Knowledge-Based Trajectory Completion from Sparse GPS Samples

Yang Li[1], Yangyan Li[1], Dimitrios Gunopulos[2], and Leonidas Guibas[1]
[1]Stanford University, [2]University of Athens
{yangli1,yangyan,guibas}@stanford.edu, dg@di.uoa.gr

ABSTRACT

Traffic trajectories collected from GPS-enabled mobile devices or vehicles are widely used in urban planning, traffic management, and location based services. Their performance often relies on having dense trajectories. However, due to the power and bandwidth limitation on these devices, collecting dense trajectory is too costly on a large scale. We show that by exploiting structural regularity in large trajectory data, the complete geometry of trajectories can be inferred from sparse GPS samples without information about the underlying road network — a process called *trajectory completion*. In this paper, we present a knowledge-based approach for completing traffic trajectories. Our method extracts a network of road junctions and estimates traffic flows across junctions. GPS samples within each flow cluster are then used to achieve fine-level completion of individual trajectories. Finally, we demonstrate that our method is effective for trajectory completion on both synthesized and real traffic trajectories. On average 72.7% of real trajectories with sampling rate of 60 seconds/sample are completed without map information. Comparing to map matching, over 89% of points on completed trajectories are within 15 meters from the map matched path.

CCS Concepts

•Mathematics of computing → Interpolation ; • Computing methodologies → Spatial and physical reasoning;

Keywords

Trajectory completion, junction network, trajectory skeleton

1. INTRODUCTION

Data completion, the task of recovering missing data from limited sparse data, is a fundamental problem for most research fields. Large traffic trajectory data is no exception.

SIGSPATIAL '16 October 31-November 03, 2016, Burlingame, CA, USA.
© 2016 ACM. ISBN 978-1-4503-4589-7/16/1. . . $15.00
DOI: 10.475/123_4

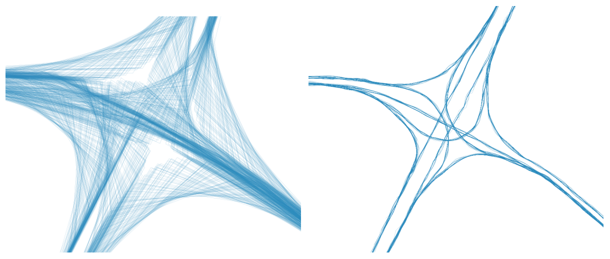


Figure 1: Linear Interpolation (left) vs. Knowledge-Based Trajectory Completion (right) on sparse trajectories at a road junction.

Due to the power cost of GPS-enabled devices and the limited communication bandwidth for sending continuous position data to a centralized location, most trajectory data are stored at low sampling rates. However, applications in urban computing and traffic management typically require dense trajectories with rich spatial and temporal knowledge [24, 25]. Therefore, trajectory completion from sparse GPS samples is an important task to improve the utilization of large trajectory data.

Trajectory completion is a challenging problem. Simple interpolation between neighboring samples is far from a faithful approximation of the trajectory a user actually traversed. For example, in Figure 1 (left), a collection of simulated traffic trajectories across a road junction, sampled every 15 seconds is linearly interpolated. This results in interpolated points being scattered off the roads. One solution to this problem is *map matching*, which aligns GPS samples with the road network on a digital map. However, road network information may not be accurate for many reasons, such as missing roads, constructions, or incomplete information on turn restrictions. For bicycle or pedestrian traffic, the underlying map may not even be available.

In this work we circumvent the limitations of map matching by inferring user paths using trajectory data alone. We rely on two basic assumptions of the input data: (i) trajectories exhibit structural regularity, and (ii) there are enough GPS samples on the same sub-path as the target trajectory to provide detailed geometry information. Traffic trajectories can satisfy both of these assumptions, as they are constrained by roads and are available in large volume.

A key challenge in finding structural regularity of trajectories is to detect *common sub-trajectories* that share the same sub-path on the underlying map, so that GPS samples in these sub-trajectories can be used to complete each other. In places where multiple roads intersect, such as a highway junction with looping ramps, GPS samples from

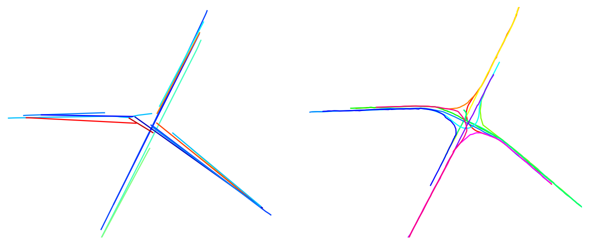


Figure 2: Trajectory clusters (displayed as center curves) using the TraClus Algorithm [13] (Left), and the clusters using our algorithm (Right). The latter preserves the continuity of traffic across the intersection.

different users on the same path might be distant from each other due to the low sampling rate. We are required to use a large search radius, which often includes more noise than information. Hence clustering algorithms that rely on local features, such as DBScan [7] does not perform well in these places. Also, image-based path finding algorithms similar to [15] can produce too many shortcuts at junctions. On the other hand, sample points that are far away from junctions can be much more reliably clustered as their topology and geometry are much simpler. Inspired by this observation, we propose a knowledge-based method that extracts information about road junctions and how traffic flows between them. This allows us to cluster subtrajectories with large intracluster variance at junctions (Figure 2.b).

The main contributions of this paper are as follows

1. *Discover junction and junction branches.* We find junction branches by efficiently extracting a skeleton of the GPS point cloud. This is achieved by an L1-medial skeleton extraction on a voting image of the GPS points. This process essentially decomposes the map into non-junction (reliable) and junction (unreliable) regions.
2. *Junction network construction.* We capture distinct traffic flows at junctions in a *junction network*, where every edge represents a sub-trajectory cluster of the input trajectories. This method is robust since we use shared topology to collect information from reliable regions (non-junction), and propagate such information into unreliable regions (junctions).
3. *Fine-level trajectory completion.* We develop a *borrowed-sample tracking* technique to concatenate dense sub-trajectories into a complete trajectory at fine level. Using this technique, each trajectory will "borrow" samples from other trajectories with a greedy approach.

The rest of this paper is organized as follows. We present the formal definition of trajectory completion and an outline our approach in Section 2. Details of the algorithm are discussed in Sections 3-5. Section 6 shows experimental results on the proposed algorithm, followed by a discussion on related works in Section 7. We conclude the paper in Section 8.

2. OVERVIEW

2.1 Problem definition

A *trajectory* $\tau = \langle (p_1, t_1), ..., (p_{|\tau|}, t_{|\tau|}) \rangle$ is a sequence of time-stamped points in $\mathbb{R}^2$, where $|\tau|$ is the cardinality of trajectory τ. Since this work focuses on completing the geometry of trajectories rather than their dynamics, we do not directly use timing information. More explicitly, we assume the majority of input trajectories are sampled from the same mode of transportation at similar sampling rates. Hence in the rest of this paper, we consider trajectories simply as polygonal curves $\tau = \langle p_1, \ldots, p_{|\tau|} \rangle$ in the plane. In many GPS captured trajectories, we also have access to the vehicle's heading direction at each GPS sample. Let $\hat{d}_1, ..., \hat{d}_n$ be the unit vectors indicating the heading directions along the trajectory (if this information is not directly available, it can be approximated from the positional and timing data).

Next we give a precise definition of a *dense* trajectory.

Definition 1. Trajectory $\tau = \langle p_1, \ldots, p_{|\tau|} \rangle$ is ϵ*-dense* if the distance between any consecutive pair of points p_i and p_{i+1} ($1 \leq i \leq |\tau| - 1$) in τ is at most ϵ meters.

