Spatio-Temporal Trajectory Simplification for Inferring Travel Paths

Hengfeng Li, Lars Kulik, Kotagiri Ramamohanarao
Department of Computing and Information Systems
The University of Melbourne
Victoria 3010, Australia
{hengfeng.li, lkulik, kotagiri}@unimelb.edu.au

ABSTRACT

Mining GPS trajectories of moving vehicles has led to many research directions, such as traffic modeling and driving predication. An important challenge is how to map GPS traces to a road network accurately under noisy conditions. However, to the best of our knowledge, there is no existing work that first simplifies a trajectory to improve map matching. In this paper we propose three trajectory simplification algorithms that can deal with both offline and online trajectory data. We use weighting functions to incorporate spatial knowledge, such as segment lengths and turning angles, into our simplification algorithms. In addition, we measure the noise degree of a GPS point based on its spatio-temporal relationship to its neighbors. The effectiveness of our algorithms is comprehensively evaluated on real trajectory datasets with varying the noise levels and sampling rates. Our evaluation shows that under highly noisy conditions, our proposed algorithms considerably improve map matching accuracy and reduce computational costs compared to the state-of-the-art methods.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications–*Spatial databases and GIS*

General Terms

Algorithms, Performance

Keywords

trajectory simplification; map matching; GPS trace; path inference

1. INTRODUCTION

A GPS trajectory is a temporal sequence of observed locations. Each data point consists of a coordinate, i.e., latitude and longitude, and a corresponding timestamp. The noise in GPS measurements leads to uncertain GPS locations, which makes it difficult to locate the true visited path in a road network. Map matching is a technique that aims to deal with this issue through mapping GPS points to a road network.

SIGSPATIAL '14 November 04 - 07 2014, Dallas/Fort Worth, TX, USA

ACM 978-1-4503-3131-9/14/11...$15.00.
http://dx.doi.org/10.1145/2666310.2666409.

Challenges

In order to successfully infer an accurate path, the following challenges need to be addressed:

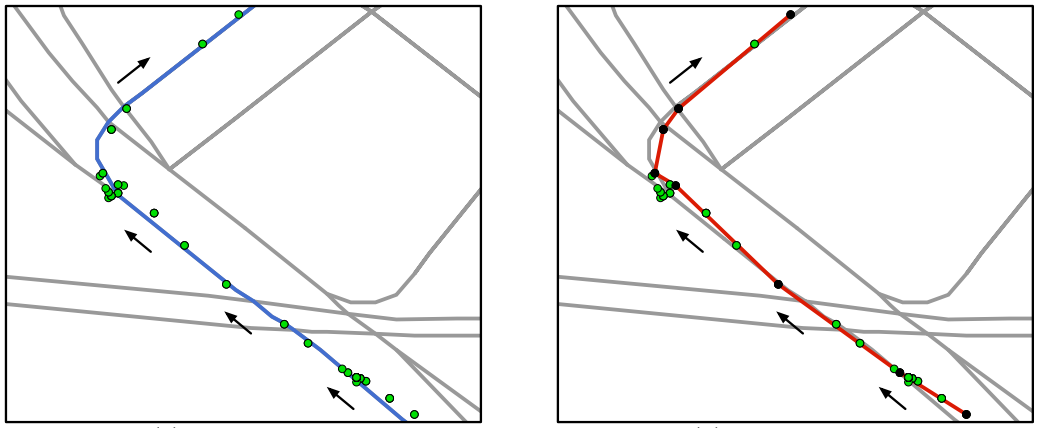
- *Noisy GPS points:* Noise in GPS measurement causes the uncertainty of recorded locations. The coordinates of points deviate from their true position on the roads. Thus, it is difficult to identify to which road a GPS point should be mapped.
- *Stop points:* If vehicles get stuck in traffic jams or have to stop at intersections, GPS points are still recorded. Consequently, GPS points are randomly distributed around the stop point due to the general GPS noise. Stop points can cause significant errors in map matching algorithms.
- *Variable road density:* In some regions, roads are fairly well distributed. However, in urban areas, roads are densely placed due to the space limitations. For example, parallel roads can pose significant challenges to mapping a GPS trace to correct roads when the trace is exactly located in the middle of two parallel roads.

Our research is guided by the following observations: (i) Many vehicles are equipped with on-board GPS devices leading to a large amount of GPS data. (ii) A GPS trajectory is a temporal sequence of spatial coordinates. The question of how to accurately match GPS trajectories to a road network is a fundamental spatial problem. (iii) We can use spatial knowledge, such as the segment lengths of trajectory parts or the turning angles between two straight line segments of the trajectory, to simplify a GPS trajectory to remove noisy points and stop points.

Contributions

In this paper, we use trajectory simplification to enhance map matching to obtain a higher accuracy and a lower computational costs.

