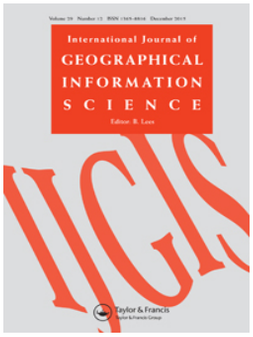


International Journal of Geographical Information Science

ISSN: 1365-8816 (Print) 1362-3087 (Online) Journal homepage: http://www.tandfonline.com/loi/tgis20

Robust inferences of travel paths from GPS trajectories

Hengfeng Li, Lars Kulik & Kotagiri Ramamohanarao

To cite this article: Hengfeng Li, Lars Kulik & Kotagiri Ramamohanarao (2015) Robust inferences of travel paths from GPS trajectories, International Journal of Geographical Information Science, 29:12, 2194-2222, DOI: 10.1080/13658816.2015.1072202

To link to this article: http://dx.doi.org/10.1080/13658816.2015.1072202

Published online: 29 Jul 2015.

International Journal of Geographical Information Science, 2015
Vol. 29, No. 12, 2194–2222, http://dx.doi.org/10.1080/13658816.2015.1072202

Robust inferences of travel paths from GPS trajectories

Hengfeng Li*, Lars Kulik and Kotagiri Ramamohanarao

Department of Computing and Information Systems, The University of Melbourne, Melbourne, Australia

(*Received 12 April 2015; accepted 7 July 2015*)

Monitoring and predicting traffic conditions are of utmost importance in reacting to emergency events in time and for computing the real-time shortest travel-time path. Mobile sensors, such as GPS devices and smartphones, are useful for monitoring urban traffic due to their large coverage area and ease of deployment. Many researchers have employed such sensed data to model and predict traffic conditions. To do so, we first have to address the problem of associating GPS trajectories with the road network in a robust manner. Existing methods rely on point-by-point matching to map individual GPS points to a road segment. However, GPS data is imprecise due to noise in GPS signals. GPS coordinates can have errors of several meters and, therefore, direct mapping of individual points is error prone. Acknowledging that every GPS point is potentially noisy, we propose a radically different approach to overcome inaccuracy in GPS data. Instead of focusing on a point-by-point approach, our proposed method considers the set of relevant GPS points in a trajectory that can be mapped together to a road segment. This clustering approach gives us a macroscopic view of the GPS trajectories even under very noisy conditions. Our method clusters points based on the direction of movement as a spatial-linear cluster, ranks the possible route segments in the graph for each group, and searches for the best combination of segments as the overall path for the given set of GPS points. Through extensive experiments on both synthetic and real datasets, we demonstrate that, even with highly noisy GPS measurements, our proposed algorithm outperforms state-of-the-art methods in terms of both accuracy and computational cost.

Keywords: vehicle trajectory; map matching; GPS traces; path inference

1. Introduction

Monitoring and predicting traffic conditions are of utmost importance in reacting to emergency events in time and for computing the real-time shortest travel-time path. Large datasets collected from various types of sensors are being frequently used to better predict the traffic conditions of an area such as a downtown area or an entire city. In general, there are two types of sensor sources for traffic data: stationary sensors and mobile sensors. Stationary sensors (e.g., loop detectors and infrared speed detectors) are installed at a fixed position of a street to measure traffic volume or speeds, respectively. Mobile sensors, such as onboard GPS devices and smartphones, are mostly attached to vehicles or carried by human beings. Mobile sensors usually sample the vehicle location based on a certain time interval. The mobile sensors play an important role in current transportation research, because they can cover a much larger area and are relatively inexpensive as they are easier to install than the stationary sensors. Researchers have

*Corresponding author. Email: henli@student.unimelb.edu.au

employed many applications exploring mobile sensors, such as activity pattern recognition (Liao *et al*. 2007), route navigation (Yuan *et al*. 2010a), traffic prediction (Yuan *et al*. 2011), and driving turn prediction (Krumm 2008).

Map matching is a well-known technique for associating GPS trajectories to a road network. To predict paths that vehicles actually travelled, we map the GPS points to road segments and estimate a route on the road network. Trajectory data collected by mobile sensors cannot be directly employed for estimating the traffic of the road network because of its imprecise reported coordinates. The inaccuracy of trajectories comes from the GPS measurement error caused by signal noises during data transmission between the GPS receiver and positioning satellites, e.g., refraction in the Earth's atmosphere or reflection by crowded and tall buildings. According to the USFAA (Federal Aviation Administration) GPS performance analysis report published in July 2014,[1] the GPS error level is approximately 4.6 m vertically and 3.3 m horizontally at a 95% confidence level provided by their high-quality GPS receivers. However, in real-world scenarios, many uncontrollable factors, such as atmospheric conditions, receiver quality, and sky-scrapers, can affect actual GPS accuracy.

Existing methods for addressing the *map matching* problem are based on a point-by-point strategy, e.g., finding the closest road segment for each GPS point (White *et al*. 2000). An example is illustrated in Figure 1(a). In the example, e_1 is the closest segment to p_1, p_2, and p_3. e_4 is the closest segment to p_4. e_2 is the closest segment to p_5 and p_6. Then, the algorithm generates $e_1e_4e_2$ as the best estimation. This estimation is incorrect because p_4 is an outlier, leading to a mismatch. Another example of point-by-point approaches is proposed by Brakatsoulas *et al*. (2005). This approach considers two consecutive GPS points as a line segment, which is mapped to a road segment. For example, in Figure 1(a), p_1p_2 and p_2p_3 can be mapped to e_1, p_3p_4 and p_4p_5 to e_4, and p_5p_6 to e_2. In addition to distance, this approach takes the orientation into consideration. The next example of point-by-point approaches is Hidden Markov Model (HMM) map matching (Newson and Krumm 2009), which is known as the best algorithm for solving map matching problem in the literature. The key point of this method is the travel cost in the graph. The authors use the shortest path distance as that cost, e.g., in Figure 1(a), p_4 have two projection points, p'_4 and p''_4, on e_4 and e_2, respectively. Although the travel distance of $p'_3p''_4$ in the graph is less than $p'_3p'_4$'s, p'_4 is more close to p_4 in terms of perpendicular distance. Therefore, HMM can also give the incorrect result $e_1e_4e_2$ with a high probability. HMM does not effectively avoid the influence from noisy points (outliers) such as p_4 in Figure 1(a). Bierlaire *et al*. (2013) proposed generation of a set of candidate paths for the entire GPS trajectory. In Figure 1, the algorithm may generate three possible ranked routes, $e_1e_4e_2$, e_1e_2, and $e_1e_3e_2$, and choose one of them as the

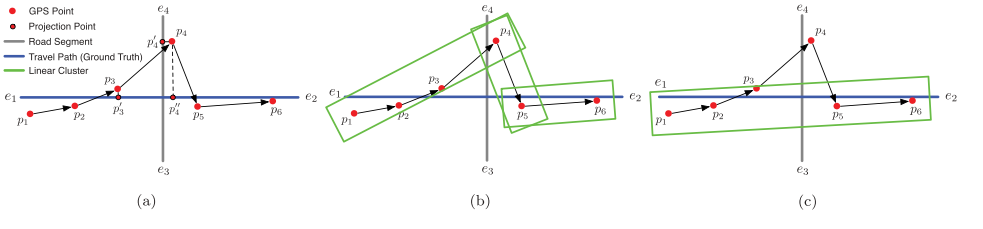


Figure 1. A simple example to illustrate the difference between our method and the existing point-by-point approach. (a) A GPS trace (1 Hz sampling rate). (b) A possible solution of spatial-linear clustering. (c) A better solution.

predicted path. The path-generation procedure uses a greedy point-based strategy computing the possible routes between two consecutive points with the shortest path algorithm. This algorithm is not able to guarantee an optimal solution and, moreover, is very inefficient in space to store all possible routes. The algorithm proposed by Chen *et al.* (2014) is similar to an HMM technique and considers the travel cost between candidate edges, but provides a greedy variant that incorporates the speed limit. Unlike HMM, the algorithm has a greedy strategy; for example, if the candidates for p_1, p_2, and p_3 are determined, they will be used for searching the candidate for p_4 and will not change in the future. There is no dynamic technique to search for an optimal path.

One possible way to deal with outliers, such as p_4 in Figure 1, is trajectory preprocessing (Zheng and Zhou 2011) that mainly focuses on two aspects: *trajectory data reduction* and *trajectory filtering*. Trajectory data reduction applies lossy-compression techniques to remove redundant or unnecessary points from a trajectory. Data reduction helps to improve the map matching accuracy under high sampling intervals. For example, a spatial sampling method is used by Newson and Krumm (2009) that discards GPS points based on a certain distance. Another example is the trajectory simplification (Li *et al.* 2014) that simplifies the trajectory data based on its geometric characteristics. However, both methods discard data points that may be an outlier or redundant, and would lead to the loss of some information from the original trace. In addition to the data reduction, trajectory filtering can smoothen the noise and decreases the measurement error, such as a mean and median filter. The mean or median filter estimates the location of a noise GPS point by calculating the mean or median with its predecessors, for example, moving a sliding window over the trajectory. There are more sophisticated filters, such as Kalman filters (Gelb 1974) and particle filters (Hightower and Borriello 2004). However, a filtering technique is rarely used for GPS data unless the data is particularly noisy (Zheng and Zhou 2011).

In this paper, our goal is to develop an algorithm that matches the correct travel path to a GPS trace. This goal is different from individually matching each GPS point to its true location on the road network, a goal that is shared by many point-based matching algorithms. There are three reasons for our approach. First, for high-level transportation applications such as traffic monitoring and prediction for road segments, we need to build a complete and connected travel path from a GPS trace. From our experience, some road segments in a travel path are not captured by a single GPS point even when the trajectory data are dense. A fully connected travel path is more meaningful than the aggregation of the true locations of GPS points, which may miss some road segments, especially when data are sparse. Second, many point-based matching algorithms evaluate their matching accuracy by building travel paths from their point-based results. They compare the estimated path with the ground truth in order to compute their similarity. Last, we only know that the true location of a GPS point is located in a part of the travel path covered by the measurement error circle of that point. Therefore, the true locations of GPS points are unknown. However, the travel path (footprint) of a GPS trace can be the real ground truth without any assumption. Although our goal is to infer an accurate path, we still need to understand the use cases for different outcomes of map matching algorithms. In our approach, we can use our matching results for monitoring traffic flow (see our case study in Section 6.5). By further estimating the enter and exit time of each road segment for travel paths, we can acquire the accurate distribution of travel time for visited road segments. However, point-based matching also has important use cases, such as matching a moving vehicle and discovering stop-and-go traffic patterns. Our proposed algorithm does not apply to these use cases.

To achieve our goal, we propose an approach that maps GPS points to road segments based on GPS point clusters. This cluster-based method is more accurate and efficient for high-sampling rate datasets. We adopt spatial-linear clusters in this paper that assumes a set of GPS points can be represented by a line segment with an error-bounded buffer. Examples of two possible solutions for spatial-linear clustering are shown in Figure 1(b) and (c). The first solution fits the trajectory perfectly, but it generates two short clusters due to p_4. If we can ignore p_4, then we can obtain a better solution, as shown in Figure 1(c), that is closer to the actual travel path. The clustering method can identify outliers because it has knowledge of the local movement, which is not possible with methods that are point-based. For example, in Figure 1(c), we can see that p_4 is on the outside of the spatial-linear cluster but all other points are in the cluster. In addition, for high-sampling rate datasets, a set of GPS points can better represent the overall movement of a trajectory (e.g., e_1e_2 is more similar to the spatial-linear cluster than other possible route segments). In addition, mapping a set of GPS points contributes to significant improvement in performance in terms of accuracy and computational cost compared to method based on mapping individual points.