Now we formally define the trajectory completion problem as follows:

Definition 2. **The Trajectory Completion Problem**: Given a collection of sparse GPS trajectories T on an unknown road network and a positive real number ϵ, *trajectory completion* is the process of finding an ϵ-dense trajectory $\tau' = \langle q_1, ..., q_{|\tau'|} \rangle$ for every sparse trajectory $\tau \in T$. Each point $q_i \in \tau'$ belongs to some input trajectory in T that is on the same common sub-path as τ in the underlying road network.

Note that the perfect completion of the entire trajectory dataset T may not be possible if there is any point whose ϵ-neighborhood doesn't include any other points from T. This is often the case on less frequently traveled roads. Therefore our goal is to complete trajectories from major traffic flows, where we assume input point sets are sufficiently large and distributed.

2.2 Algorithm outline

Our trajectory completion algorithm consists of three main steps: (i) Extract a skeleton from the trajectory point cloud, (ii) Construct a junction network from the skeleton and the input trajectories, (iii) Complete input trajectories based on the junction network topology.

Figure 3 illustrates the algorithm on a single-junction scenario. Consider the trajectories in Figure 3.a as part of many sparse trajectories. For clarity, we color sample points based on their traffic flow directions across the junction. During trajectory skeleton extraction, we first apply an L1-medial skeleton extraction algorithm on the filtered point cloud of GPS samples. The resulting skeleton is represented by a graph G_{sk}, whose edges (skeleton branches) are densely traveled roads, and vertices are branch endpoints. In Figure 3.b, 4 branches are extracted from the input point cloud and the branch intersection is marked as a junction candidate.

The next step computes a junction network J, which captures the traffic flow between pairs of skeleton branches. First all trajectory points are projected to nearby skeleton branches. Then we identify traffic flow clusters via a voting process on pairs of branches. The junction network in Figure 3.c contains 8 nodes, each representing a half-edge of G_{sk}. It has 3 edges representing the 3 unique traffic flows in the input. The weight of an edge (u, v) is the number of trajectories with consecutive projections on u and v.

Finally, input trajectories are completed using trajectory points within the same traffic flow cluster. This step constructs a constrained nearest neighborhood graph within

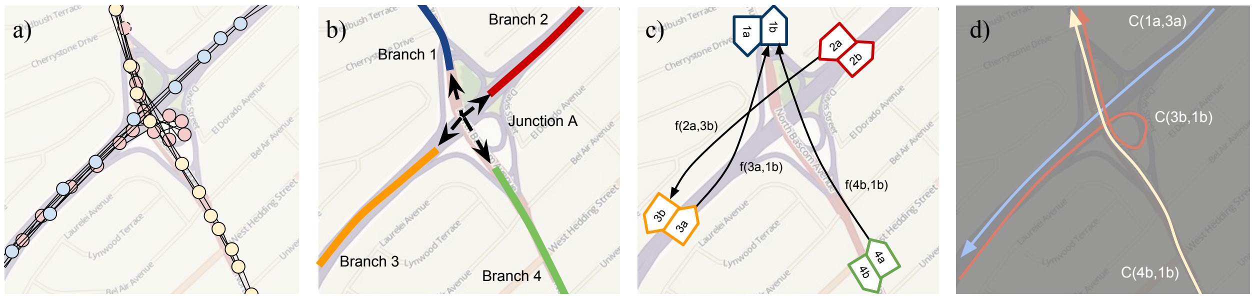


Figure 3: Overview of the trajectory completion pipeline. Details can be found in Section 2.2

each cluster, then fills in missing points between consecutive samples using shortest paths in the graph. Figure 2.d illustrates three of the completed trajectories in the example. When a trajectory is mapped to more than one sub-trajectory clusters, we concatenate the recovered sub-trajectories into full trajectories.

3. SKELETON EXTRACTION

We adapt the L1-medial skeleton extraction algorithm proposed by Huang et. al. [11] for trajectory data. First we will briefly describe the original algorithm for a generic point set. Then we will explain several optimizations to extract the skeleton of GPS trajectories.

3.1 L1-Medial skeleton

Definition 3. The *L1-medial skeleton of a point set* Q is a curve that represents the 1D local center of the underlying shape sampled by Q. Skeleton points $X = \langle x_1, \ldots, x_{|X|} \rangle$ satisfy the local median property:

$$X = \underset{x}{\operatorname{argmin}} \sum_{i \in I} \sum_{j \in J} ||x_i - q_j|| \theta(||x_i - q_j||) + R(x)\,, \qquad (1)$$

where I and J are indices of skeleton points X and input points Q, respectively. $\theta(y) = exp(-y^2/(\eta/2)^2)$ is an exponentially decaying weight function parametrized by positive constant η [11].

Neighborhood size constant η determines how fast the weight decays as distance y grows. The first term of the objective function (1) measures the contraction force on X towards local medians of Q; the second term $R(x)$ regularizes the local distribution of X to preserve the linearity of local branches emerged from the data. Function $R(X)$ is defined as follows:

$$R(X) = \sum_{i \in I} \gamma_i \sum_{i' \in I \setminus \{i\}} \frac{\theta(||x_i - x_{i'}||)}{\sigma_i ||x_i - x_{i'}||}\,.$$

γ_i is a normalizing constant. Variable σ_i measures the local linearity at skeleton point x_i using a PCA approach. Consider the weighted covariance matrix C_i of point x_i's k-nearest neighbors.

$$C_i = \sum_{x_{i'} \in k\mathrm{NN}(x_i) \setminus \{x_i\}} \theta(||x'_i - x_i||)(x'_i - x_i)^T (x'_i - x_i)\,.$$

Let λ_i^0, λ_i^1 be the eigenvectors of C_i. Without loss of generality, we assume $\lambda_i^0 < \lambda_i^1$, and define the linearity measure as $\sigma_i = \lambda_i^1/(\lambda_i^0 + \lambda_i^1)$.

This optimization problem can be solved in an iterative approach similar to k-means clustering. After skeleton point set X converges, branches can be traced from the point $x_i \in X$ with the highest linearity score σ_i. The final skeleton is a collection of polylines or *branches*, and line segments that bridge two or more branches at their endpoints.

3.2 Aggregate trajectory points

We choose the L1-medial skeleton as the abstract representation of a trajectory because of its robustness against outliers and incomplete data. However, due to the large number of trajectories, it is impractical to use all GPS samples as the input point cloud Q. A naive approach would be taking a uniformly random sample from the input points. The drawback of this approach is that sparse GPS samples are not uniformly distributed along a route. When points are clustered at the endpoints of a road segment, random sampling is less likely to recognize the underlying road as a linear feature.

We propose a sampling strategy by aggregating trajectory points into a grayscale image of resolution r using interpolation. Let p_i and p_{i+1} be two consecutive points on a trajectory with heading directions $\hat{d}_i$, $\hat{d}_{i+1}$, we define a fast interpolation scheme as follows :
Case 1. If the angle between $\hat{d}_i$ and $\hat{d}_{i+1}$ is less than $\pi/6$ or greater than $6\pi/7$, linearly interpolate between p_i and p_{i+1}.
Case 2. Otherwise, compute the intersection o_i between the ray originating from p_i in the direction of $\hat{d}_i$ and the ray originating from p_{i+1} in the direction of direction $-\hat{d}_{i+1}$. Then perform an "L-shape" piecewise linear interpolation across p_i, o_i, and p_{i+1}.

