance of GTI with respect to the ground truth road network extracted from OpenStreetMap (OSM). We then compare it to both linear interpolation and the raw "dense" dataset. Our goal is to confirm that the imputed trajectories are within the actual road network and that GTI makes meaningful imputations. Figure (15) reports the results of this experiment, which is set up as follows: Given one of the datasets, we try to map each of their GPS samples to the closest road from the OSM road-network. If such a road exists within a defined tolerance threshold (ranging from 10 to 75 meters). Then, we consider the GPS sample as matched. Otherwise, we label that sample as a non-match. After gathering the results, we report the OSM accuracy, which is the number of matched samples divided by the total number of samples. Note that in Figure (15) GTI has the highest accuracy, even higher than the raw "dense" dataset. This occurs because of the following reasons: 1) GTI imputation is denser than the raw dataset. 2) All the samples of GTI are part from the raw dataset. 3) GTI gives preference to the most visited paths that reduces noise. Hence, GTI attains higher accuracy. Starting from a 20 meters tolerance, GTI achieves an accuracy of 90% or higher. At a threshold of 10 meters, GTI is more than 1.6 times better than linear interpolation. However, the latter catches up to performing 1.12 times worse at 75 meters tolerance. It is important to note that 75 meters is sufficient to cover the width of a road with around 20 lanes (3.3 meters per lane).

7.8 GTI Temporal Evaluation

Here, we evaluate the temporal aspect of GTI, as illustrated in Figure (16). We performed time of arrival tests on four services, Google Maps [17], OSRM [25], GraphHopper [16] and GTI, using road segments from both Doha and NYC datasets. Because the others are commercial services, we perform the analysis on the same 20,000 randomly chosen road segments for all services including GTI. Then, to support and validate the partial results, we perform the same analysis for GTI on the entire datasets for both Doha and NYC. While Google Maps and GraphHopper are both traffic-aware, OSRM primarily relies on segment lengths and the maximum speed of the queried road segments to make predictions. To filter inaccurate time predictions, we eliminate predictions that took longer than 10 minutes in Doha and 25 minutes in NYC, and those that have a difference of more than 5 minutes with the ground truth in Doha and 12.5 minutes in NYC. After filtering, we are left with around 18,000 segments, and we report the Mean Absolute Error, Median Absolute Error, Mean Relative Error, and the Median Relative Error.

As depicted in Figure 16 (a), our findings show that GTI has the most accurate time prediction, with a Mean Error of 42 seconds (35%) and Median Error of 22 sec (22%) in the Doha Dataset. On the same dataset, Google Maps is the second best, having predictions twice as worse as GTI. GraphHopper has comparable results to Google Maps performing around 10% worse, while OSRM is far off with Mean Error of 142 sec (168%) and Median Error of 128 sec (98%). The NYC dataset, reported in Figure 16 (b), gives a similar output, with GTI giving time predictions as close as 186 seconds (20%) on average to the ground truth and with a Median Error of 81 seconds (15%). Google Maps is again in second place, peaking at an absolute median of 141 seconds (25.63%) away from the raw "dense"

Measure	Distance	Metric	Linear Interpol.	TrImpute	GTI
Complete match	DTW	✗	5287.16	1471.39	**970.40**
	Euclidean	✓	43.44	13.88	**10.04**
Partial match	ERP	✓	5215056	137934	**55355**
	FRECH	✓	122.34	46.18	**34.28**

Table 3: Trajectory similarity metrics on Doha dataset.

Junction Type	#Trajectories	Linear Interpolation	TrImpute	GTI
Sidra	24348	10383	17939	**21305**
Gharrafa	520	300	207	**341**
Samari	277	33	83	**98**
Elite	1113	78	**415**	401
Souq Ali	11594	9608	9385	**10629**

Table 4: Range query on different junction types in Doha.

dataset. GraphHopper had a lot of errors in segment duration in the NYC dataset, leaving the third place for absolute errors to OSRM, while both of them maintain a Mean and Median Error of ~ 50%.

7.9 Trajectory Similarity Application

In this section, we evaluate the similarity of GTI imputed trajectories to raw "dense" trajectories. GTI shows significant improvements when compared to both linear interpolation and TrImpute. From Table (3), it is obvious that GTI outperforms both linear interpolation and TrImpute in terms of these measures in the city-wide experiment. In the ERP context, GTI is around 100 times better than linear interpolation and provides results 2.5 times more similar to ground truth than TrImpute, while for the rest of the measures, GTI is between 3.5 - 15 times better than linear interpolation and 1.3 - 1.5 times better than TrImpute. In the city-wide experiment, the input segments are sparse (1 *kilometer*); therefore the room for error for both linear interpolation and TrImpute is large.