Furthermore, our cluster-based method integrates HMM into map matching for low-sampling rate datasets, when each GPS point matters. In this case, clustering is not possible and HMM performs well by connecting GPS points. If the sampling rate is high, our algorithm can robustly compute the clusters based on a given threshold. For very sparse trajectory data, each point may become its own cluster. Route segments for each cluster are ranked based on the similarity to a cluster. The task of connecting spatial-linear clusters is delegated to HMM that searches for an optimal path from all possible combinations.

More specifically, our contributions are as follows:

- Efficient cluster-based mapping algorithm: We propose an algorithm for map matching based on clusters. The algorithm clusters GPS points of a trajectory based on the similarity comparing to their neighbors in time and space. Noisy points (outliers) are ignored because they are significantly different from its neighbors in the cluster. We employ clusters for searching similar route segments in the road network. Our method employs HMM to connect all route segments to an optimal estimation of the path the vehicle traveled. Our cluster-based method can integrate HMM to achieve a higher accuracy in low-sampling situations.
- Extensive experiments on both synthetic and real datasets: We evaluate our algorithm on three real datasets through extensive experiments. We generate data with varying sampling intervals and noise levels to ensure our algorithm can work properly under highly noisy conditions. Moreover, we sub-sample real trajectories to evaluate performance under real scenarios. In these experiments, we show the accuracy and efficiency of our proposed algorithm, comparing with other state-of-the-art algorithms. In addition, we make our code publicly available to the community.[2]

The rest of the paper is organized as follows: in Section 2, we review existing methods to address the *map matching* problem. We give preliminary definitions and define the problem in Section 3. Section 4 introduces our spatial-linear clustering algorithm. In Section 5, we describe a cluster-based method for map matching. The experimental study is given in Section 6. Section 7 outlines future research directions.

2. Related work

We categorize state-of-the-art map matching methods into two types: local and global algorithms. A local algorithm can also be called an incremental algorithm or an online algorithm, because it only considers the trajectory data that is available while processing the trajectory step-by-step (e.g., the projection distance of a GPS point to an edge in the road network or the orientation of trajectory segments). However, compared to a local algorithm, a global algorithm is able to use the information of an entire trajectory because a global view of the trajectory is available.

Local algorithms. They usually combine local geometry and the topology of road network to decide which road segment a GPS point should be mapped to. Determining the similarity between trajectory segment and candidate edges is a common approach in the field. For example, four map matching algorithms are introduced in White *et al*. (2000), finding the closest road segment, considering the heading of consecutive points, considering the topology of road network, and computing the similarity between trajectories and road segments. Other examples of local algorithms are given in Greenfeld (2002) and Quddus *et al*. (2003); their approach first locates the closest node (intersection) and then retrieves all its connected edges as potential candidates. Their methods evaluate the proximity and orientation of candidate edges relative to the trajectory segment. Similarly, Zhou and Golledge (2006) proposed to incrementally record the encountering nodes (street intersections) and to measure the accumulated degree of variance in terms of distance and orientation between the GPS trace and the estimated travel path. Instead of searching the closest node, the approach in Ochieng *et al*. (2003) and Velaga *et al*. (2009) searches a set of potential candidates close to a point. It selects candidate edges that are similar to the trajectory segment based on proximity and orientation. Similarly, the authors in Brakatsoulas *et al*. (2005) propose an incremental algorithm based on proximity and turning angles between trajectory segment and candidate edges. Based on this, they also describe an improved approach that considers several points ahead of the current point to be matched. Instead of only considering the local segments, it is more robust to evaluate the similarity between the entire estimated path and the global trajectory. The approaches in Marchal *et al*. (2005) and Chen *et al*. (2014) define the score of a projected path, which is the summation of the Euclidean distance to the closest road segment for each point in the trajectory. The algorithms expand the estimated path incrementally following the network topology and continue searching for best local choices to minimize the score of that path. These incremental curve-to-curve algorithms are simple and straightforward, but can have a high error rate under noisy conditions and complex road networks (e.g., when the trajectory becomes sparse, existing local algorithms can result in low accuracy levels).

Global algorithms. Global algorithms for map matching can be divided into two categories: those that aim to maximize the similarity between an estimated path and that GPS trajectory globally and those that present the problem as a probabilistic model. For the former category, a distance function is defined to measure the similarity of two curves. Using this function, the algorithms search for the path in the road network that is the closest to the original GPS trajectory. The global matching algorithm in Brakatsoulas *et al*. (2005) is a typical global curve-to-curve algorithm that aims to minimize the Fréchet distance between an estimated path and a GPS trajectory. Similarly, Yin and Wolfson (2004) also proposes a curve-to-curve method that precomputes the Euclidean distance between each trajectory segment and edges in the map. Then, the algorithm searches a path in the graph with the minimum weight. The drawback of these algorithms is that computing the similarity of two

curves globally is very slow for a large number of GPS points. For the latter group, the map matching problem is modeled as a probabilistic problem, e.g., the probability of a GPS point matching to a candidate edge and the probability of representing the topology of a road network. The final prediction is the path with the maximum probability. A classic approach is the HMM map matching (Newson and Krumm 2009), which assumes that GPS measurement errors are under a zero-mean Gaussian distribution and the ratio of comparing Euclidean distance of two consecutive points to their route distance obeys an exponential distribution. Similar methods using HMM are proposed in Lou *et al.* (2009) and Yuan *et al.* (2010b). Bierlaire *et al.* (2013) proposed to generate all possible true paths and rank these paths based on the predefined probability function considering spatial and temporal properties of GPS points. In another work, Li *et al.* (2013) proposed first clusters with connected edges as a segment, given a specific distance. The algorithm merges the selected segments for GPS points into a projected path, and the defined objective function is used to search the optimal selection of joint segments.

The *selective look-ahead map matching (SLAMM)* in Weber *et al.* (2010) is somewhat similar to our proposed algorithm but fundamentally differs in that SLAMM is still a point-based matching algorithm that identifies critical samples (GPS points close to intersections) and searches candidate edges for them. However, most cities have a very complex road network, in which some curved roads are represented by small straight segments in digital maps. With many intersections (nodes) among these small segments, determining critical samples becomes difficult. In their paper, intersections (nodes) have an ideal distance to each other, which is not true in actual road maps. The unique feature of our proposed algorithm is that we map a set of GPS points together to a road segment. We care more about the similarity of a candidate route to the set of GPS points instead of the closeness of a single edge to a single GPS point. In other words, we consider the similarity of a curved road to our GPS points rather than choosing a small segment close to a GPS point.

Based on the current work, we aim to develop a path inference algorithm that satisfies the three following properties:

- Accuracy: Our goal is to infer the travel path from a GPS trace. A high accuracy means that the estimated path (footprint) and the ground truth are close. In our experiments and case study, we empirically show that our proposed algorithm outperforms the state-of-the-art algorithms in nearly all situations.
- Efficiency: A good map matching algorithm should have an efficient execution time for a long trace with a large number of GPS points. By mapping a set of points together, our proposed algorithm is significantly faster than the state-of-the-art algorithms while maintaining high accuracy.
- Robustness: A robust algorithm has a good performance even in the presence of high noise. Our proposed algorithm clusters a group of points with error-bounded buffers and maps them together. A cluster represents the moving direction of a vehicle. GPS points with high noise do not affect the overall moving direction. In the experiments, our proposed algorithm achieves high accuracy even in the presence of high noise.

3. The map matching problem

In this section, we present preliminary definitions that are commonly used. On this basis, we will describe the map matching problem.

3.1. Preliminary definitions

Definition 3.1: (Road network). $G = (V, E)$ is a graph representing the road network. The vertices V represent intersections or end points in a road network and the edges E correspond to directed road segments.

Thus, a vertex $v \in V$ can only be a start or an end point of an edge $e \in E$.

Definition 3.2: (Original path). The original path P is a sequence of road segments to represent the footprint of a vehicle's movement, where $P = [e_{p_1}, e_{p_2}, ..., e_{p_N}]$ and $e_{p_N} \in E$. Additionally, $\forall\ e_{p_i}, e_{p_{i+1}}\ \exists\ v_l, v_m, v_k\ \in V$ such that $e_{p_i} = <v_l, v_m>$ and $e_{p_{i+1}} = <v_m, v_k>$.

Definition 3.3: (Raw trajectory). The raw trajectory T consists of measured GPS points along the original path P during a fixed or varied time intervals. A trajectory function f_T can be defined as:

$$for\ t \in [t_1, t_2, \ldots, t_n]: \qquad t \mapsto f_T(t) = (\lambda, \delta)$$

Each GPS point is represented by the spherical coordinates with its measured longitude λ ($0 \leq \lambda < 2\pi$) and latitude δ ($-\pi \leq \delta \leq \pi$) at a certain timestamp t. Due to GPS measurement noise, the location of each point is uncertain.

Definition 3.4: (Map matching). Map matching is a process of path inference that maps a raw trajectory T to a complete and fully-connected inferred path P' in the road network $G = (V, E)$. The process can be defined as a function *Map*:

$$P' = Map(T)$$

where $T = [p_{t_1}, p_{t_2}, ..., p_{t_n}]$, $P' = [e_{p'_1}, e_{p'_2}, ..., e_{p'_M}]$, and $e_{p'_M} \in E$.

Definition 3.5: (Path distance measure). The path distance measure is a function $Dist(P_1, P_2)$ to calculate the closeness of two paths, P_1 and P_2. A high value given by the measure indicates a low level of similarity between two paths.

$$\forall P_1, P_2: \qquad P_1, P_2 \mapsto Dist(P_1, P_2) \in \mathbb{R}_{\geq 0}$$

Some possible path distance measures, such as the synchronous Euclidean distance (Potamias *et al*. 2006), the closest-pair distance, and the sum-of-pairs distance (Zheng and Zhou 2011), are based on computing the distance of each pair on two trajectories of the same length. However, they cannot deal with trajectories that have different number of points. To compute the distance between trajectories of different lengths, both the Hausdorff distance (Huttenlocher *et al*. 1993) and Fréchet distance (Alt and Godau 1995) compute a global distance between two trajectories. However, a single outlier can easily affect the similarity result. Generally, these measures cannot distinguish between two trajectories if they have the same global distance to a reference trajectory. To address the above issue, other methods are used in computing the similarity of trajectories, such as Dynamic Time Warping (DTW) (Berndt and Clifford 1994), Common Longest Subsequence (LCSS) (Vlachos *et al*. 2002), and Edit Distance on Real Sequences (EDR) (Chen *et al*. 2005). They compute the best alignment of two trajectories and

then search the matched and the mismatched parts of two trajectories. However, the search of the best alignment is very slow when a trajectory consists of thousands of points. Since we evaluate the similarity of paths that consists of a sequence of road segments, the F_1 score is more efficient as a path distance measure (see the evaluation methodology in Section 6.3).

3.2. Problem description

The map matching algorithm estimates a path P' from a sampled GPS trajectory T. The map matching accuracy is measured by the similarity between the projected path P' and its actual path P. The map matching error mainly arises from GPS measurement error. The GPS measurement noise leads to uncertain locations of sampled GPS points that are not on the actual path P. Such uncertain locations may lead to many mismatches in map matching.