For example, given a trajectory $\langle p_1, p_2, p_3 \rangle$ with heading vectors $\langle \hat{d}_1, \hat{d}_2, \hat{d}_3 \rangle$ in Figure 4.a, we interpolate between p_1 and p_2 using Case 2 since the angle difference between $\hat{d}_1$ and $\hat{d}_2$ is $\pi/2$. Then interpolate between p_2 and p_3 using Case 1. Note that the "L-shape" interpolation between p_1 and p_2 is better than a linear interpolation (red dashed line) since the interpolated points are closer to the underlying route. This simple interpolation method can be inaccurate, but it reduces the undesired artifacts of linear interpolation, making them easier to be eliminated in subsequent steps.

Let h and w be the height and width of the bounding box of the interpolated trajectory collection $\overline{T}$. Define the **voting image of** $\overline{T}$, V to be a pixel grid of height $\lfloor h/r \rfloor$ and width $\lfloor w/r \rfloor$. The intensity of the pixel at row i, column j is the total number of trajectories passing through the grid cell of index (i, j). i.e.

$$V_{i,j} = \sum_{\bar{t} \in \overline{T}} \sum_{p \in \bar{t}} 1(i, j, p)\,.$$

Identifier function $1(i, j, t)$ evaluates to 1 if $(i-1)r \le p_x < (i)r$, and $(j-1)r < p_y < (j)r$, and 0 otherwise. Figure 4.b shows the voting image of the junction in Figure 1. Given a

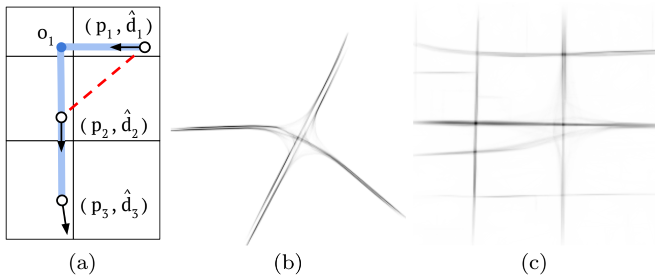


Figure 4: a.) Two interpolation cases between consecutive sample points p_i and p_{i+1}. The red dashed line highlights the difference between piecewise linear interpolation and linear interpolation. b.) The voting image of a 4-way junction. c) A more complex voting image (See TAXI_1 in Section 6.1).

threshold parameter V_{min}, the input point cloud Q consists of pixels in the voting image whose intensity is at least V_{min}. The value of V_{min} is typically a fraction of the maximum intensity of the voting image. e.g. $V_{min} = 0.06 \max_{\substack{0 \le i < w \\ 0 \le j \ne h}} V_{i,j}$.

3.3 Skeleton branch extraction

We initialize X, the set of skeleton points as a uniformly random sample of size N from Q. A challenge in the optimization process mentioned in Section 3.1 is that certain points are slow to converge to the medial skeleton. Such points tend to be far from the underlying road, either due to GPS noise or the artifact of interpolation. We resolve this issue by adding a density weight to each input point in Q during the optimization, such that skeleton points in X will be attracted to locations of higher intensity.

The density of an input point $q_j \in Q$ is approximated by the weighted average of neighbor intensities in the voting image. The contraction term in Equation 1 becomes

$$\sum_{i \in I} \sum_{j \in J} ||x_i - q_j|| \theta(||x_i - q_j||) density(q_j) \,. \qquad (2)$$

Neighborhood size parameter η is set to be larger than the half-width of the road so that noisy skeleton points along the same road can be effectively pulled into a linear configuration. In our problem settings, $\eta = 12$ suffices extracting local roads, while $\eta = 15$ is better for extracting highways.

After skeleton points converge, branches are traced using Algorithm 1. Notice that condition $|b| > 4$ on Line 6 implies a skeleton branch contains at least 4 points. Therefore if the non-junction portion of a road is L meters, r has be be at most $L/4$ in order to represent this road as a junction branch. Therefore we can determine the desirable r based on the street density. The effect of r on trajectory completion is discussed in Section 6.3.

3.4 From branches to skeleton graph

To obtain skeleton branches at multiple scales, we repeat the aforementioned process multiple times with increasing neighborhood constant η. The output is represented by the *skeleton graph* G_{sk}, an undirected graph whose vertices are branch endpoints. G_{sk} contains two types of edges. The first type is a *branch edge*, represented as a polygonal curve. The second type is a *bridge edge*, represented as a line segment that connects two branch edges in G_{sk}. Bridge edges are constructed as follows: At the end of every iteration, we identify *bridge points*, the closest non-branch points in X to either endpoint of each branch and in the same direction as the branch. Bridge points that are close together are then merged into a single bridge point. Finally, we add

Algorithm 1: Extract skeleton branches

```
   Input: Skeleton points $X$
          Linearity score $\sigma_i$ for all $i \in I$
          Principal direction $\gamma_i$ for all $i \in I$
   Output: List of skeleton branches $B = \{b_1, \dots, b_m\}$
   parameter: $thresh_\sigma$ (default value = 0.9 )
1  Define branch candidates $C = \{x_i \in X | \sigma_i > thresh_\sigma\}$
2  Mark all candidates $c \in C$ as unvisited
3  while $C$ contains unvisited candidates do
4      Select branch seed $s \in C$ with largest $\sigma_i$
5      $b, noise$ = TraceBranchFrom($s$, $C$)
6      if $|b| \geq 4$ then
7          add $b$ to $B$
8          mark all points in $b$ as visited
9          remove $b$ and noise from $C$
       else
10         mark $s$ as visited
       end
   end
   Function TraceBranchFrom($s$, $C$)
       parameter: $\alpha$ (default value = 0.95)
1      $branch \leftarrow \{s\}$
2      while True do
3          $x \leftarrow$ nearest point in $C$ such that $x - s \cdot \gamma_j < -\alpha$
           and $||x - s||_2 < r$
4          if $x == NONE$ then
5              Break
           end
6          Append $x$ to $branch$
7          $s \leftarrow$ x
       end
8      return $branch$, $\{p \in C | dist(p, branch) < 2r\}$
```

bridge edges between the merged bridge point and the closer endpoint of every branch affected by the merge. If a bridge point joins a branch at a later iteration, we add a bridge edge between the bridge point to the endpoint of its former branch. A description of the overall skeleton extraction process is given by Algorithm 2.

Algorithm 2: Multi-scale branch extraction

```
   Input: Input points $Q$
   Output: Skeleton graph $G_{sk}$
   parameter: $\eta_0$, $\alpha_0$, $\Delta$, max_num_iterations
1  $\eta \leftarrow \eta_0$; $\alpha \leftarrow \alpha_0$; $i \leftarrow 0$
2  Initialize $G_{sk}$ as an empty graph
3  while $i <$ max_num_iterations do
4      Extraction skeleton branches from $Q$
5      Compute bridge points
6      Add branch edges and bridge edges to $G_{sk}$
7      if No edge has been added in this iteration then
8          $\eta \leftarrow \eta + \Delta\eta_0$
9          $\alpha \leftarrow max(0, \alpha - \Delta\alpha_0)$
       end
10     $i \leftarrow i + 1$
   end
```

A few post-processing steps are performed on the skeleton graph, such as removing degree two vertices, collapsing short edges and finding intersections between the polygonal curves of two branches. If two branches intersect, we split the edges at their intersection.

4. JUNCTION NETWORK CONSTRUCTION

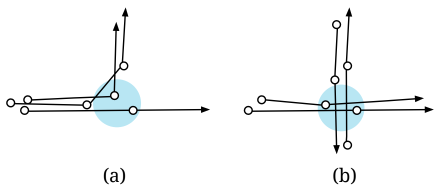


Figure 5: a) The blue region is a trajectory junction since trajectories belonging to the same horizontal road segment split at this region. b) This is not a trajectory junction since neither a split nor a merge is present.