7.10 Trajectory Search Engine Application

Range Query Accuracy: In this experiment, we evaluate the accuracy of GTI on raw "dense" trajectories within junctions of three different types and compare them to linear interpolation and TrImpute results. We identify junctions from the Doha dataset of the following types: roundabouts (Sidra and Gharrafa), T-intersections (Samari and Elite), and interchanges (Souq Ali). To set up the experiment, we first define a rectangular shape bounding box that covers the area of interest from each of the mentioned junctions. Then, we proceed to counting the number of trajectories that have sample points that pass through the rectangle.

Table (4), shows that GTI allows to increase the overall accuracy of range queries that are of a capital importance for many transportation studies, e.g., counts, flux analysis, etc. We notice that in areas that are very dense, such as Sidra and Souq Ali, GTI is able to deduce that the trajectory imputations should go through them. On the other hand, junctions such as Samari and Elite, because the amount of trajectories that pass by them are relatively small, GTI finds it harder to make the imputation through them, even though it is around 3-4 times more accurate than linear interpolation. The only junction where GTI is not the most accurate is Elite, where TrImpute performs better by a small margin, while linear interpolation is able to capture only 7% of those trajectories.

7.11 Map Inference Application

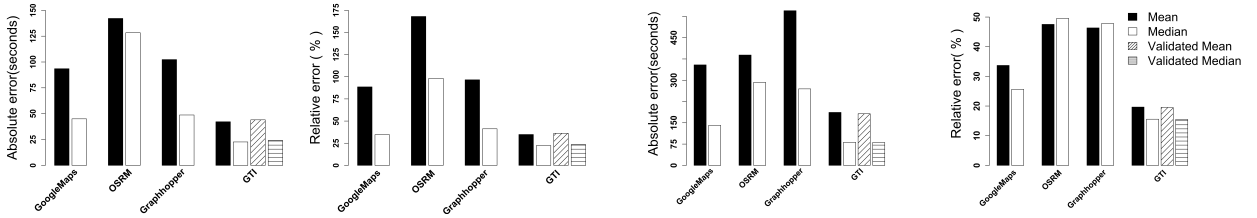
In this experiment, we evaluate the impact of imputing sparse trajectories using GTI on the quality of a map inference application, Kharita [33]. We run this experiment on the Chicago dataset with sparse trajectories, raw "dense" trajectories and then with the imputed trajectories from TrImpute and GTI. Figure (17), quantifies the impact of using data imputation algorithms as a preprocessing step for the map inference application. We use Holes & Marbles to perform such a comparison. We report the F_{score} of the matching between Kharita+Sparse, Kharita+TrImpute and Kharita+GTI compared to Kharita+Raw "dense". As supported with numerical evidence in Figure (17), sparse trajectories are incapable to infer the road network. They only get a 20% F_{score} with a 50 meter tolerance, where the recall is very low, because most of the road segments are not captured. On the other hand, TrImpute starts with a 44% F_{score}, and then it maintains a difference of 6-7% below GTI. As mentioned before, TrImpute has a low completion rate in Chicago dataset, which shows again the importance of high imputation. GTI on the other hand, provides a result that is the closest to the Kharita+Raw map, reaching an F_{score} of 81% with 50 meters tolerance.

8 RELATED WORK

A trajectory interpolation application is tremendously studied under various names such as: trajectory imputation [10, 14], trajectory completion [22], trajectory data cleaning [38], and trajectory interpolation [23], with the objective of filling gaps between consecutive GPS samples in a trajectory. Various techniques have been proposed such as linear, cubic spline, random walk, Bézier curve, and kinematic path. However, a specific interpolation technique could be beneficial in one domain and worthless in others. For example, random walk interpolation [35] is useful for free movement objects such as interpolating wild animal behavior. Chen et al. [10], proposes a probabilistic framework that imputes raw trajectories with a focus on enriching the semantic meaning of raw trajectories in real time. The main issue with this study [10] is that it needs an updated road network available. Recently, various deep learning architectures [31, 32] were exploited to reconstruct trajectories. However, these architectures assume that the provided trajectories are *dense*. Li et. al. [22] is one study that doesn't assume the underlying road network is available. However, it suffers from few shortcomings such as it requires spatially dense trajectories to precompute the junction network, and it is not scalable enough to work on large regions (city scale). Elshrif et. al. [14] is another study that doesn't assume the underlying road network is available. The work proposes TrImpute, a state-of-the-art approach for trajectory imputation, which takes the sparse GPS data and exploits the *crowd wisdom* idea to guide its imputation process by placing artificial samples in between the sparse ones. The main drawback of this work is the scalability issue which impacts its performance.