To quantify map matching accuracy, we need to define the distance measure $Dist(P_1, P_2)$ for two paths P_1 and P_2. This measure is used to compare the closeness of the original path P and the projected path P'. To evaluate two map matching methods M_1 and M_2, M_1 is better than M_2 if $Dist(P, P'_{M_1}) < Dist(P, P'_{M_2})$ exists, and vice versa. In our experiments, we combine the precision and recall as the distance measure by calculating the harmonic mean of both, called the F_1 score (Section 6.3).

4. Spatial-linear clustering algorithm

In this section, we illustrate our proposed algorithm with examples and analyze its time complexity under the worst case and the average case scenarios. Finally, we discuss the parameterization of our algorithm.

4.1. Illustration of algorithm

Under significant noise in GPS data, a single measured GPS point might be in an incorrect location, which leads to an error in the estimated path. Our approach to reduce the impact of noise in GPS data acquisition is to use multiple GPS points to predict the general movement, and match these points jointly to obtain an accurate estimation. We propose the Oriented Bounding Rectangle (OBR) algorithm, which clusters GPS points based on the moving direction of objects. In the OBR algorithm, we take into consideration the error-bounded distance (d_{error}), the minimum number of support points in a rectangle (k), and the maximum number of outliers (n_{outlier}).

We create an initial OBR from two circles that are error bounds of the first two consecutive GPS points (e.g., two circles of p_2 and p_3 in Figure 2(a)). A pair of points, such as p_2 and p_3, are chosen as the anchor points. The radius of circles are determined by the error-bounding distance d_{error}. An OBR is the minimum rectangle that completely covers the area of that two circles (e.g., the rectangle R_2 covers the circles of p_2 and p_3). Moreover, as more points are added into the OBR, the OBR increases its length and the anchor points may change to a new pair, such as p_2 and p'_3. When switching to new anchor points, the orientation of the OBR changes as well. Therefore, the bounding rectangle is directed from one anchor point to another. Anchor points are chosen when a new cluster is created due to direction changes. For example, in Figure 2(a), when moving from p_2 to p_3, we treat p_3 as a potential anchor point. We make sure that points originally in the cluster are kept if

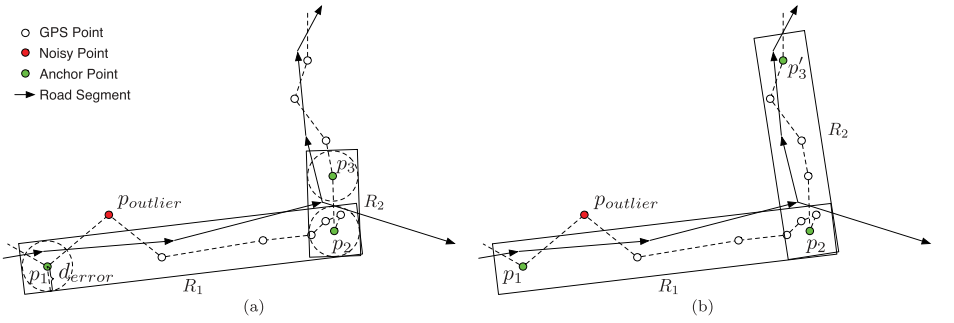


Figure 2. An example of the Oriented Bounding Rectangle (OBR) algorithm to build groups of GPS points with error-bounded distance d_{error}. The rectangle R_1 has been built with an outlier point p_{outlier} and the rectangle R_2 is growing gradually.

the current anchor point changes to p_3. If points are not part of the cluster, it indicates a significant change on the moving direction and we need to create a new cluster in order to maintain the integrity of current cluster. It is important to note that the selection of anchor points does not affect the accuracy and performance of the proposed algorithm. For example, anchor point p_2 can be changed to any other point around its current location. This does not affect the final matching results because the algorithm takes the similarity of candidate routes and the cluster of points into account. The number of anchor points are determined by the formed clusters which are given by the error-bounded distance d_{error}.

An example of the OBR algorithm is illustrated in Figure 2. In the figure, GPS points (white circles) on a trajectory are ordered along road segments (black arrows). The GPS points represented by filled colors are the anchor points (green circles) and the noisy points (red circles). More specifically, the anchor points, such as p_1, p_2, and p_3 in Figure 2(a), are important to build the oriented rectangles, which determine the direction of its major axis. The noisy points are outside the oriented rectangle. For example, in Figure 2(a), p_{outlier} is a noisy point that can be ignored when building the oriented rectangle.

In Figure 2(a), we can see that there are two oriented rectangles R_1 and R_2. R_1 is formed by two anchor points p_1 and p_2, while R_2 is created by p_2 and p_3. An important parameter of the OBR algorithm is d_{error}, which is the assumed GPS error. Figure 2(a) shows that d_{error} is the distance from an anchor point to the closest three borders of a rectangle. The width of an oriented rectangle is $2 \times d_{\text{error}}$. As the value of d_{error} is determined by the GPS measurement error, we can get a confidence level of the number of GPS points that should be inside the oriented rectangle. For example, in our experiments, we assume that the GPS measurement error obeys a Gaussian distribution; therefore, if the value of d_{error} is two standard deviations, approximately 95% of GPS points should be included by our OBRs. More details about the parameter d_{error} are given in Section 4.3.

Figure 2(b) shows that R_2 is growing to a longer oriented rectangle and its anchor point is switched from p_3 to p'_3. All GPS points between p_2 and p'_3 are located inside the OBR, which indicates that these points are within the error-bounded distance d_{error} to the major axis of OBR. The OBR keeps increasing its size to include as many points as possible until the vehicle turns at an intersection or follows a curved road.

The key point of our OBR approach is to group GPS points spatial-linearly with an error-bounded buffer. For the clustering step, the algorithm has no knowledge of the road

network. The algorithm identifies outliers as follows: (1) they are not part of the spatial-linear cluster; and (2) their close temporal neighbors are inside the cluster.

One simple way to exclude the outliers is to calculate the error bounds of each GPS point and check whether any candidate edges is within its error bounds. If there is no edge in the error bound of a GPS point, that point is classified as an outlier. However, we present two reasons why this method does not work well. First, in Figure 2, if there is an edge close to p_{outlier}, then p_{outlier} will not be classified as an outlier any more according to the description. Second, in Figure 2, if p_2 has no edge in its error bound, p_2 will be erroneously classified as an outlier because all p_2's neighbors are close to it. Therefore, calculating error bounds based on single points does not work well. Compared to the above approach, our proposed method works well because it considers the error bound of a spatial-linear approximation of a set of points.

The algorithm to generate an OBR is given in Algorithm 1, which can be described as follows:

(1) Start with the first two consecutive (GPS) points and compute their error bounds (circles) with the given distance d_{error}.
(2) Compute the minimum bounding rectangle to fully cover the two error bounds (Line 2).
(3) Check the next (GPS) point in temporal order (Line 3) and explore whether the OBR created can include this point as one of its support points using the following conditions:
    (a) Assign a label to an interior point that would become exterior, if the current rectangle uses the new point as an anchor point (Lines 6–11).
    (b) If an interior point and half of its k close neighbors are labeled, the current rectangle cannot include the new point (Lines 13 and 14).
    (c) If an interior point is labeled but its k close neighbors are not labeled, we consider this point as an outlier. The outlier will not be included if the number of removed outliers is less than n_{outlier} (Lines 15–17).
    (d) If the current rectangle cannot include the new point, the algorithm returns the rectangle and the starting position for next rectangle if the number of its support points is over the minimum support k (Lines 18–21); otherwise, we add this new point to existing rectangle (Line 23).
(4) Repeat (3) if there are more unprocessed points in the trajectory.

Algorithm 1: OBR Generation Algorithm

Input : $T = [p_{t_1}, p_{t_2}, ..., p_{t_N}]$ is a raw GPS trajectory that consists of measured GPS points; the starting position t_s for the current rectangle; the error-bounded distance d_{error}; the minimum number of support points in a rectangle k; and the maximum number of outliers n_{outlier}.

Output : A set of GPS points G_{OBR}; the starting position t_{next} for the next rectangle.

begin

$R \leftarrow$ create the initial rectangle based on the error bounds of p_{t_s} and p_{t_s+1} according to d_{error}.

for $p_{\text{new}} \in [p_{t_s+2}, p_{t_s+3}, ..., p_{t_N}]$ **do**

$p_{\text{start}}, p_{\text{end}} \leftarrow$ the first and last point of R

$seq[p_{\text{start}}]$, $seq[p_{\text{end}}] \leftarrow$ *false, false*
for $p_{\text{interior}} \in R$ **do**
  // p_{interior} is one of interior points in R except p_{start} and p_{end}.
  if $dist_{\text{perpendicular}}(p_{\text{interior}}, p_{\text{start}}, p_{\text{new}}) > d_{\text{error}}$ **then**
    // if the perpendicular distance from p_{interior} to the line
    // p_{start} p_{new} is larger than d_{error}, then assign a label to p_{interior}.
    $seq[p_{\text{interior}}] \leftarrow$ *true*
for $s_i \in seq$ **do**
  if s_i *and (half of* s_i*'s* k *neighbors are labeled)* **then**
    isSplit $\leftarrow$ *true*
  else if s_i and (s_i's k neighbors are not labeled) and $count_{\text{outlier}} < n_{\text{outlier}}$
  then
    Mark s_i as an outlier which will not be included in R
    $count_{\text{outlier}} \leftarrow count_{\text{outlier}} + 1$
if *isSplit* **then**
  // Return the current rectangle.
  $t_{\text{next}} \leftarrow$ retrieve the position of p_{new}
  if $length(R) > k$ return R, t_{next}; otherwise, only return t_{next}.
else
  R.append(p_{new})

The algorithm is allowed to remove multiple outliers and relies on more close neighbors to deal with systematic errors. There are two cases of systematic errors. First, several subsequent points are very noisy but they have many neighbors with low noise levels in a cluster. In this case, the algorithm will identify them as multiple outliers because of the dissimilarity relative to the majority of their neighbors. The second case is that, when there are few neighbors with low noise levels, many points in a cluster have high noise levels. This leads to an erroneous cluster for which two outcomes could occur: the algorithm computes non-existing routes or wrong routes. The issue of non-existing routes can be fixed by amalgamating two neighbor clusters based on the assumption that the vehicle takes the shortest path. For the second issue, we use the speed limit to determine whether a path is possible; if not, the shortest path will replace the wrong route.

4.2. Complexity analysis

We show that the time complexity of the OBR algorithm is $O(c \times k \times n)$ according to Algorithm 1, where c is the average number of interior points in the OBR, k is the number of neighbors visited, and n is the total number of GPS points in a trajectory. k is a constant and, in our experiments, we set k to 2. The worst case is $O(n^2)$ where $c = n$, which is an exceptional case wherein the entire trajectory can be represented by a single OBR. This scenario might occur in some trajectories with short distances, in which n is normally very small. For the average case, c can be treated as a constant such that the OBR algorithm can achieve a linear time complexity. d_{error} affects the formation of OBRs; therefore, they influence the value of c as well. When d_{error} increases, the width of OBRs becomes larger so that more points are included, which leads to the increase of c, and vice versa. Thus, the choice of d_{error} influences the running time of the algorithm.