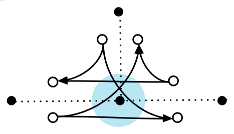


Figure 6: Road network (dotted line) of a T-junction, and its equivalent junction network (solid line)

The skeleton graph summarizes the geometry of the underlying road network utilized by input trajectories, however, it doesn't have accurate topological information needed for trajectory completion. This section introduces the concept of *junction network*, a topological data structure to represent traffic flows across road junctions. We show how to construct the junction network from the skeleton graph, and how it relates to common sub-trajectory clustering.

4.1 Junction network

Unlike a road network, the junction network focuses on the continuity of traffic movement across the unknown road network. To describe such traffic, first introduce a *trajectory junction*, a region where a group of trajectories merge or split (Figure 5.) Since we often study trajectories that span a bounded region (e.g. a bounding box), we designate a special trajectory junction *nil* to represent all locations where trajectories enter or leave the bounded region.

We define the vertices in a junction network as minimal traffic flows:

Definition 4. A *minimum trajectory flow* $f(a,b)$ is a set of sub-trajectories that enters trajectory junction a into trajectory junction b without passing any other trajectory junctions in between. We denote the trajectory indices of sub-trajectories in $f(a,b)$ as $T(a,b)$.

Let the *size of traffic flow* $f(a,b)$ be the cardinality of its trajectory indices, denoted as $|T(a,b)|$. A junction network over minimal traffic flows is defined as follows:

Definition 5. Given a collection of trajectories T and a positive integer M, the *junction network* of T is a directed graph $J=(V,E)$ on all minimal traffic flows. Two vertices $u=f(a,b)$ and $v=f(c,d)$ are connected if 1.) At least M trajectories that pass through both u and v in order. i.e. $|T(a,b)\cap T(c,d)|\geq M$. and 2.) there is no other minimal traffic flow $w=(e,f)$ such that $|T(a,b)\cap T(e,f)|>M$ and $|T(e,f)\cap T(c,d)|>M$. The *flow size*, or weight of edge (u,v) is $|T(a,b)\cap T(c,d)|$.

In the ideal scenario when all possible turns on a road network has been traversed by at least M trajectories, the junction network can be considered the *pseudo-dual graph* of the road network (Figure 6 .) A pseudo-dual graph transformation of a road network is a useful way to represent turn restrictions and variable turn cost in route planning and traffic analysis [20]. For trajectory completion, it is the preferred approach to recover consistent flow across junctions, and infer the correct topology of the road network utilized by the traffic.

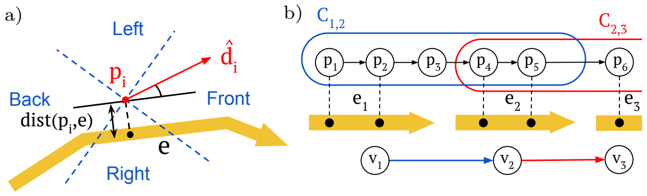


Figure 7: (a) Let γ be the angle between heading vector $\hat{d}_i$ and half-edge e's tangent vector at p_i's projection. $SameDirection(\hat{d}_i,e)=true$ if $\gamma<\pi/4$. $\hat{d}_i$ points to the "front" direction with respect to e. (b) Trajectory $\langle p_1,\ldots,p_6\rangle$ has 5 stable projections to half-edges e_1, e_2 and e_3. They belong to two overlapping clusters $C_{1,2}$ and $C_{2,3}$, which correspond to edges (v_1,v_2) and (v_2,v_3) in the junction network.

The junction network $J=(V,E)$ has several important characteristics.

1. An edge $(f(a,b),f(b,c))\in E$ encodes a *flow cluster* $C_{u,v}=f(a,b)\cap f(b,c)$, which captures the traffic flow from one junction branch to another over a junction. Sub-trajectories within the same flow cluster has much lower geometrical variance than trajectories in different clusters.
2. A path in the junction network represents continuous traffic on a route. Each input trajectory is defined by a sequence of flow clusters. Consecutive clusters in the sequence are partially overlapped.

4.2 From skeleton graph to junction network

Now we discuss how to construct the junction network $J=(V,E)$ from trajectories using skeleton graph G_{sk}. To distinguish traffic flows in opposite directions of a road segment, we first transform each edge (s,t) in undirected graph G_{sk} into two directed half-edges $e=(s,t)$, and $e'=(t,s)$. The junction network can be constructed based on the following definition.

Definition 6. A *Junction network* is a directed graph $J=(E,V)$ where V is the set of all half edges in G_{sk}. Two vertices u and v are connected if (i) the number of trajectories passing directly from u to v is larger than a positive integer M, and (ii) the closest distance between u and v is less than some constant l. The edge weight is the number of trajectories from u to v.

Since G_{sk} is approximated, and the matching between trajectory and routes on the skeleton graph is unknown, we have to estimate the topology of J from trajectories based on the most stable parts of the matching.

4.3 Estimate Junction Network Topology

Initially let J be a complete graph with zero edge weights. First, we find a stable projection for each trajectory $\tau=\langle p_1,...,p_{|\tau||}\rangle$ in trajectory collection T onto the half-edges derived from G_{sk}. For each $p_i\in T$ with heading vector $\hat{d}_i$, let $N_k(p_i)$ be the set of k nearest half-edges from p_i. We define the heading-restricted neighborhood as follows:

$$N_k(p_i,\hat{d}_i)=\Big\{e\in N_k(p_i)\mid SameDirection(\hat{d}_i,e)\Big\}$$

Predicate $SameDirection(u, v)$ makes sure the general heading direction of p_i is consistent with skeleton half-edge e (Figure 7.a). Let $var(D_i)$ be the variance of p_i's projection distances $D_i = \{dist(p_i, e) | e \in N_k(p_i)\}$. We consider p_i's projection on the nearest half-edge as a *stable projection* if

$$1 - \sqrt{var(D_i)}/var_{max} > \beta,$$

where var_{max} is the largest variance among all distance sets. We will discuss the choice of β in Section 6.3. As a result, trajectory τ is mapped to a sequence of half-edges $e_1, \ldots, e_m$ (Figure 7.b .) Then for each consecutive pair (e_j, e_{j+1}) in the sequence such that $e_j \neq e_{j+1}$, we increment the weight of edge (v_j, v_{j+1}) in J. Finally, we retain only edges whose weights are at least M to filter out incorrect projections.

5. TRAJECTORY COMPLETION

With the coarse level junction structure, we infer fine-level trajectory completion using a borrowed-sample tracking scheme. Particularly, the algorithm "borrows" some samples from other trajectories to complete the target sparse trajectory.

Recall that in the junction network, each edge (v_j, v_k) is associated with a group of sub-trajectories with stable projections on both v_j and v_k, denoted as flow cluster. Each input trajectory τ is partially covered by a sequence $C_{j,k}$ of overlapping, pairwise distinct flow clusters (Figure 7.b). Our goal is to fill in the "gap" between every pair of consecutive points p_i, p_{i+1} in the input trajectory using the sequence of flow-clusters associated with this trajectory. This step is outlined in Algorithm 3. Flow clusters in $C_{j,k}$ are re-indexed for clarity.