9 CONCLUSION

This paper presented GTI; a novel framework for trajectory imputation that has the ability to impute sparse trajectory data, *without* knowledge of the underlying road network. Hence, it depends on the nearby trajectory samples to perform its imputation process. Fundamentally, it creates a directed connected graph that GTI uses



(a) Doha Dataset (b) NYC dataset

Figure 16: Time of Arrival Errors from four services (Google Maps, OSRM, GraphHopper and GTI).

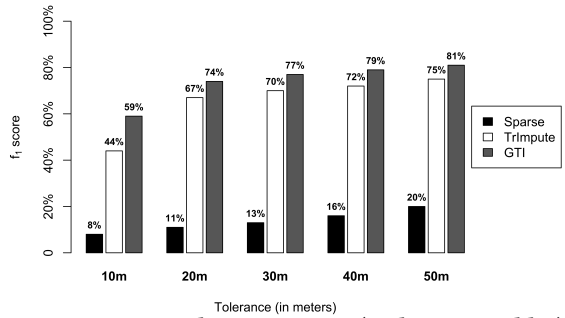


Figure 17: Numerical comparison (Holes & Marbles) between Sparse, TrImpute and GTI when using the Kharita algorithm [33] on Chicago Dataset.

as a *Right-of-Way* (RoW) to find a plausible path that connects two consecutive trajectory GPS samples. Then, GTI formalizes its spatial imputation process for each trajectory segment as to find the shortest path between every two consecutive samples. GTI follows up by doing temporal imputation, where the spatially imputed samples are annotated by timestamp information that respect the traffic conditions of the trajectory segment end samples. Extensive experimental results based on real data and deployment show that GTI is highly accurate and scalable. In addition, GTI significantly boost the performance of four trajectory-based applications.