The OBR algorithm is an incremental algorithm for characterizing the shape of a trajectory. It only looks at local situations to decide the group of points. Its early grouping can lead to a global non-optimal solution. However, this results in a fast runtime and improves the efficiency when dealing with large-scale trajectory datasets. The performance of the OBR algorithm is largely determined by d_{error}, which will be discussed in the next section.

4.3. Parameterization

In our algorithm, we consider three parameters: the error-bounded distance (d_{error}), the minimum number of support points in a rectangle (k), and the maximum number of outliers (n_{outlier}). Specifically, the shape of OBRs produced by our proposed algorithm is parameterized using the error-bounded distance d_{error}. In Figure 3, we can see that, when decreasing the value of d_{error} from 4σ to σ, the width of OBRs becomes more thinner and an increasing number of noisy points are excluded from the rectangles. A small d_{error} can lead to the overfitting problem, where a large number of OBRs are created in order to better fit the shape. The issue becomes worse, e.g., when a vehicle is waiting for the traffic light, which leads to an excessive generation of short OBRs. Therefore, we define the minimum number of support points k, which removes OBRs that lack enough support in terms of the number of GPS points. On the other hand, a large d_{error} might fail to provide an accurate geometric shape of the trajectory. As we assumed that the GPS measurement error is under a Gaussian distribution, σ for the OBR algorithm can be directly learned. An appropriate d_{error} is important to the accuracy of our proposed algorithm. In addition, n_{outliers} can effectively control the number of outliers that are removed. In our experiment, we set n_{outlier} to unlimited, which allows to delete as many as points if they are identified as outliers.

5. Cluster-based path search

In this section, we introduce how to map a path in the road network to the generated OBRs. We first present the underlying idea of searching routes inside OBRs and computing an optimal connected path. We then define the objective function and formulate the search procedure as an optimization problem. Finally, we give a dynamic programming solution to compute the optimal connected path given the GPS data.

5.1. Underlying idea

After clustering the GPS points as OBRs, we can search for possible internal routes inside each OBR. Each candidate route in the OBR is assigned to a weight that indicates the probability of being a part of the final route. With multiple groups of candidates, the question becomes how to find an optimal solution to connect them. We will use a dynamic programming approach to address this problem.

The path search algorithm for OBRs follows these steps:

(1) Generate possible routes for each OBR. Routes are formed within the bounded area of each OBR.
(2) Assign a weight to each candidate route by calculating the similarity to the major axis of OBR.
(3) Calculate the probability of a route segment for a cluster. The probability is determined by two parts: the similarity to the point cluster and the cost of

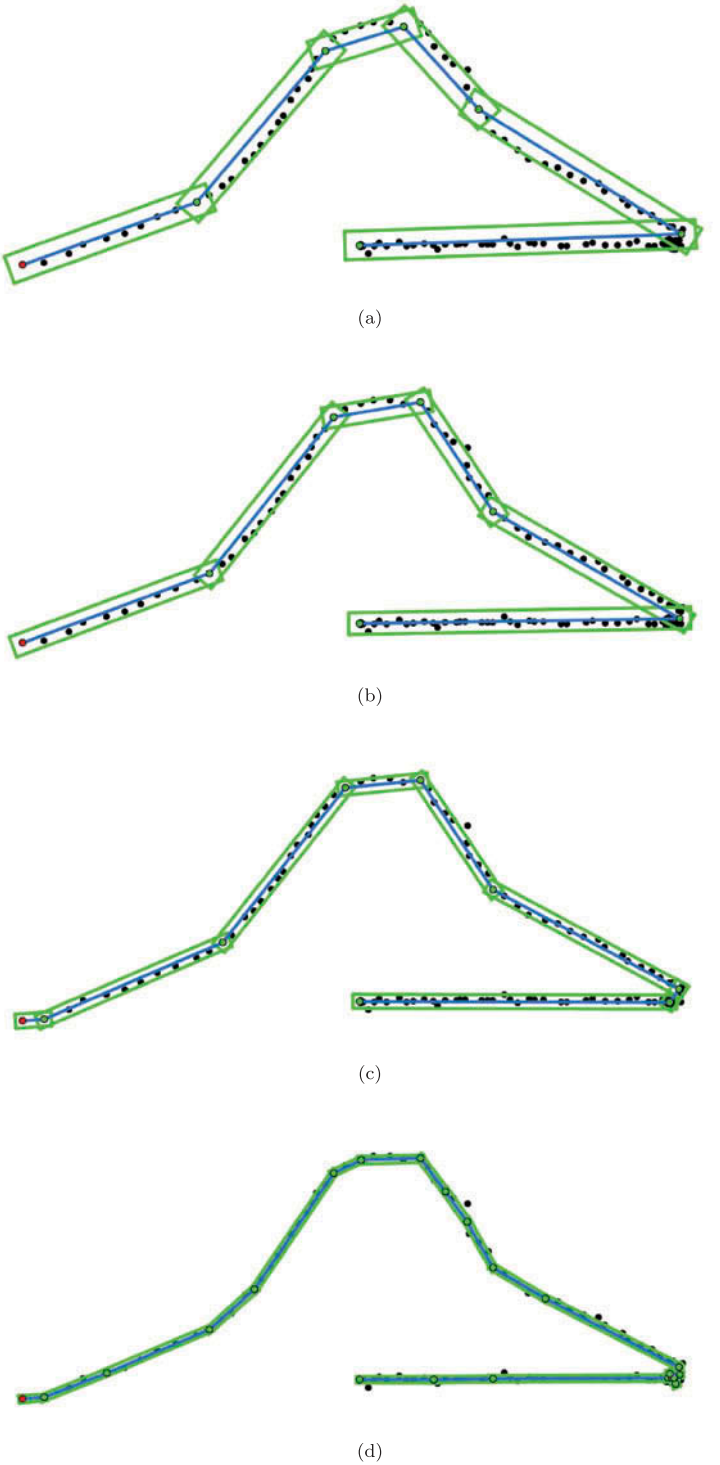


Figure 3. Examples of varying d_{error} parameter for the OBR algorithm. (a) $d_{\text{error}} = 4\sigma$, 6 OBRs; (b) $d_{\text{error}} = 3\sigma$, 6 OBRs; (c) $d_{\text{error}} = 2\sigma$, 8 OBRs; (d) $d_{\text{error}} = \sigma$, 24 OBRs. The red point is the end point of the trajectory.

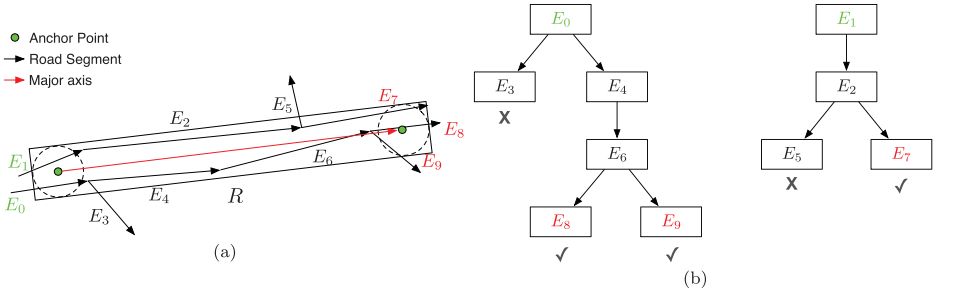


Figure 4. An example of generating possible routes bounded by the OBR. (a) The algorithm records possible routes, e.g., $E_1 \rightarrow E_2 \rightarrow E_7$ and $E_0 \rightarrow E_4 \rightarrow E_6 \rightarrow E_8$, but prunes unlikely routes, like $E_0 \rightarrow E_3$. (b) Two trees are constructed for searching possible routes with starting edges (green color) and ending edges (red color).

connecting the route segment to its previous cluster. The route segment with the maximum probability is chosen.

(4) Determine the optimal route according to the aggregated weight at the last OBR. Backtracking outputs the path that is the optimal combination of the candidate routes in the OBRs.

The strategy of forward-propagation and backtracking is a typical dynamic programming technique to address the optimal path search problem.

5.2. *Searching route segments for spatial-linear clusters*

The algorithm generates potential route segments for each OBR. Figure 4(a) shows that the search for possible routes is bounded by the area of the OBR. This reduces the search space to generate all possible routes. Firstly, the algorithm generates routes based on the road network. The route generation is based on the idea of Breath-First Search (BFS) in order to cover all possible candidates for an OBR. The difference to BFS is that this algorithm aims to generate routes for multiple sources and destinations. In addition, with known destinations, we could prune the trees to reduce the number of candidate routes (Figure 4(b)). Routes ending with E_7, E_8, and E_9, are selected as candidates for this OBR.

Algorithm 2: Cluster-based Path Search

Input : e_{start} is a starting edge; E_{end} is a set of ending edges; R is an OBR; G is the directed graph of road network.
Output : P is a set of possible routes in R starting from e_{start}.
begin
  endEdges ← []
  a ← diffAngle(e_{start}.angle, R.angle)
  mapBacktrack ← {e_{start}.to(): <a, e_{start}>}
  q ← [e_{start}.to()] // Initialize an empty queue
  while !q.*isEmpty()* **do**
    currentNode ← q.pop()
    currAngle ← *mapBacktrack*.get(*currentNode*).first()
    for $e \in G.adj(currentNode)$ **do**

```
        // Iterate all adjacent edges
        nextAngle ← mapBacktrack.get(e).first()
        if currAngle + e.angle > nextAngle then
          continue
        mapBacktrack.add({e.to(): < currAngle + e.angle, e > })
        if E_end.contains(e) then
          endEdges.add(e)
        else if R.isInside(e.to())then
          q.push(e.to())
    for e ∈ endEdges do
      path ← []
      currentEdge ← mapBacktrack.get(e.to())
      while currentEdge ! = e_start do
        path.add(currentEdge)
        currentEdge ← mapBacktrack.get(currentEdge.from()).second()
      path.add(e_start)
      path.reverse()
      P.add(path)
```

Algorithm 2 demonstrates the search implementation for a single starting edge. The algorithm creates a dictionary for backtracking and a queue for recording unvisited nodes in an breath-first order (Lines 4 and 5). For the next unvisited node, we iterate all its adjacent edges (Line 9). The algorithm selects an edge with the minimum turning angle and adds it into the backtracking dictionary (Lines 10–14). If the edge is a terminal edge, we will use it as a starting point for backtracking later (Lines 15–16). If the edge is inside the bounded area, the algorithm adds its ending node into the queue (Lines 17 and 18). Finally, each ending edge will be backtracked for forming a possible route and the algorithm will return all possible routes (Lines 19–27).