Algorithm 3: Trajectory completion

```
  Input: Input trajectory $\tau = p_0, \ldots, p_{|\tau|}$
         Flow cluster sequence $C(\tau) = \langle C_1, \ldots, C_m \rangle$
         Starting and ending index of sub-trajectory
         $I_i = (start_i, end_i) \ \forall\ 0 \leq i < |\tau|$
  Output: Completed trajectory $\tau'$ such that $\tau \subset \tau'$
1 Sort $C(\tau)$ by cluster size in descending order
2 foreach $C_i \in C(\tau)$ do
3     for $start_i \leq j < end_i$ do
4         if $gap[p_j, p_{j+1}]$ is not filled then
5             FillGap($p_j, p_{j+1}, C_i$)
          end
      end
  end
6 Concatenate filled gaps into dense trajectory $\tau'$
```

The key procedure in this algorithm is `FillGap`(p_j, p_{j+1}, C_i) on line 5, which finds a polygonal curve $P = \langle x_0 = p_j, x_1, ..., x_{p-1}, x_p = p_{j+1} \rangle$ within flow cluster C_i using a shortest path method. Specifically, we construct a constrained k-nearest neighborhood graph g_k over all trajectory points in C_i. Let $u_1 \in \tau_1$ and $u_2 \in \tau_2$ be two vertices in g_k. Let $\hat{d}_1$ be the heading angle of u_1. g_k contains a directed edge (u_1, u_2) if the following conditions are satisfied:

a) $u1 \neq u2$,

b) u_2 is one of u_1's k-nearest neighbors such that $\frac{u_2 - u_1}{||u_2 - u_1||} \cdot \hat{d}_1 < \alpha$

The second condition implies that the cosine angle between the heading of u_1 and vector $u_2 - u_1$ is less than α, which is introduced in Section 3.4. In practice, we choose k to be 5.

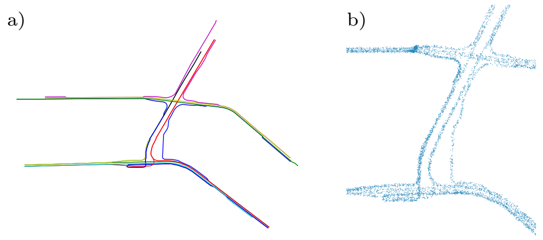


Figure 8: a.) Ground truth routes of benchmark dateset BM_3. b.) Synthesized trajectory points in BM_3.

Since p_i and p_{i+1} are vertices of graph g_k, we can find a shortest path from p_i to p_{i+1}. Then we apply a simple smoothing function, such as linear local regression, on the path to produce the final trajectory.

6. RESULTS

6.1 Experiment setup

To evaluate the performance of our trajectory completion algorithm, we create a collection of benchmark datasets using synthesized trajectories, and several testing datasets using real taxi trajectories.

Synthesized trajectories. The benchmark dataset contains trajectories generated using SUMO, an urban traffic simulation software [5]. Since the challenge of this paper is to recover trajectory geometry at complex junctions, we select 8 junction regions on the map of Beijing, each has size approximately $3km \times 3km$. Using Open Street Map [2] as the road network, we generate a large number of vehicle trips on predefined routes within each region via a 3 hour simulation. For instance, Figure 8.a shows the road network of two adjacent junctions. Each colored line represents a unique route that is used to generate the sparse GPS samples in Figure 8.b. The simulation outputs dense trajectories at 1 sample per second, which is used as the ground truth. We then down-sample the dense trajectories at 15s, 20s, 30s and 60s intervals to create the testing datasets. A random offset of radius $R = 5m$ is added to the down sampled trajectories to emulate the variations of vehicle trajectories and GPS noise. In particular, given a point p on the predefined route, its perturbed position is $p' = p + n + o$, where n is a random vector from Gaussian distribution $\mathcal{N}(0, \frac{R}{2})$, and o is a uniform random vector of length R. The specification of the first three benchmark datasets (BM_1 to BM_3) can be found in Table 1.

Name	# of routes	# of points at sample interval			
		15s	20s	30s	60s
BM_1	12	22,020	16,500	12,386	2,730
BM_2	13	23,529	17,627	11,673	1,317
BM_3	15	30,111	22,608	15063	5,209
Mean trajectory size		11.58	8.68	5.88	3.96

Table 1: Specifications of the first three benchmark datasets. Each route is sampled by 200 trajectories.

Real trajectories. The taxi trajectories used in this work were part of the Beijing Taxi Open Dataset [3], which contains GPS traces of 8602 taxis in Beijing, China in May, 2009. The following preprocessing steps are performed to moderate the large amount of noise present in the raw data. First, we partition all trajectories such that the sampling intervals are between 50s to 60s, which is the default sampling interval of GPS loggers used for data collection. Based on the observation that user trajectories are most irregular at the beginning or ending parts of a trip, we further strip the first and last 30 sample points from each trajectory, and retain those with at least 15 samples. Finally, we select filtered trajectories from 3 regions in Beijing during May 1st to May 5th as the taxi datasets TAXI_1 to TAXI_3. In the following experiments, we always compute the junction network using the entire taxi dataset, but only perform trajectory completion on the first 5000 trajectories in each dataset.

Name	Size (km)	# of trajectories	Mean trajectory size
TAXI_1	3.5×3	32,121	7.19
TAXI_2	6×4	67,501	9.34
TAXI_3	6×4	60,299	10.68

Table 2: Specification of taxi trajectory datasets TAXI_1, TAXI_2 and TAXI_3.

6.2 Evaluation Method

When applying trajectory completion to the test trajectories, we consider the following metrics that evaluate the completeness, accuracy, and impact of parameters.

Projection rate. r_{proj} is defined as the percentage of input trajectories that have been projected to a path on the junction network. Intuitively, it measures the proportion of all input trajectories that can be up-sampled using the trajectory completion algorithm.

Completion rate. $r_{cpl}(\epsilon)$ measures the percentage of ϵ-dense trajectories among all completed trajectories. Let D be the subset of ϵ-dense trajectories in T',

$$D = \left\{\tau' \in T' \mid ||q_i - q_{i+1}||_2 < \epsilon \text{ for all } 1 \le i \le |\tau'|\right\}.$$

The completion rate is $r_{cpl}(\epsilon) = |D|/|T'|$.

Completion accuracy. $\rho(\delta)$ is the percentage of points in the completed trajectories whose error, measured by the shortest Euclidean distances to the ground truth trajectory, is less than δ. It evaluates the overall completion accuracy of a dataset.

6.3 Benchmark Results

Performance. We run the trajectory completion algorithm through all 32 benchmark datasets (8 regions $\times$ 4 sampling profiles) on an Intel Precision T3400 workstation with 4 cores and 16GB RAM. Each test is performed over multiple trials, and the average statistics are used as results. Each trail executes the full pipeline of trajectory completion, which takes approximately 1-2 minutes. The majority of time has been spent on the final trajectory completion step. In future works, this step can be sped up by parallelizing the FillGap operations on multiple trajectories.

Projection rate. Figures 9.a-h show the output ϵ-dense trajectories of all benchmark tests with 20sec sampling interval and $\epsilon = 100m$. In most cases, the junction structures can be clearly identified from the output. Table 6. summarizes projection rate of each benchmark dataset at different sampling profiles. When sampling interval is 30 seconds, 93.8% of all trajectories have been up-sampled using trajectory completion. A few test cases such as BM_2 and BM_8 have lower projection rates than others. This is due to the fact that junction areas in these cases are large comparing to non-junction areas, where stable projections are found.

When sampling interval is 60 seconds, both the number of trajectories in the dataset and the number of samples in an input trajectory decreases. Hence the average projection rate drops to 73.1%. i.e. There are not sufficient samples at non-junction regions to infer which flow cluster a trajectory belong to.