Our insight that simplifying a trajectory can remove noisy points and stop points is demonstrated by the example shown in Figure 1. According to Figure 1(a), the GPS noise causes GPS measurements that are not accurate. In addition, stop



(a) Ground truth (b) Simplified trace

Figure 1: (a) A noisy GPS trajectory (green points) and its true path (dark blue) on roads (light grey). The observed locations from GPS points are uncertain. Stop points can incur clusters of GPS points. (b) A simplified GPS trajectory (8 points, black point and red line). It is a subset of original noisy data (87 points, green), which is more similar to its ground truth.

points are often prone to oversampling. Figure 1(b) shows that a simplified trajectory reduces noisy points and removes oversampling at stop points. Although a reduced sampling of GPS points would also lead to a simplified trajectory, there is less GPS data available, which could exacerbate the error in map matching.

More specifically, our **contributions** in this paper are:

1. **Three simplification algorithms:** we propose Incremental Simplification (IS) that simplifies a trajectory point-by-point by maintaining an incremental window, Sliding Window Simplification (SWS) that keeps a fixed size of window moving forward with an increasing number of points, and Global Simplification (GS) that considers the entire trajectory while reducing the number of GPS points, to deal with both offline and online trajectory data.

2. **Use of weighting functions in simplification:** we use different weighting functions that incorporate spatial knowledge into the trajectory simplification process. The functions measure the noise degree of a GPS point based on the spatio-temporal relationship to its neighbors.

3. **Comprehensive experiments on real data:** We evaluate our algorithms on two real datasets – Seattle and Melbourne. Through variation of the noise levels and sampling rates, the algorithms are evaluated under less favorably conditions. In addition to using existing datasets, we collect our own GPS data to ensure ground truth, which we aim to make available for other researchers.

The rest of the paper is organized as follows: Section 2 reviews related work and Section 3 formally presents preliminary definitions and defines the problem. Section 4 describes our proposed simplification algorithms and Section 5 discusses particulars of our algorithms. The experimental study is given in Section 6 and Section 7 outlines future research directions.

2. RELATED WORK

Trajectory Simplification. The Douglas Peucker (DP) algorithm [6] is a well-known line simplification approach that has been applied to the problem of spatio-temporal data reduction in mobile phones or PDAs datasets [4]. In addition, a variant of DP for moving object trajectories [12] takes into account the travel time difference and derived speeds rather than only calculating perpendicular distance. Other DP variants adopt strategies, like approximating the distance from a point to a line segment [7] and choosing a point close to the middle of a trajectory [8].

In addition to DP, a semantic simplification algorithm was proposed in [5], which uses location semantics, e.g., landmarks, to determine important points of a trajectory. Further, in [14] the distance function is similar to the perpendicular distance based on projection of points, but it takes speed into consideration. A simplification algorithm was given in [10], which chooses *characteristic points* from a trajectory according to the defined cost function. A trajectory compression framework PRESS [15] maps trajectories into sequences of edges and compresses these sequences based on spatial path and temporal information.

Map Matching. Most proposed methods use geometric knowledge to address the problem. Four map matching algorithms were introduced in [17], finding a closest road, adding orientation, considering topology, and calculating the similarity between trajectories and road segments. In addition, authors in [3] proposed two approaches, the incremental algorithm based on perpendicular distance and turning angle and the global matching algorithm minimizing the Fréchet distance of two curves. Another branch models the problem as a probability prediction. A representative approach is Hidden Markov Model (HMM) map matching [13], which assumes that GPS measurement errors is under a zero-mean Gaussian distribution and the ratio of comparing Euclidean distance of two consecutive points to their route distance is under an exponential distribution. The Viterbi algorithm is applied to search an optimal path. Some similar research using HMM has been done in [11] and [18].

3. PROBLEM STATEMENT

In this section, we present preliminary definitions that are commonly used in the path inference. Based on these definitions, we describe our approach to the simplification problem. We also present the basic methodology to evaluate different simplification strategies to improve map matching accuracy. Lastly, we give an overview of our path inference system.

3.1 Preliminary Definitions

DEFINITION 1. (ROAD NETWORK). *$G(V, E)$ is a graph of road network. Vertices V represent intersections or end points in a road network and edges E are the directed road segments.*

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

DEFINITION 2. (ORIGINAL PATH). *The original path P is a sequence of road segments to represent the route of a vehicle's movement, $P = [e_{p_1}, e_{p_2}, ..., e_{p_N}]$, 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. (RAW TRAJECTORY). *The raw trajectory T consists of measured GPS points along the true path P during a fixed or varied time intervals. A trajectory function f_T can be defined as:*

$$for\ t \in [t_1, t_2, ..., t_n]: \qquad t \mapsto f_T(t) \in \mathbb{R}^2$$

Each GPS point has its latitude, longitude, and measured timestamp. Due to GPS measurement noise, the location of each point is uncertain.