5.3. *Optimal path problem formulation*

To project the best route $\vec{r}_{\text{best}}$ according to the clustered groups that we built by the OBR algorithm, we formulate the problem as maximizing the conditional probability of $p(\vec{r}|\vec{g})$ where $\vec{g} = [g_1, g_2, ..., g_n]$ is the sequence of groups of GPS points and $\vec{r} = [r_{g_1}, r_{g_2}, ..., r_{g_n}]$ is the sequence of a single route combination.

$$\vec{r}_{\text{best}} = \arg\max_{\vec{r}\in R} p(\vec{r}|\vec{g}) = \arg\max_{\vec{r}\in R} \frac{p(\vec{r},\vec{g})}{p(\vec{g})} \tag{1}$$

$$= \arg\max_{\vec{r}\in R} \frac{p(\vec{r})\cdot p(\vec{g}|\vec{r})}{p(\vec{g})} \tag{2}$$

$$= \arg\max_{\vec{r}\in R} p(\vec{r})\cdot p(\vec{g}|\vec{r}) \tag{3}$$

$$= \arg\max_{\vec{r}\in R} \prod_{i=1}^{n} p(r_i|r_{i-1})\cdot p(g_i|r_i) \tag{4}$$

In Equation (1), R is the set of all possible combinations for $\vec{r}$. As the set of candidate routes for each group is $\vec{c} = [c_{g_1}, c_{g_2}, ..., c_{g_n}]$ and $r_{g_i} \in c_{g_i}$, we can compute the set of all possible combinations R by enumerating all possible combinations from $\vec{c}$. For a group g_i, we can choose a potential route r_{g_i} from the collection of candidate routes c_{g_i}. Equation (2) is deduced from Equation (1) based on Bayes' theorem and, therefore, we need to calculate the conditional probability $p(\vec{g}|\vec{r})$. Note that we ignore $p(\vec{g})$ in Equation (3) because the set of groups $\vec{g}$ is same for all different $\vec{r}$ and $p(\vec{g})$ does not change its value. In Equation (4), according to the Markov assumption, we calculate the probability of r_i, given the condition that the vehicle travels from the previous route r_{i-1}. We look backward for a single route (assuming $p(r_1|r_0) = 1$ for the virtual route r_0).

In the following part of this section, we discuss the modeling of probabilities $p(r_i|r_{i-1})$ and $p(g_i|r_i)$. First, we model the probability $p(r_i|r_{i-1})$ as a linear weighting function:

$$d_{\text{Euclidean}} = dist_{\text{Euclidean}}(r_{i-1}, r_i)$$

$$d_{\text{Network}} = dist_{\text{ShortestPath}}(r_{i-1}, r_i)$$

$$p(r_i|r_{i-1}) = 1 - \frac{|d_{\text{Network}} - d_{\text{Euclidean}}|}{d_{\text{Euclidean}}} \tag{5}$$

where d_{Network} is the travel distance on the road network, $d_{\text{Euclidean}}$ is the Euclidean distance between two points, and $d_{\text{ShortestPath}}$ is the length of shortest path between two points. If $d_{\text{Network}} > 2d_{\text{Euclidean}}$ is true, we assign a small value ε to $p(r_i|r_{i-1})$, indicating that traveling from r_{i-1} to r_i is highly unlikely.

If no GPS points are reported between routes, we assume that drivers select the shortest path. Therefore, we assign more weight to a connection with a short distance. We normalize $p(r_i|r_{i-1})$ to a range 01.0 that a high value means more likely, and vice versa.

In addition, $p(g_i|r_i)$ is the probability of the group of GPS points g_i belonging to the route r_i. We model $p(g_i|r_i)$ as the similarity of the route r_i and the major axis of g_i's OBR. We calculate the average difference of angle between each edge segment $e \in r_i$ and the major axis a_{major}, and normalize the result to a range 01.0 where a small value refers to a low probability:

$$p(g_i|r_i) = 1 - \frac{1}{n \cdot 180} \sum_{e \in r_i} \text{diff}_{\text{angle}}(e, a_{\text{major}}) \tag{6}$$

where $\text{diff}_{\text{angle}}$ calculates the smallest angle between two vectors.

Finally, with the definitions of $p(g_i|r_i)$ and $p(r_i|r_{i-1})$, we use a dynamic programming technique, the Viterbi algorithm (Forney 1973), to calculate the path with maximum probability $\vec{r}_{\text{best}}$.

When searching for the best path with OBRs, we need to consider the following aspects. Firstly, during the implementation of our algorithm, we observe that $p(\vec{r}, \vec{g})$ is smaller than the smallest positive nonzero value of a float number, 2^{-1074}. This is because $p(\vec{r}, \vec{g})$ is the multiplication of thousands of float numbers that are less than 1.0. To solve this issue, we apply logarithm to $p(\vec{r}, \vec{g})$:

$$\log\ p(\vec{r},\vec{g}) = \log \prod_{i=1}^{n} p(r_i|r_{i-1}) \cdot p(g_i|r_i) = \sum_{i=1}^{n} \log(p(r_i|r_{i-1}) \cdot p(g_i|r_i)) \quad (7)$$

Secondly, we can give different definitions to $p(r_i|r_{i-1})$ and $p(g_i|r_i)$. For example, we could use a more sophisticated distance measure to define $p(g_i|r_i)$, such as Hausdorff distance or Fréchet distance, but those measures may result in longer runtimes. In our approach, we calculate $p(g_i|r_i)$ based on the differences between angles for efficient processing.

6. Experimental study

In this section, we evaluate the accuracy and performance of our method and present the datasets, comparing methods, and evaluation methodology. In terms of evaluation metric, we adopt precision, recall, and F_1 score to calculate the distance $Dist(P_1, P_2)$ between P_1 and P_2. We analyze our experimental results in terms of both accuracy and computational cost. Lastly, we conduct a case study for a specific scenario – estimating the traffic flow from map matching results. We conduct all experiments on a 2011 MacBook Pro with 2.3 GHz Intel Core i5 and 16 GB RAM.

6.1. Datasets

We evaluate the performance of our proposed algorithm using two synthetic trajectory datasets and one trajectory dataset (see Table 1).

6.1.1. Synthetic trajectories

Seattle Dataset. The dataset has been used in Newson and Krumm (2009) that captures 2-hour driving in Seattle. It has 7531 GPS points and contains several time gaps without GPS-recorded points. The GPS trace covers a region between $-122.357316°$ and $-122.086483°$ longitude and 47.563166° and 47.671083° latitude. Moreover, the dataset provides the ground truth, i.e., the road segments on vehicle trajectories.

Melbourne Dataset. The dataset has been used in Li *et al.* (2014), which recorded a 40-minute trip in Melbourne with 2512 GPS points. The road data is extracted from the OpenStreetMap,[3] which covers a region of 144.9316° to 145.0753° longitude and $-37.8269°$ to $-37.7221°$ latitude. Road segments overlapping with the actual trajectory were collected as the ground truth.

Based on the ground truth, we generate noisy trajectory data, in which noise is created under a zero-mean Gaussian distribution with a standard deviation of σ meters. More

Table 1. Description of experimental datasets.

Source	Location	Sampling	Trips	Size	Duration	Length (km)
Newson and Krumm (2009)	Seattle	per second	1	7531	~2 hours	80
Li *et al.* (2014)	Melbourne	per second	1	2512	~40 minutes	16
Ali *et al.* (2012)	Seattle	per second	10	14,436	~4 hours	228

precisely, assuming the coordinate of a GPS point is (x, y), then the location of a noisy point is $(x + r\cos\theta, y + r\sin\theta)$, where the radian θ is under an uniform distribution $\mathcal{U}(0, 2\pi)$ and the radius r (in meters) is under a Gaussian distribution $\mathcal{N}(0, \sigma^2)$. In addition, we create input data with different sampling rates by taking a sub-trajectory from the generated trajectory for a given time interval. The noise levels are set to 1, 2, 4, 8, 16, and 32 m and the sampling intervals are varied to 1, 2, 4, 8, 16, and 32 seconds. For each setting of noise level and sampling rate, we run 20 separate experiments, in which the input trajectories are different. In total, we generate 720 trajectories for each dataset.

6.1.2. *Real trajectories*

GIS Contest 2012. The dataset has been used in a map matching challenge (Ali *et al.* 2012). The dataset contains the GPS traces of 10 trips. The total travel time of the trips is approximately 4 hours. The GPS traces contain 14,436 GPS points that cover a region between $-122.385985°$ and $-122.022721°$ longitude and 47.443856° and 47.844388° latitude. The ground truths are also given in Ali *et al.* (2012).

We sub-sample the real trajectories to test the performance of our algorithm on different sampling-rate conditions. The original trajectory is sampled once per second. We degrade the sampling interval to 2, 4, 8, 16, and 32 seconds. Our experiments cover all sub-sampled data, e.g., a trip with 1-second sampling interval is divided into two sets with 2-second sampling interval, 4 sets with 4-second sampling interval, 8 sets with 8-second sampling interval, and so forth. In total, based on the 10 trips that we have, we can obtain $(1 + 2 + 4 + 8 + 16 + 32 + 64) \times 10 = 1270$ trajectories with various sampling rates.

6.2. *Map matching algorithms*

In order to evaluate the accuracy and efficiency, we implemented two competing algorithms for map matching as described in Newson and Krumm (2009) and Li *et al.* (2014), respectively. In total, we compare four algorithms in our experiments:

- *HMM*: An HMM algorithm was used in Newson and Krumm (2009); the algorithm searches an optimal path in a Markov network of candidate roads by using the Viterbi algorithm.
- *SIMP*: This is an algorithm proposed in Li *et al.* (2014) that simplifies trajectories before mapping them to road networks. It can achieve a higher accuracy than HMM under high noise levels.
- *OBR*: The OBR algorithm builds spatial-linear clusters, maps every cluster to the road networks, and finally connects the clusters. Our insight is that a set of GPS points can give us a better estimation than mapping individual points.
- *OBR-HMM*: The OBR-HMM approach combines the advantages of OBR and HMM together. OBR can achieve well-formed spatial-linear clusters to compute a route close to the ground truth while saving significant computation costs. HMM is good for connecting two clusters. Therefore, we select spatial-linear clusters with smaller error-bounded distances and connect them through HMM.

6.3. Evaluation methodology

The F_1 score is commonly used in evaluating classification problems in data mining, machine learning, and information retrieval. We use the F_1 score to test the accuracy of map matching, which takes into consideration both *precision* and *recall*. In the following formulas, we define that $P = [e_1, ..., e_n]$ and $P' = [e'_1, ..., e'_m]$ are actual and predicted paths where $e_n, e'_m \in E$ and E is the set of all road segments on the road network.

Precision calculates the percentage of the correct length in the estimated path P'. This measure considers the length of erroneously added segments, e.g., if the estimated path P' adds more incorrect segments, $length(P \cap P')$ does not change and $length(P')$ increases leading to a decrease of the precision.

$$Precision = \frac{length(P \cap P')}{length(P')} \tag{8}$$

Recall calculates the percentage of the ground truth P that is covered by the estimated path. This measure considers the length of erroneously subtracted segments, e.g., if the estimated path P' subtracts some correct segments, $length(P \cap P')$ decreases and $length(P)$ keeps the same value leading to a decrease of the recall.

$$Recall = \frac{length(P \cap P')}{length(P)} \tag{9}$$

The F_1 score is the harmonic mean of *precision* and *recall*, and we use the F_1 score to define the error rate:

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{10}$$

$$F_1 \ Error \ Rate = 1 - F_1 \tag{11}$$

where $length(x)$ is the total length of edges x instead of the number of edges, e.g., $length(P')$ is the length of the predicted path P'.

In addition, we use an order-sensitive measure proposed by Zhou and Golledge (2006) in our evaluation to complement the F_1 measure. This measure considers the order of segments, which is ignored by the F_1 measure.

$$Percentage_{\text{matching}} = 100 \times (1 - d_{\text{edit\_distance}}/n) \tag{12}$$

where n is the total number of segments in the actual path and $d_{\text{edit\_distance}}$ is the edit distance between the actual path and the predicted path. The edit distance computes the minimal cost of transforming one path (a sequence of edges) to another by considering three operations, i.e., insertion, deletion, and replacement.