Completion quality. Figure 9.i plots average completion rate $r_{cpl}(\epsilon)$ of benchmark tests at different sampling intervals. For instance, when the sampling interval is 30s, 60.6% of the output trajectories have no gaps longer than 20 meters. Similar to projection rate, completion rate is also lower when sampling interval is 60s, implying that longer gaps are more common.

The average completion accuracy $\rho(\delta)$ of the benchmark trajectories is plotted in solid lines in Figure 9.j. As expected, accuracy is higher when the input trajectories contain more samples. For a baseline comparison, we compute the completion accuracy of cubic spline interpolation and plot the results in dashed lines in the same figure. Let $\delta = 10m$ be the tolerance of error for completion accuracy. When the sampling interval is 30s, trajectory completion has 57% less errors than cubic spline interpolation.

Impact of voting image resolution. During the skeleton extraction step, voting image resolution r influences the level-of-detail of the trajectory skeleton. See Figure 10 as an example. Table 4 compares the projection rate and completion rate of BM_6 when $r = 10, 12.5, 15$ and 17.5 meters. Notice that both metrics are highest when $r = 12.5$, and decreases as r increases. We pick $r = 13$ for the rest of the experiments to balance between quality and performance.

r (meters)	10	12.5	15	17.5
Running time (sec)	180.3	223.0	89.26	66.58
r_{proj}	0.926	0.981	0.833	0.865
$r_{cpl}(20)$	0.861	0.902	0.786	0.793

Table 4: Completion results of BM_6 (30 seconds/sample) for different resolution r.

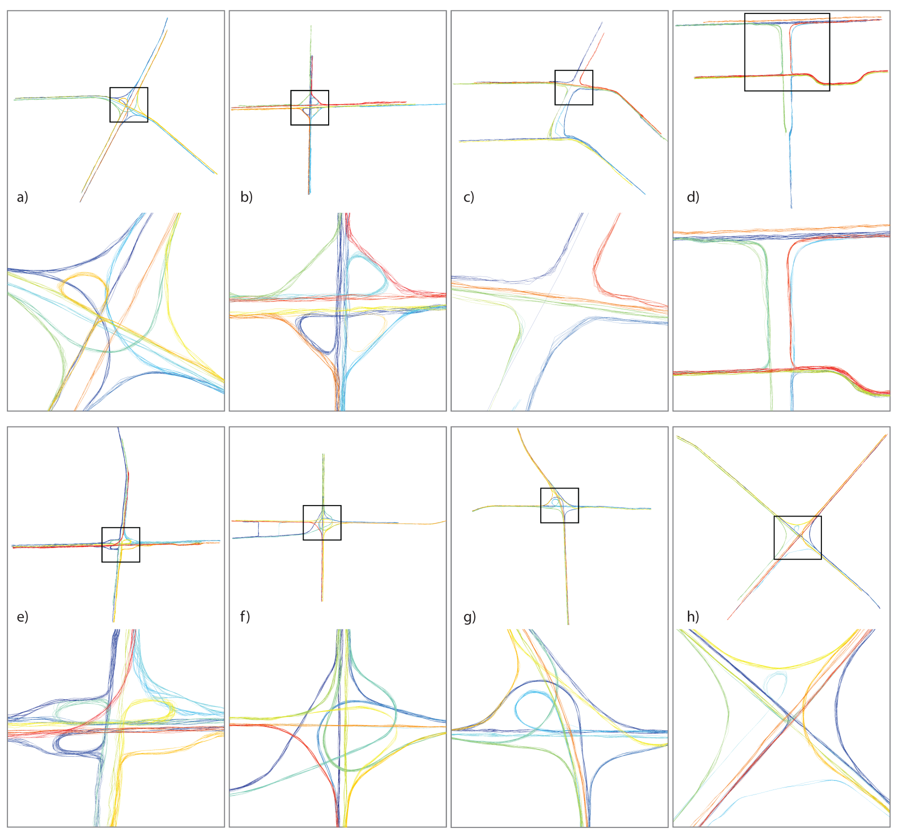
Impact of trajectory projection ratio. Skeleton projection ratio β controls the trade off between the number of flow clusters discovered, and the correctness of trajectory projection. Table 5 shows that both cluster count and projection rate decrease notablly when $\beta > 0.7$. Meanwhile, completion rate increases as r increases (Figure 12).

β	0.5	0.6	0.7	0.8	0.9
# of flow clusters	26	26	23	21	13
r_{proj}	0.999	0.998	0.997	0.995	0.825
$r_{cpl}(20)$	0.423	0.505	0.599	0.692	0.734

Table 5: Completion results of BM_3 (30 seconds/sample) for different projection ratio β.

Input size. To find out how input size affects completion accuracy, we experiment with different input sizes on the same set of routes. Figure 12.a visualizes the completion accuracy of BM_1 using 500, 1000, and 2000 trajectories. We can see that there is a clear difference in completion rate between 500 trajectories and 1000 trajectories. This experiment also demonstrates that our algorithm can effectively leverage the information in individual trajectories. When rare trajectories are discovered, the aggregated information is more than what each trajectory contains, but not by far; as more trajectories are fed into our algorithm, the aggregated information increases, eventually leading the trajectory completion towards the actually map structure.

6.4 Completing taxi trajectories



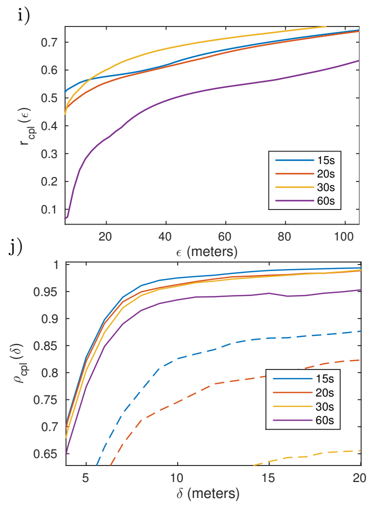


Table 3: Projection Rates of BM_1-8

Benchmark	15 sec	20 sec	30 sec	60 sec
a. BM_1	0.976	0.971	0.970	0.880
b. BM_2	0.933	0.878	0.832	0.508
c. BM_3	0.995	0.982	0.953	0.531
d. BM_5	1.000	0.976	0.925	0.713
e. BM_4	0.986	0.978	0.962	0.650
f. BM_6	1.000	1.000	0.960	0.779
g. BM_7	1.000	1.000	0.999	0.914
h. BM_8	0.858	0.837	0.904	0.874
Average	0.969	0.953	0.938	0.731

Figure 9: Trajectory completion results for the 8 benchmark datasets. Each sub-plot (a)-(h) contains two views of the benchmark trajectory sets BM_1-8. The top view shows the global configuration of the trajectories. The main trajectory junction is outlined in a black bounding box, which is enlarged into the bottom view. Trajectories are colored by route for visual distinction. Sub-plots (i) and (j) show the average completion rate $r_{cpl}(\epsilon)$, and average completion accuracy $\rho(\delta)$ of BM_1-8 at different sampling intervals. The dashed lines in sub-plot (j) shows the completion accuracy of cubic spline interpolation when the sampling intervals are 15s, 20s and 30s.

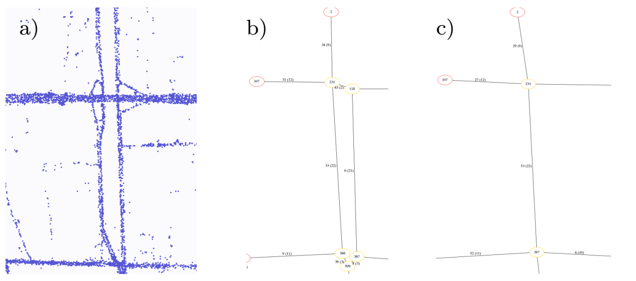


Figure 10: This example shows the effect of resolution r on skeleton graph for two parallel roads from the taxi trajectories, displayed as a point cloud in Figure (a). Figures (b) and (c) show extracted skeleton graphs with $r = 8m$ and $r = 15m$ respectively. As r increases, two vertical streets identified as unique branches merge into a single branch.