REFERENCES

[1] OpenStreetMap Public GPS Traces. In *Online*. 2016.
[2] https://www.cs.uic.edu/bin/view/Bits/Software, 2023.
[3] S. Abbar, R. Stanojevic, M. Musleh, M. ElShrif, and M. Mokbel. A demonstration of qarta: An ml-based system for accurate map services. *Proc. VLDB Endow.*, 14(12):2723–2726, jul 2021.
[4] H. Alt and M. Godau. Computing the fréchet distance between two polygonal curves. *International Journal of Computational Geometry and Applications*, 05(01–02):75–91, Mar. 1995.
[5] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, AAAIWS'94, page 359–370. AAAI Press, 1994.
[6] P. Bhatt, S. Gupta, P. Singh, and P. Dhiman. Accident and road quality assessment using android google maps API. pages 1061–1064, May 2017.
[7] J. Biagioni and J. Eriksson. Inferring road maps from global positioning system traces: Survey and comparative evaluation. *Transportation Research Record*, 2291(1):61–71, 2012.
[8] L. Cao and J. Krumm. From GPS traces to a routable road map. In *ACM SIGSPATIAL GIS*, pages 3–12, 2009.
[9] P. Chao, W. Hua, and X. Zhou. Trajectories know where map is wrong: an iterative framework map-trajectory co-optimisation. *WWW*, pages 1–27, 2019.
[10] C. Chen, S. Jiao, S. Zhang, W. Liu, L. Feng, and Y. Wang. TripImputor: Real-time imputing taxi trip purpose leveraging multi-sourced urban data. *IEEE Transactions on Intelligent Transportation Systems*, 19(10):3292–3304, 2018.
[11] L. Chen and R. Ng. On the marriage of lp-norms and edit distance. In *VLDB - Volume 30*, VLDB '04, page 792–803. VLDB Endowment, 2004.
[12] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. math.*, 1(1):269–271, 1959.
[13] X. Ding, L. Chen, Y. Gao, C. S. Jensen, and H. Bao. UlTraMan: A unified platform for big trajectory data management and analytics. *The International Journal on Very Large Data Bases*, 11(7):787–799, 2018.
[14] M. Elshrif, K. Isufaj, and M. Mokbel. Network-less trajectory imputation. In *ACM SIGSPATIAL GIS*, SIGSPATIAL, page 1–10, 2022.
[15] D. Eppstein and M. T. Goodrich. Studying (non-planar) road networks through an algorithmic lens. In *ACM SIGSPATIAL GIS*, pages 1–10, 2008.
[16] G. GmbH. Graphhopper directions api. https://www.graphhopper.com/products/directions-api/, 2023.
[17] Google. Google maps platform apis. https://developers.google.com/maps, 2023.
[18] S. He, F. Bastani, S. Abbar, M. Alizadeh, H. Balakrishnan, S. Chawla, and S. Madden. RoadRunner: improving the precision of road network inference from GPS trajectories. In *ACM SIGSPATIAL GIS*, pages 3–12, 2018.
[19] D. Hopper. 7 times google maps straight up ruined people's lives. In *Online*. 2018.
[20] J. Hu, B. Yang, C. Guo, and C. S. Jensen. Risk-aware path selection with time-varying, uncertain travel costs: a time series approach. *The International Journal on Very Large Data Bases*, 27(2):179–200, 2018.
[21] S. Karagiorgou, D. Pfoser, and D. Skoutas. Segmentation-based road network construction. In *ACM SIGSPATIAL GIS*, pages 460–463, 2013.
[22] Y. Li, Y. Li, D. Gunopulos, and L. Guibas. Knowledge-based trajectory completion from sparse GPS samples. In *ACM SIGSPATIAL GIS*, pages 1–10, 2016.
[23] J. A. Long. Kinematic interpolation of movement data. *International Journal of Geographical Information Science*, 30(5):854–868, 2016.
[24] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang. Map-matching for low-sampling-rate GPS trajectories. In *ACM SIGSPATIAL GIS*, pages 352–361, 2009.
[25] D. Luxen. Open source routing machine. http://project-osrm.org/, 2011.
[26] S. Maneewongvatana and D. M. Mount. Analysis of approximate nearest neighbor searching with clustered point sets. 2002.
[27] M. Musleh, S. Abbar, R. Stanojevic, and M. Mokbel. Qarta: An ml-based system for accurate map services. *Proc. VLDB Endow.*, 14(11):2273–2282, jul 2021.
[28] New York City Taxi and Limousine Commission. TLC Trip Record Data, 2016, 2016. Accessed: March 27, 2023.
[29] T. M. News. Dramatic big rig fatality highlights hazards of Bay Bridge S-curve. In *Online*. September 2009.
[30] S. A. Pedersen, B. Yang, and C. S. Jensen. Fast stochastic routing under time-varying uncertainty. *JVLDB*, pages 1–21, 2019.
[31] N. G. Polson and V. O. Sokolov. Deep learning for short-term traffic flow prediction. *Transportation Research Part C: Emerging Technologies*, 79:1–17, 2017.
[32] S. Ruan, C. Long, J. Bao, C. Li, Z. Yu, R. Li, Y. Liang, T. He, and Y. Zheng. Learning to Generate Maps from Trajectories. 2020.
[33] R. Stanojevic, S. Abbar, S. Thirumuruganathan, S. Chawla, F. Filali, and A. Aleimat. Robust road map inference through network alignment of trajectories. In *ICDM*, pages 135–143, 2018.
[34] H. Su, K. Zheng, J. Huang, H. Wang, and X. Zhou. Calibrating trajectory data for spatio-temporal similarity analysis. *The International Journal on Very Large Data Bases*, 24(1):93–116, 2015.
[35] G. Technitis, W. Othman, K. Safi, and R. Weibel. From a to b, randomly: a point-to-point random trajectory generator for animal movement. *International Journal of Geographical Information Science*, 29(6):912–934, 2015.
[36] Y. Wang, X. Liu, H. Wei, G. Forman, C. Chen, and Y. Zhu. Crowdatlas: Self-updating maps for cloud and personal use. In *MSAS*, pages 27–40, 2013.
[37] Y. Yang, J. Cai, H. Yang, J. Zhang, and X. Zhao. TAD: A trajectory clustering algorithm based on spatial-temporal density analysis. *Expert Systems with Applications*, 139:1128–1146, 2020.
[38] A. Zhang, S. Song, J. Wang, and P. S. Yu. Time series data cleaning: From anomaly detection to anomaly repairing. *VLDB*, 10(10):1046–1057, 2017.
[39] Y. Zheng. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(3):1–41, 2015.
[40] Y. Zheng, L. Capra, O. Wolfson, and H. Yang. Urban computing: concepts, methodologies, and applications. *TIST*, 5(3):1–55, 2014.