6.4. Results

6.4.1. Synthetic experiments

Map Matching Accuracy. The extensive experiments on synthetic trajectories show that OBR and OBR-HMM significantly improve the accuracy compared to pure HMM and the

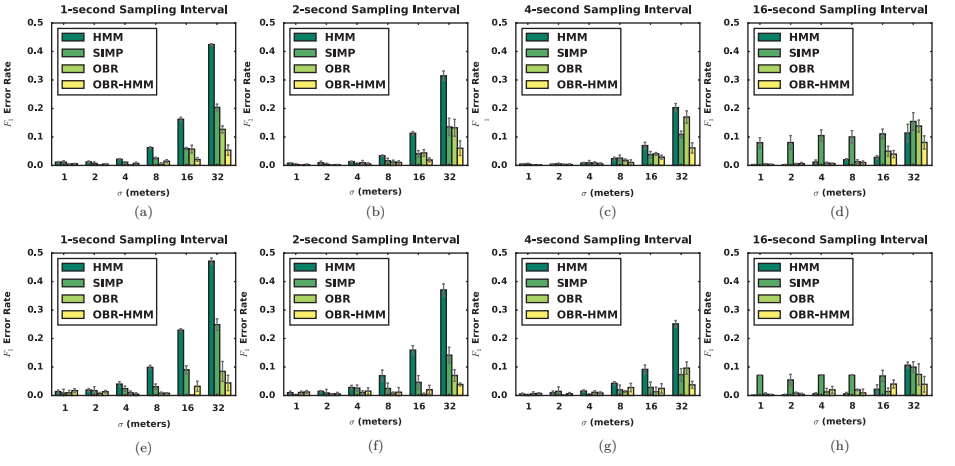


Figure 5. (a)–(d) shows the comparison of map matching accuracy on the synthetic datasets (Seattle) with varying noise levels. (e)–(h) shows the same comparison on synthetic datasets (Melbourne).

simplification approach SIMP. Particularly, when the sampling interval is low and the noise level is high, OBR and OBR-HMM substantially improve the map matching accuracy compared to HMM and SIMP (in Figures 5 and 6). For example, in Figure 5(a), when the sampling interval is 1 second and the σ of noise level is 32 m, HMM has an error rate at approximately 40% and SIMP achieves approximately 20%, but OBR has 10% and OBR-HMM is close to 5%. With the increase of the sampling interval, OBR-HMM still keeps the lowest error rate among all the other approaches (in Figure 7). When the noise level is high, the advantage of OBR-HMM in accuracy is more obvious compared to other methods (see Figure 7(d) and (h)).

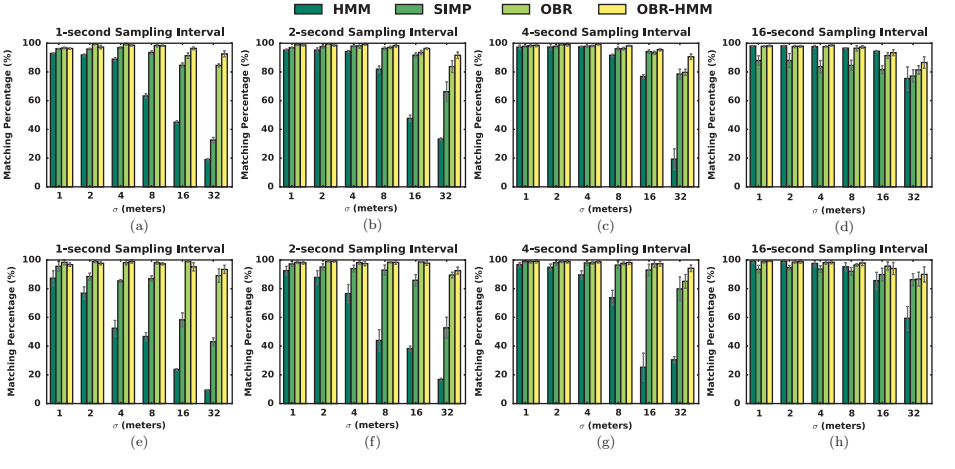


Figure 6. (a)–(d) shows the comparison of matching percentages (%) on the synthetic datasets (Seattle) with varying noise levels. (e)–(h) shows the same comparison on synthetic datasets (Melbourne).

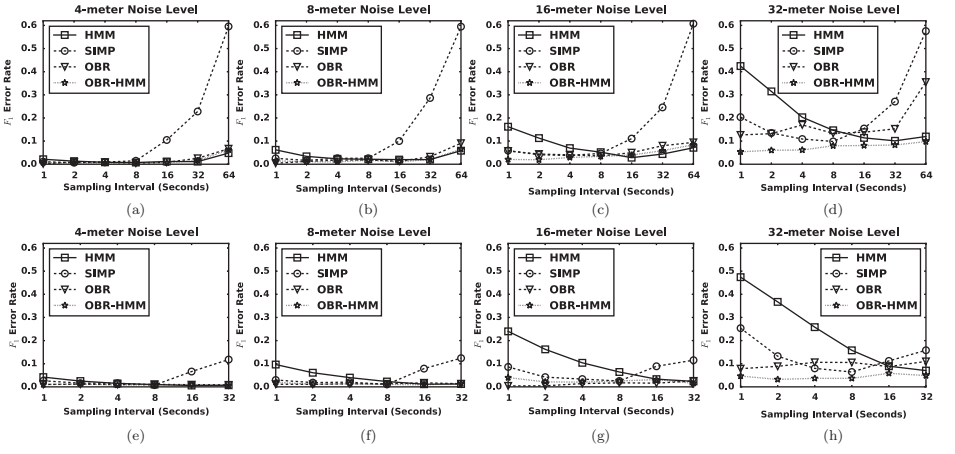


Figure 7. (a)–(d) shows the comparison of map matching accuracy on the synthetic datasets (Seattle) with varying sampling intervals. (e)–(h) shows the same comparison on Melbourne dataset.

The HMM method assumes that each GPS point matters and focuses on individual points. Thus, when sampling is frequent, its error rate is very high. The error rate decreases with higher sampling intervals, because a small number of points can give a better overall direction of traveling path. In Figure 7(d), the error rate of HMM decreases when the trajectory becomes sparse, but the trend stops at the sampling interval of 32 seconds and the error rate starts to increase as not enough information is available for individual point matching. The drawback of focusing on individual points causes a high error rate for high sampling data (see Figures 5 and 6).

In Figure 5, we can see that the SIMP method improves the accuracy significantly in the high sampling rate data compared to HMM. However, a limitation of SIMP is that the error rate goes up when the trajectory data becomes sparse, i.e., lower sampling rate (see Figure 5(d) and (h)). The reason is that over-simplification does not retain enough data. Thus, an adaptive simplification algorithm is needed for sparse trajectory data to avoid over-simplification.

Through building spatial-linear clusters, OBR groups GPS points and maps them together, which does not lose any important information even when the data is sparse. For OBR, when the noise level decreases and the sampling frequency goes up, we can obtain well-formed spatial-linear clusters for GPS points. These clusters are beneficial for the search of traveled paths because the number of potential candidate routes is reduced. With a low noise level, OBR can achieve a high accuracy that is comparable to HMM and SIMP. For high noise levels, OBR exhibits a significant improvement in map matching accuracy (see Figure 5(a) and (e) and Figure 7(d) and (h)). However, when the noise level increases and the width of the rectangle becomes large, spatial-linear clusters are less well-defined leading to higher errors in mapping the main axis to the road. In a complex road network, large GPS errors can occur in urban canyons, intersections, and close parallel roads, which can lead to a poor performance of map matching.

The OBR-HMM combines the strengths of both approaches such that it uses OBR to build well-formed spatial-linear clusters whenever possible and reverts to HMM for the task of connecting the rectangles. When the sampling rate is high, well-formed (thin) rectangles can be successfully established by OBR-HMM. However, when the sampling

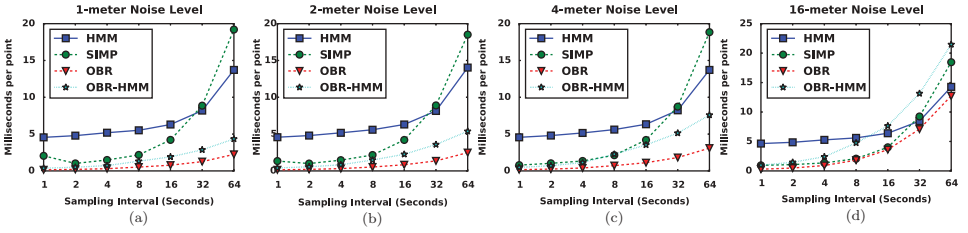


Figure 8. The comparison of runtime per GPS point in milliseconds on the synthetic datasets with different sampling intervals and noise levels.

rate is too low, the rectangle becomes too wide or does not have enough support in terms of the number of GPS points. In such cases, HMM takes over the main job of connecting individual points. OBR filters those individual points that do not matter, while HMM assumes that each GPS point matters. Therefore, these two methods can complement each other. As a result, OBR-HMM outperforms other methods in both high and low sampling rates (see Figure 7(d) and (h)).

Performance. In terms of the performance measure, we use a standardized measure in our experiments. We calculate the runtime per GPS point in milliseconds for different sampling intervals and noise levels. In Figure 8, we can see that OBR is significantly faster than all other methods for high sampling intervals. For the frequent sampling data, OBR can successfully form spatial-linear clusters as there is enough support in terms of the number of GPS points. We map spatial-linear clusters together leading to the significant speed-up compared to HMM. In contrast, HMM deals with individual points separately, which results in a slow execution time. Moreover, OBR has a better performance in execution time compared to SIMP. In terms of execution time, OBR-HMM is a bit slower than the pure OBR method, but still shows a significant speed-up compared to HMM.

6.4.2. *Real dataset experiments*

Accuracy. OBR-HMM shows a significant improvement in accuracy than other methods on the real dataset (in Figure 9(a) and (b)). In the high-sampling rate data, OBR-HMM maintains an extremely low error rate approximately 1% and a high matching percentage

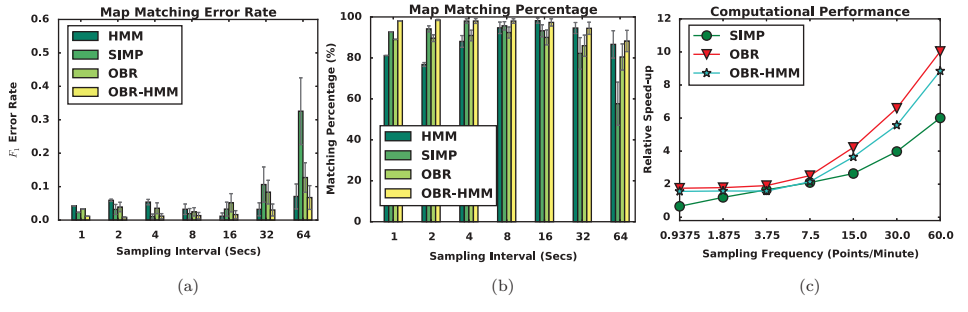


Figure 9. The comparison of accuracy and computational performance on the real dataset (GIS Contest 2012).

of approximately 98% for sampling intervals of 1–8 seconds. Under these conditions, OBR-HMM outperforms all other methods considerably. The pure HMM method does not perform well for the high-sampling rate dataset, but achieves better results for sparse datasets, e.g., in 16-second sampling interval, its error rate is only approximately 1%. SIMP shows an inferior performance with more than 30% error rate on very low-sampling rate datasets, although it performs well in high-sampling rate datasets (which it was designed for). Furthermore, pure OBR outperforms HMM in high-sampling rate datasets. However, OBR does not perform well in low-sampling rate datasets; because of the low number of GPS points, it lacks enough supports to form spatial-linear clusters. OBR considers a small section (cluster) and repairs the errors locally, but raw HMM is better under these conditions. Therefore, a combination of OBR-HMM demonstrates a good overall performance for both high-sampling and low-sampling rate datasets.