Trajectory completion results. The biggest challenge of completing the taxi dataset is that traffic flows on different roads have a large variance. Therefore the choice of minimum flow cluster size M can influence the completion outcome. We evaluate the effect of M using TAXI_1 in Table 7. As M increases, more trajectories with ambiguous projections are discarded, reducing the overall projection rate. At the same time, sub-trajectories assigned to each flow cluster become more compact, which can be completed more easily, thus increasing the completion rate. In practice, we choose $M = 2$ to maximize the chance of up-sampling, then filter unsuccessful completions through post-processing.

Table 6 shows the trajectory completion results for 5000 test trajectories in each test region. On average, 72.7% of input trajectories have been projected to junction network. The completion rate of those completed trajectories is visualized in Figure 13.a.

Region	r_{proj}	# of flow clusters	Running time (sec)
TAXI_1	0.699	125	549
TAXI_2	0.772	262	1877
TAXI_3	0.712	215	1706

Table 6: Trajectory completion results of the taxi trajectories

M	2	4	8	16	32	64
# of flow clusters	96	95	93	88	86	75
r_{proj}	**0.577**	0.573	0.575	0.567	0.567	0.559
$r_{cpl}(20)$	0.561	0.565	0.563	0.566	0.568	**0.571**

Table 7: Impact of M on projection rate and completion rate of TAXI_1.

Comparison with map matching. We evaluate the quality of completed trajectories by comparing against paths found by a state-of-the-art map matching algorithm, interactive voting map matching (IVMM) [22], which uses knowledge of the underlying road network. In our experiments, the road network is obtained from OpenStreetMap [2]. With a search radius of 125m, IVMM matches 98.3% of all test trajectories to the road network. Next, we estimate the completion accuracy $\rho(\delta)$ of test trajectories using map matched

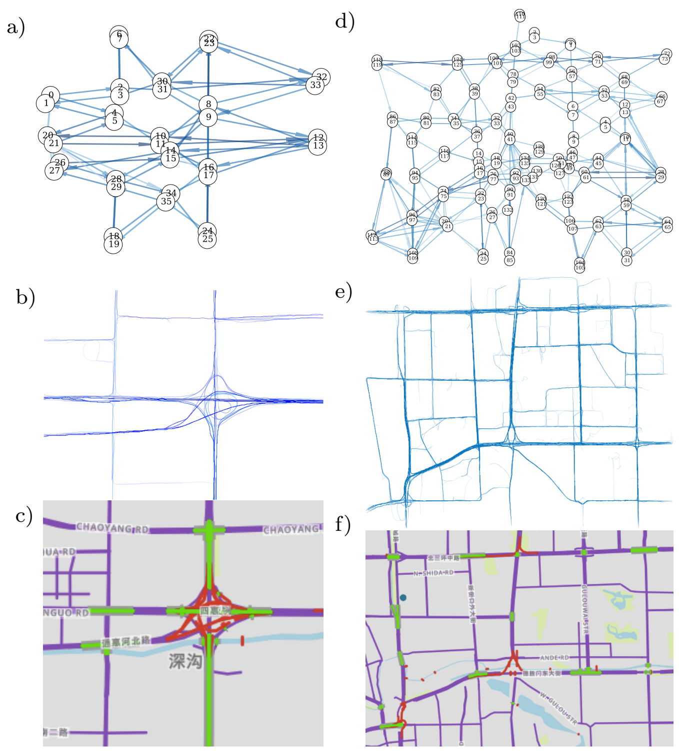


Figure 11: Visualization of junction network, completed trajectories, and the ground truth map for TAXI_1 (a-c) and TAXI_2 (d-f). For junction network edges, dark color implies larger flow cluster size; For trajectories, darker color implies denser samples. The ground truth maps are created using MapBox [1]. Only major roads are shown for clarity. Junctions and junction links are colored in red and green respectively.

paths as the ground truth. As shown in Figure 13.b, 89.5% of points in the completed trajectories are within 15 meters from the map matched path, representing 30-50% improvement from cubic spline interpolation. One contributing factor to the difference between trajectory completion results and the map matched paths is the route ambiguity from one sample point to the next in the underlying road network. Routes chosen by trajectory completion tend to follow popular traffic flows, while routes chosen by map matching tend to prioritize shortest paths on the road network.

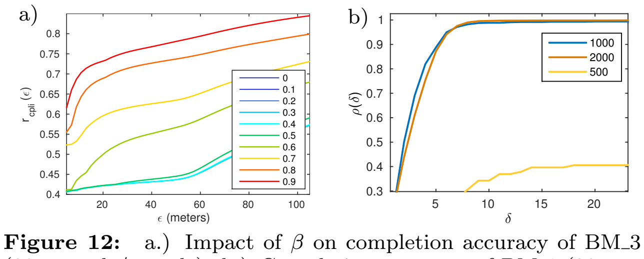


Figure 12: a.) Impact of β on completion accuracy of BM_3 (30 seconds/sample). b.) Completion accuracy of BM_1 (20 seconds/sample) using different input size.

Practical considerations. Real traffic trajectories often span across a city or an even larger region. We need to divide the bounding region of all trajectories into smaller districts to make the problem tractable. Given a trajectory that goes through multiple districts, we can complete partitions of the trajectory in different districts in parallel, then concatenate the generated sub-trajectories together. To reduce inconsistency during concatenation, it is best to place district boundaries in places of low sample density.

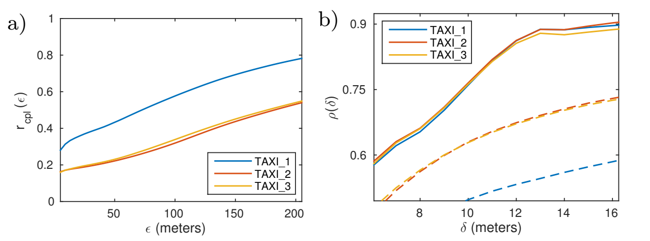


Figure 13: a.) Completion rate of 3 taxi datasets. b.) Completion accuracy of 3 taxi datasets using our method (solid lines) and cubic spline interpolation (dashed lines).

7. RELATED WORKS

Trajectory interpolation. Interpolation is often used to transform discretely sampled trajectories into parametric curves. In [21], Yu et.al. finds the most likely trajectory between two consecutive locations in a trajectory by solving a dynamic problem on sample position, velocity and acceleration. The obvious drawback is fitting error. Hence it is only useful when input trajectories are simple and dense. *Map matching* is well known for reducing uncertainty in trajectories using the geometry and topology of a given road network. There has been extensive work on map matching algorithms for sparse trajectories, such as [16] and [22]. In particular, [12] and [14] use knowledge learned from historical trajectories to resolve path ambiguity in low sampling rate map matching. These work differ from our approach since we don't assume any map knowledge.

Path inference from uncertainty trajectories. Similar to trajectory completion, its key insight is leveraging data from other trajectories sharing the same route. Zheng et. al. [23] infers missing sub-trajectories in a sparse trajectory by matching it to a sequence of reference trajectories generated from historical trajectories. It requires road network information to represent paths. For network-independent scenarios, Goren-Bar's Path Approximation Algorithm [9] first aggregates the trajectory point cloud into groups of points that belong to the same road, then finds the principal curve of each group. It also finds a 3D vector map from the groups. In [19], Wei et. al partitions space into uniform grids, and infer popular routes as connected grid cells. Both algorithms bear similarities with our work, however, they are designed for traces much simpler than urban road network, such as hiking trajectories, or bird tracks.