Relative Speed-up. Due to the different sizes between trips, we evaluate the computational performance of algorithms by calculating the ratio between their execution time and HMM's execution time:

$$Speedup_{\text{relative}} = \frac{ExecutionTime_{\text{HMM}}}{ExecutionTime_{\text{algorithm}}} \tag{13}$$

Compared to HMM, all methods exhibit significant speed-ups on execution time in Figure 9(c). Specifically, for high-sampling rate data (60 points/minute), OBR can achieve a significant speed-up of approximately 10 times that of the raw HMM. The speeding up of OBR-HMM is a little slower than the pure OBR, because the search for the shortest paths is expensive in terms of execution time. OBR-HMM outperforms SIMP in all the tests. Furthermore, SIMP is slower than the pure HMM in the very low-sampling rate dataset (0.9375 points/min), where the speed-up ratio is less than 1.0. In the sparse dataset, OBR can still run 2 times faster than pure HMM. Overall, our proposed methods achieve a significant improvement compared to the existing methods HMM and SIMP in terms of computational performance.

Error-Bounded Distance. We compare OBR with different error-bounded distances on the real dataset (Figure 10). It is clear that a large value for d_{error} increases the error rate because spatial-linear clusters cannot be well formed. With a small value for d_{error}, we can

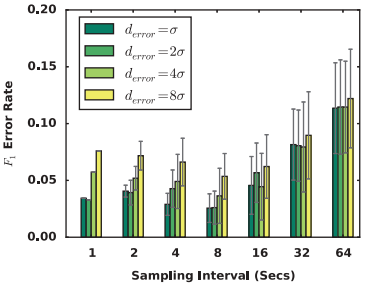


Figure 10. The accuracy of applying OBR with varying the error-bounded distance d_{error} of clusters on the real dataset (GIS Contest 2012).

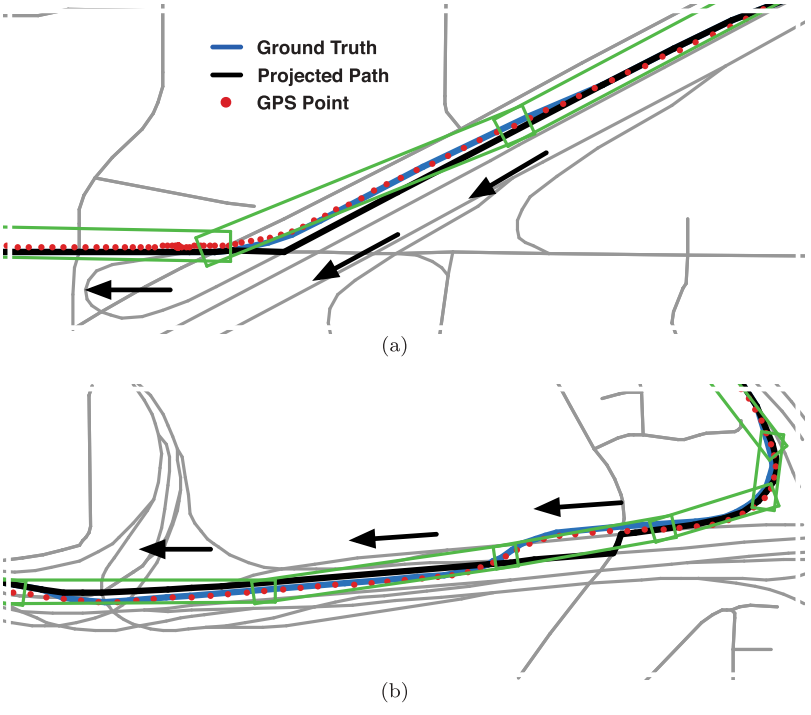


Figure 11. Examples of errors caused by alternative parallel roads on a complex road network.

achieve a better map matching accuracy. However, when d_{error} is too small, the algorithm builds too many small spatial-linear clusters in order to fit the shape (see Figure 3(d)).

Error Analysis. Figure 11 shows two examples that our algorithm may not be able to give accurate estimations. The mismatches between the estimated and the actual trajectories are caused by the alternative parallel roads in complex road networks. When parallel roads are very close to each other, certain GPS points may be mapped to the wrong roads instead of the actual traveled path. Therefore, when we search the potential routes overlapping with GPS point clusters, two parallel roads may have an equal probability to be selected. This (unavoidable) problem is a result of the complicated road networks and poor GPS signals in urban canyons.

6.4.3. *Summary*

Table 2 shows a comprehensive comparison of the investigated map matching algorithms under different conditions, e.g., high sampling rate data with high noise level. In the table, we define that high sampling rates are 1, 2, 4, and 8 seconds and low sampling rates are 16, 32, and 64 seconds. In addition, noise levels are divided into two groups: high noise levels ($\sigma = 8, 16, 32$ m) and low noise levels ($\sigma = 1, 2, 4$ m).

Overall, our proposed algorithms OBR and OBR-HMM outperform SIMP and HMM as shown in the table. OBR-HMM achieves very low error rates, ranked either in the first place or second place, under all conditions with significant improvements in running time. OBR significantly outperforms other methods on high sampling data rate, especially

Table 2. Comparison and ranking of map matching algorithms under different conditions in terms of error rate and relative speed-up.

	High sampling rate				Low sampling rate			
	High noise		Low noise		High noise		Low noise	
Algorithms	Err(%)[a]	Rlt-spd[b]	Err(%)	Rlt-spd	Err(%)	Rlt-spd	Err(%)	Rlt-spd
OBR-HMM	**3.48**(1)	2.80(3)	0.50(2)	7.80(2)	**6.12**(1)	0.74(4)	2.17(2)	2.33(2)
OBR	6.49(2)	**6.10**(1)	**0.38**(1)	**21.06**(1)	10.55(3)	0.82(3)	2.70(3)	**6.02**(1)
SIMP	7.61(3)	4.40(2)	0.88(3)	3.54(3)	33.24(4)	**1.07**(1)	31.24(4)	1.04(3)
HMM	13.84(4)	1.00(4)	1.31(4)	1.00(4)	6.29(2)	1.00(2)	**2.10**(1)	1.00(4)

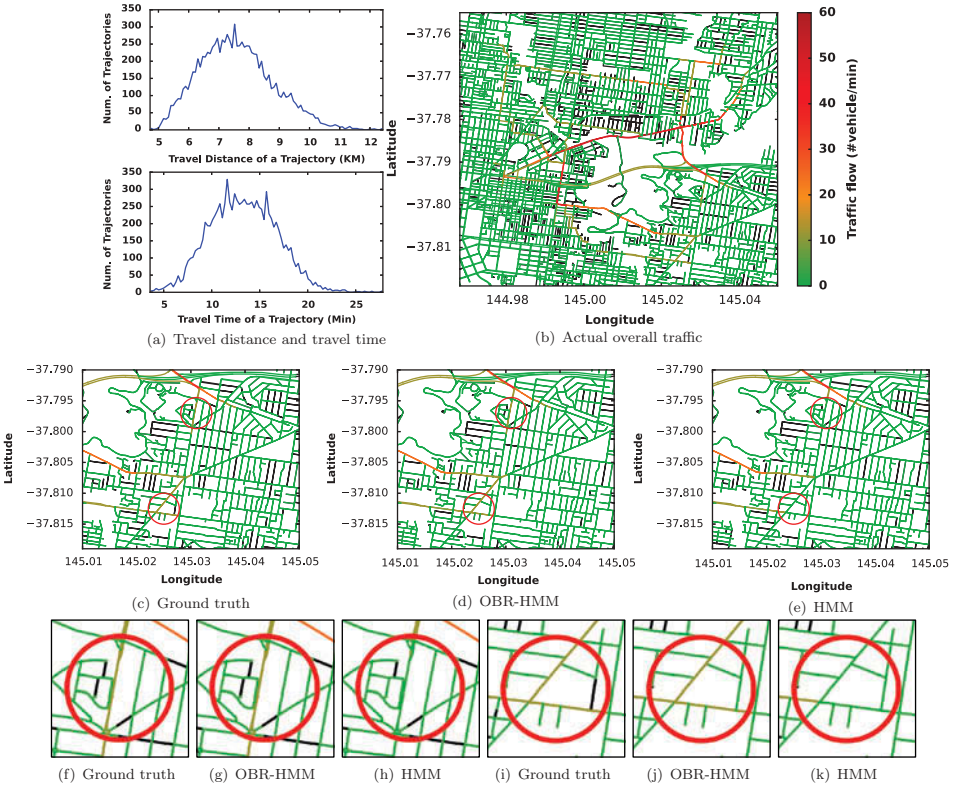
Notes: Bold values ranked as number one. [a]Err(%) refers to F_1 error rate (see Equation (11)) in which a small value means a high accuracy in map matching.
[b]Rlt-spd represents the relative speed-up in Equation (13) that computes a ratio comparing to the runtime of HMM. Note that each pair in the table corresponds to the error rate and its rank or the relative speed-up and its rank.

under low noise levels, its running time is 21 times faster than HMM. However, OBR loses its advantage for low sampling data rates because it lacks sufficient support in terms of the number of points to generate a cluster. Similar to OBR, SIMP shows improvements in both error rate and running time for high sampling data rates, but SIMP – due to a constant compression ratio – leads to high error rates for low sampling data rates. On the other hand, HMM performs very well on sparse data (low sampling rate) – 6.29% at high noise level and 2.10% at low noise level – but it has a poor performance on rich data (high sampling rate) where HMM ranks the lowest.

6.5. *A case study*

We conduct a case study to analyze the impact of map matching for the estimation of traffic flow of a large area. For the experiment we choose an 8 km × 8 km area in Melbourne and generate 10,000 travel paths using its road map as ground truth. Therefore, we know the traffic flow of each road segment. The origin–destination pair of each path is randomly chosen from the nodes of the map and their distance is at least 4 km (more than half of the width). To simulate a real scenario, we sample points along the generated travel path according to a time interval (5 seconds in our case study) using the maximum speed of the visited road. In addition, we add noise of $\sigma = 5$ m (according to the GPS performance analysis mentioned in Section 1) to the sample points. Figure 12 (a) shows the distribution of travel distance and travel time of the generated trajectories. The travel distance ranges from approximately 5 to 12 km, and its mean and standard deviation are 7.58 and 1.21 km. The travel time of trajectories ranges from under 5 minutes to over 25 minutes, and its mean and standard deviation are 13.28 and 3.43 minutes, respectively. We assume that all trajectories are collected within 30 minutes; therefore, we can estimate the traffic flow for each road segment, which ranges from 0 to under 60 vehicles per minute as shown in Figure 12(b).

Through the reconstruction of traffic flow using map matching, we can compare the estimate to the actual traffic flow and evaluate the accuracy of each algorithm. We conduct a quantitative analysis by checking the traffic flow (represented by different colors) of same road segments. For example, Figure 12(c)–(e) compare the results of OBR-HMM and HMM to the ground truth in a small area. The red circles point out the differences



(a) Travel distance and travel time (b) Actual overall traffic

(c) Ground truth (d) OBR-HMM (e) HMM

(f) Ground truth (g) OBR-HMM (h) HMM (i) Ground truth (j) OBR-HMM (k) HMM

Figure 12. (a) The distribution of travel distance and travel time of 10,000 generated noisy trajectories. (b) An overall view of traffic flow of the ground truth (black color means no visited vehicles). (c)–(e) Comparison of OBR-HMM and HMM to the ground truth on a small area (red circles highlight obvious differences). (f)–(h) and (i)–(k) show the details of two highlight circles, respectively.

between OBR-HMM and HMM. The traffic flow of three road segments are underestimated by the HMM algorithm as green color (≤ 5 vehicles/min) in the two red circles. However, the results of OBR-HMM algorithm are consistent with their true values of yellow color (10–15 vehicles/min).

To compare the ground truth and the estimated traffic flow more precisely, we compute the mean absolute error (MAE), mean relative error (MRE), and root mean square error (RMSE):

$$e_{\mathrm{MAE}} = \frac{\sum_{i=1}^{n} |p_i - t_i|}{n} \tag{14}$$

$$e_{\mathrm{MRE}} = \frac{\sum_{i=1}^{n} |p_i - t_i|}{\sum_{i=1}^{n} t_i} \tag{15}$$

Table 3. Comparison of map matching algorithms in terms of estimated traffic flow based on three measures: mean absolute error (MAE), mean relative error (MRE), and root mean square error (RMSE).

Algorithms	MAE	MRE	RMSE
OBR-HMM	0.050	0.025	0.179
OBR	0.081	0.041	0.248
SIMP	0.147	0.075	0.691
HMM	0.233	0.119	1.150

$$e_{\mathrm{RMSE}} = \sqrt{\frac{\sum_{i=1}^{n} (p_i - t_i)^2}{n}} \tag{16}$$

where n is the number of road segments, p_i is the estimated traffic flow for the road segment i, and its true value is t_i.

Table 3 shows the accuracy of estimated traffic flow according to the results of different map matching algorithms. OBR-HMM and OBR clearly outperform the other algorithms SIMP and HMM Especially, OBR-HMM improves the accuracy approximately five times compared to HMM. This case study shows that inferring the accurate travel path from a noisy trajectory is very important to disclose the true traffic flow of road networks.

7. Conclusions and future work

In this paper, we proposed a spatial-linear clustering algorithm (OBR) that achieves better accuracy and execution time than traditional approaches in mapping GPS points to road networks. The spatial-linear cluster approach can allow us to ignore outliers and provides significant speed-ups in terms of execution time. Even under less favorable conditions, our algorithm can provide a clear macro view of a GPS trajectory. Our experiments show that the combination of OBR and HMM outperforms the state-of-the-art methods in accuracy with a significant speed-up gain in execution time.

We highlight two issues for a map matching algorithm: missing GPS points and missing road segments. The first issue is how to deal with missing GPS points in a trajectory, e.g., due to the unavailability of signals in tunnels and underpasses. If there are no GPS points to support our algorithm, we have to rely on the partial information given by predecessors and successors of that gap. In our work, we assume that a vehicle takes the shortest path during such a gap of missing GPS points. This is generally a reasonable assumption because if the vehicle goes through a tunnel, there are not many (if any) alternative routes. Usually, the tunnel is the shortest path. In this paper, we assume that the underlying road network has no missing road segments. If there is a missing road segment, building the ground truth becomes a hard problem. We will investigate this problem further in future work.

Two possible future directions that result from this research are clusters of different geometries and the use of domain knowledge.

Clusters of different geometries. In our proposed algorithm, spatial-linear clusters are defined to use only one type of geometry, i.e., rectangles. In addition to rectangles, we

will look for other possible clusters of different geometries in order to better characterize a trajectory. We will test whether other non-linear geometries, such as L-shape for modeling right angle turns, can achieve better results in performance.

The use of domain knowledge. We use a spatial-linear clustering approach in our map matching procedure. OBR is a pure geometry-based method and does not rely on any domain knowledge of a road network. Therefore, errors can occur when the algorithm has to disambiguate between two parallel roads. We aim to investigate in the future whether the use of domain knowledge, such as road types, speed limits, and direction (GPS time component), can help further reduce such errors and improve accuracy.

Disclosure statement

No potential conflict of interest was reported by the authors.

Notes

1. 'FAA GPS Performance Analysis Report' published in July 2014, available from http://www.nstb.tc.faa.gov/reports/PAN86_0714.pdf#page=22, accessed 7 June 2015.
2. http://people.eng.unimelb.edu.au/henli/projects/robust-inferences-of-travel-paths/
3. (http://www.openstreetmap.org)

References

Ali, M., *et al*., 2012. ACM SIGSPATIAL GIS Cup 2012. *In*: *Proceedings of the 20th International Conference on Advances in Geographic Information Systems, SIGSPATIAL '12, Redondo Beach, California*. New York: ACM, 597–600.

Alt, H. and Godau, M., 1995. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5, 75–91. doi:10.1142/S0218195995000064

Berndt, D.J. and Clifford, J., 1994. Using dynamic time warping to find patterns in time series. *In*: U.M. Fayyad and R. Uthurusamy, eds. *KDD workshop*. Seattle, WA: AAAI Press, 359–370.

Bierlaire, M., Chen, J., and Newman, J., 2013. A probabilistic map matching method for smartphone GPS data. *Transportation Research Part C: Emerging Technologies*, 26 (0), 78–98. doi:10.1016/j.trc.2012.08.001

Brakatsoulas, S., *et al*., 2005. On map-matching vehicle tracking data. *In*: K. Böhm, *et al*., eds. *Proceedings of the 31st international conference on very large data bases, Trondheim, Norway, August 30–September 2, 2005*. New York: ACM, 853–864.

Chen, B.Y., *et al*., 2014. Map-matching algorithm for large-scale low-frequency floating car data. *International Journal of Geographical Information Science*, 28 (1), 22–38. doi:10.1080/13658816.2013.816427

Chen, L., Özsu, M.T., and Oria, V., 2005. Robust and fast similarity search for moving object trajectories. *In*: *Proceedings of the* 2005 ACM *SIGMOD International Conference on Management of Data*, SIGMOD '05, Baltimore, Maryland. New York: ACM, 491–502.

Forney, G.D.J., 1973. The viterbi algorithm. *Proceedings of the IEEE*, 61 (3), 268–278. doi:10.1109/PROC.1973.9030

Gelb, A., 1974. *Applied optimal estimation*. Cambridge, MA: MIT Press.

Greenfeld, J.S., 2002. Matching GPS observations to locations on a digital map. *In*: *81th annual meeting of the transportation research board*, January, Washington, DC.

Hightower, J. and Borriello, G., 2004. Particle filters for location estimation in ubiquitous computing: a case study. *In*: N. Davies, E. Mynatt, and I. Siio, eds. *UbiComp 2004: ubiquitous computing, Vol. 3205 of lecture notes in computer science*. Berlin, Heidelberg: Springer, 88–106.

Huttenlocher, D.P., Klanderman, G.A., and Rucklidge, W.J., 1993. Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15 (9), 850–863. doi:10.1109/34.232073

Krumm, J., 2008. A Markov model for driver turn prediction. Technical report, SAE Technical Paper.

Li, H., Kulik, L., and Ramamohanarao, K., 2014. Spatio-temporal trajectory simplification for inferring travel paths. *In*: *Proceedings of the 22nd* ACM *SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '14, Dallas/Fort Worth, TX, USA. New York: ACM.

Li, Y., *et al*., 2013. Large-scale joint map matching of GPS traces. *In*: C.A. Knoblock, *et al*., *et al*., eds. *21st SIGSPATIAL international conference on advances in geographic information systems, SIGSPATIAL 2013*. Orlando, FL: ACM, 214–223.

Liao, L., Fox, D., and Kautz, H.A., 2007. Extracting places and activities from GPS traces using hierarchical conditional random fields. *The International Journal of Robotics Research*, 26 (1), 119–134. doi:10.1177/0278364907073775

Lou, Y., *et al*., 2009. Map-matching for low-sampling-rate GPS trajectories. *In*: *Proceedings of the 17th* ACM *SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '09, Seattle, Washington. New York: ACM, 352–361.

Marchal, F., Hackney, J., and Axhausen, K.W., 2005. Efficient map matching of large global positioning system data sets: tests on speed-monitoring experiment in Zürich. *Transportation Research Record: Journal of the Transportation Research Board*, 1935 (1), 93–100. doi:10.3141/1935-11

Newson, P. and Krumm, J., 2009. Hidden Markov map matching through noise and sparseness. *In*: *Proceedings of the 17th* ACM *SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '09, Seattle, Washington. New York: ACM, 336–343.

Ochieng, W.Y., Quddus, M., and Noland, R.B., 2003. Map-matching in complex urban road networks. *Revista Brasileira de Cartografia*, 2 (55), 1–14.

Potamias, M., Patroumpas, K., and Sellis, T.K., 2006. Sampling trajectory streams with spatiotemporal criteria. *In*: *Proceedings of 18th international conference on scientific and statistical database management*, Vienna. Washington, DC: IEEE Computer Society, 275–284.

Quddus, M., *et al*., 2003. A general map matching algorithm for transport telematics applications. *GPS Solutions*, 7 (3), 157–167. doi:10.1007/s10291-003-0069-z

Velaga, N.R., Quddus, M.A., and Bristow, A.L., 2009. Developing an enhanced weight-based topological map-matching algorithm for intelligent transport systems. *Transportation Research Part C: Emerging Technologies*, 17 (6), 672–683. doi:10.1016/j.trc.2009.05.008

Vlachos, M., Gunopulos, D., and Kollios, G., 2002. Discovering similar multidimensional trajectories. *In*: R. Agrawal and K.R. Dittrich, eds. *Proceedings of the 18th international conference on data engineering*, San Jose, CA. Washington, DC: IEEE Computer Society, 673–684.

Weber, M., *et al*., 2010. On map matching of wireless positioning data: a selective look-ahead approach. *In*: *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, San Jose, California. New York: ACM, 290–299.

White, C.E., Bernstein, D., and Kornhauser, A.L., 2000. Some map matching algorithms for personal navigation assistants. *Transportation Research Part C: Emerging Technologies*, 8 (1–6), 91–108. doi:10.1016/S0968-090X(00)00026-7

Yin, H. and Wolfson, O., 2004. A weight-based map matching method in moving objects databases. *In*: *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM* 2004), 21–23 June 2004. Santorini Island, Greece: IEEE Computer Society, 437–438.

Yuan, J., *et al*., 2010a. T-drive: driving directions based on taxi trajectories. *In*: *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, San Jose, California. New York: ACM, 99–108.

Yuan, J., *et al*., 2010b. An interactive-voting based map matching algorithm. *In*: T. Hara, *et al*., eds. *Eleventh international conference on mobile data management, MDM 2010*, Kansas City. Missouri: IEEE Computer Society, 43–52.

Yuan, J., *et al*., 2011. Driving with knowledge from the physical world. In: C. Apté, J. Ghosh, and P. Smyth, *et al*., eds. *Proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining*. San Diego, CA: ACM, 316–324.

Zheng, Y. and Zhou, X., 2011. *Computing with spatial trajectories*. Berlin: Springer.

Zhou, J. and Golledge, R., 2006. A three-step general map matching method in the GIS environment: travel/transportation study perspective. *International Journal of Geographical Information System*, 8 (3), 243–260.