Trajectory clustering. This problem groups similar trajectories into clusters and compute a representative trajectory for each cluster. It is often involved in path inference and trajectory pattern mining applications [8]. One category of approaches treats trajectory data as point cloud and iteratively move points toward the center line, such as principal curves [6] and partical simulation [9]. Their effectiveness are limited due to not directly using the sequential information in trajectories. Other approaches focus on shared sub-trajectories. One representative algorithm is TraClus, which first partitions trajectories into a set of line segments, then groups similar line segments into a cluster [13]. Trajectory clustering may also use prior knowledge of the road network. Han et. al. proposed a road network aware trajectory clustering (NEAT) algorithm that discover sub-trajectories that describe both dense and highly continuous traffic flows of the user [10]. This work share similar intuition with our work on using traffic flows to represent sub-trajectory clusters. Another related problem is *trajectory calibration*, introduced by Su et. al. [18]. The proposed algorithm transforms heterogeneously sampled trajectories into a uniform sampling profile by aligning trajectories to a set of anchor points. The up-sampling step in this work is a case of trajectory completion.

Map inference. A different approach to solve trajectory completion is first reconstructing the road network from trajectories, a process called "map inference", then applying map matching algorithms. See [4] for a survey of recent advancement on this topic. The accuracy of trajectory-based map reconstruction is often limited when input trajectories are collected in low sampling rates. A recent work by Shan et. al. presents a map update system COBWEB, which takes unmatched trajectories and computes missing edges for an existing map [17]. Our algorithm is similar to COBWEB as they both use GPS samples from all trajectories to infer unknown map structure. The novelty of our work is using junction-based clustering of sub-trajectories to ensure all GPS samples in a flow cluster are from the same route, making trajectory completion more stable.

8. CONCLUSION

In this paper, we proposed a methodology for the trajectory completion problem. Its core idea is that when the whole data is not equally hard to analyze, we can extract information starting from relatively easier regions, and propagate to solve for more challenging regions. We adapted the L1-medial skeleton algorithm to extract trajectory skeletons. From the skeleton branches that are simple in terms of road geometry and traffic flow, information from different sub-trajectories is robustly aggregated. We built a junction network out of the aggregated information, where traffic flows between junction branches can be computed. Guided by the junction knowledge recovered from data, sub-trajectories that follow same topology in the junction network can be clustered and used for completing individual trajectories.

Our approach does not rely on existing map, yet still produces quality trajectory completion from sparse GPS samples. Comparing to other path inference works, it is most powerful when individual trajectories are sparsely sampled, but over all trajectories, there are sufficient GPS samples covering most of the underlying network. Although we have focused on urban traffic trajectories, this algorithm may also be used to complete trajectories where accurate map is less available, such as biking trails in rural area.

In future work, we will take into consideration of GPS noise, so that completed trajectories don't have to contain the input GPS samples exactly. In addition, as real world road networks are constantly updating, it will be interesting to apply our algorithm to trajectory-based map update applications. We believe that the idea of extracting junction traffic flow can improve existing map update methods.

9. ACKNOWLEDGMENTS

The US authors acknowledge the support of NSF grants DMS-1228304 and CCF-1514305, a Google Research Award, and the Max Planck Center for Visual Computing and Communication. D. Gunopulos acknowledges support from the European Union Horizon 2020 Programme (Horizon2020/2014-2020), under grant agreement n688380 and by a Yahoo Faculty Research and Engagement Program Award.

10. REFERENCES

[1] Mapbox. https://www.mapbox.com.

[2] Open street map. http://www.openstreetmap.org.

[3] Taxi trajectory open dataset. http://sensor.ee.tsinghua.edu.cn, 2009.

[4] M. Ahmed, S. Karagiorgou, D. Pfoser, and C. Wenk. A comparison and evaluation of map construction algorithms. *CoRR*, abs/1402.5138, 2014.

[5] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz. SUMO - Simulation of Urban MObility: An Overview. In *SIMUL 2011*, pages 63–68, Barcelona, Spain, Oct. 2011.

[6] C. Brunsdon. Path estimation from gps tracks. 2007.

[7] M. Ester, H. peter Kriegel, J. S, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.

[8] F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi. Trajectory pattern mining. In *ACM SIGKDD 2007*, pages 330–339, 2007.

[9] T. Goren-Bar and J. Greenfeld. A method for constructing 3d traveling routes from gps navigation data. In *ACM SIGSPATIAL International Workshop on GeoStreaming*, IWGS '12, pages 91–100, 2012.

[10] B. Han, L. Liu, and E. Omiecinski. Neat: Road network aware trajectory clustering. In *IEEE ICDCS 2012*, pages 142–151, 2012.

[11] H. Huang, S. Wu, D. Cohen-Or, M. Gong, H. Zhang, G. Li, and B. Chen. L1-medial skeleton of point cloud. *ACM Trans. Graph.*, 32(4):65:1–65:8, July 2013.

[12] T. Hunter, P. Abbeel, and A. M. Bayen. The path inference filter: model-based low-latency map matching of probe vehicle data. In *Algorithmic Foundations of Robotics X*, pages 591–607. Springer, 2013.

[13] J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: a partition-and-group framework. In *ACM SIGMOD 2007*, pages 593–604, 2007.

[14] Y. Li, Q. Huang, M. Kerber, L. Zhang, and L. Guibas. Large-scale joint map matching of gps traces. In *ACM SIGSPATIAL 2013*, pages 214–223, 2013.

[15] S. Morris and K. Barnard. Finding trails. In *IEEE CVPR 2008*, pages 1–8, 2008.

[16] P. Newson and J. Krumm. Hidden markov map matching through noise and sparseness. In *ACM SIGSPATIAL 2009*, pages 336–343, 2009.

[17] Z. Shan, H. Wu, W. Sun, and B. Zheng. Cobweb: A robust map update system using gps trajectories. In *ACM UbiComp '15*, pages 927–937. ACM, 2015.

[18] H. Su, K. Zheng, H. Wang, J. Huang, and X. Zhou. Calibrating trajectory data for similarity-based analysis. In *ACM SIGMOD 2013*, pages 833–844, 2013.

[19] L.-Y. Wei, Y. Zheng, and W.-C. Peng. Constructing popular routes from uncertain trajectories. In *ACM SIGKDD 2012*, pages 195–203. ACM, 2012.

[20] S. Winter. Modeling costs of turns in route planning. *GeoInformatica*, 6(4):345–361, 2002.

[21] B. Yu and S. H. Kim. Interpolating and using most likely trajectories in moving-objects databases. In *Database and Expert Systems Applications*, pages 718–727. Springer, 2006.

[22] J. Yuan, Y. Zheng, C. Zhang, X. Xie, and G.-Z. Sun. An interactive-voting based map matching algorithm. In *IEEE MDM'16*, pages 43–52. IEEE Computer Society, 2010.

[23] K. Zheng, Y. Zheng, X. Xie, and X. Zhou. Reducing uncertainty of low-sampling-rate trajectories. In *ICDE 2012*, pages 1144–1155. IEEE, 2012.

[24] Y. Zheng. Location-based social networks: Users. In *Computing with Spatial Trajectories*, pages 243–276. Springer New York, 2011.

[25] Y. Zheng. Trajectory data mining: An overview. *ACM Trans. Intell. Syst. Technol.*, 6(3):29:1–29:41, May 2015.