DEFINITION 4. (SIMPLIFIED TRAJECTORY). *Simplified trajectory T_s is a subset of T. T_s contains fewer or equal number of GPS points after removing noisy or stop points from T. A simplified trajectory function f_{T_s} can be defined as:*

$$for\ t \in [t_1, t_2, ..., t_m], m \leq n: \qquad t \mapsto f_{T_s}(t) \in \mathbb{R}^2$$

The aim is to obtain a simplified trajectory T_s that can be used for reconstructing a traveled path. This reconstructed path is highly similar to its original (ground truth) path P, after cleaning those noisy points and less important points.

DEFINITION 5. (MAP MATCHING). *Map matching is a process to map a raw GPS trajectory T to a sequence of road segments S. The process can be defined as a function Map:*

$$S = Map(T)$$

where $T = [p_{t_1}, p_{t_2}, ..., p_{t_n}]$, $S = [e_{t_1}, e_{t_2}, ..., e_{t_n}]$, and $e_{t_n} \in E$.

DEFINITION 6. (PATH INFERENCE). *Path inference computes a complete and fully-connected path through removing duplicates and interpolating missing edges in S. The process can be defined as a function $Path$:*

$$P' = Path(S)$$

where $P' = [e_{p'_1}, e_{p'_2}, ..., e_{p'_M}]$ and $e_{p'_M} \in E$.

DEFINITION 7. (PATH DISTANCE MEASURE). *The path distance measure $Dist(P_1, P_2)$ calculates the similarity of two paths, P_1 and P_2. A smaller distance between two paths refers to a higher similarity.*

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

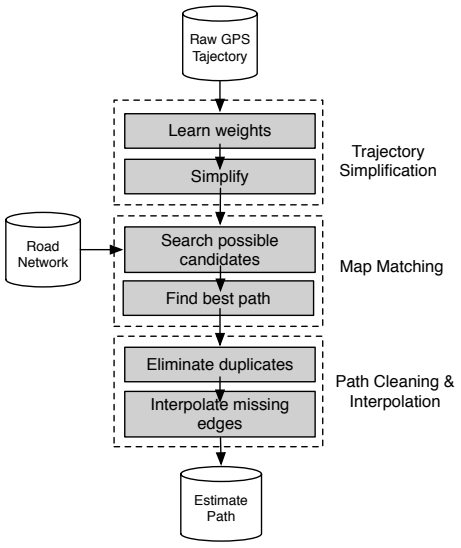


Figure 2: An overview of path inference system for a raw GPS trajectory.

3.2 Problem Description

Our goal is to estimate a path P' from a raw GPS trajectory T which is as similar as possible to its actual path P. Due to GPS measurement noise, collected GPS points have certain deviations to their actual positions on the true path P. Such uncertain locations may lead to many errors in map matching. To address the above problem, a simplification process can be applied to the raw trajectory T to remove noisy and stop points, which leads to a simplified trajectory T'.

We assume that P'_T and $P'_{T'}$ are the estimated paths of the raw data T and simplified data T' using the path inference process. We also use the distance measure $Dist(A, B)$ to compute the similarity between the trajectory A and B. The lower value of $Dist(A, B)$, the more similar A and B are. If $Dist(P'_T, P) > Dist(P'_{T'}, P)$ exists, it means that $P'_{T'}$ is more similar to P than P'_T, which implies that the simplified trajectory improves the map matching accuracy. Next, we assume that P'_1 and P'_2 are the estimated results using two different simplification approaches M_1 and M_2, respectively. If $Dist(P'_1, P) > Dist(P'_2, P)$ exists, M_2 is better than M_1.

Overall, we want to validate that the simplification process improves the performance of path inference and also our proposed algorithms outperform the state-of-the-art methods.

3.3 System Overview

The system has three components: *trajectory simplification*, *map matching*, and *path cleaning & interpolating* (Figure 2). It needs two inputs, raw GPS trajectories and the graph of a road network. Firstly, we assign a value (a weight) to each GPS point, which indicates the importance of the point to the GPS trajectory. We predefine a weighting function using spatial knowledge such as the segment lengths of trajectory parts or the turning angles between two straight line segments of the trajectory. Then, based on the weights, we gradually remove noisy points from the original trace. Removal of noisy points and stop points requires to update the weights of their neighbors. A map matching algorithm

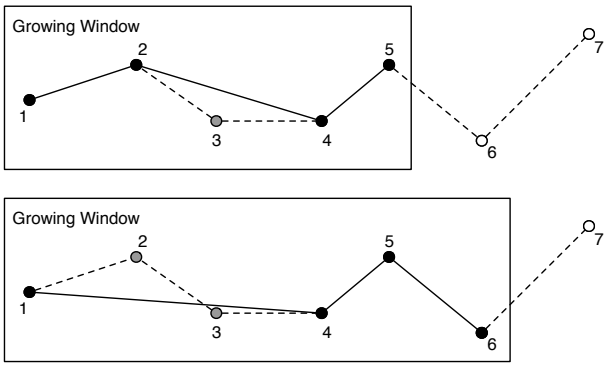


Figure 3: An example of a simplifying update of the incremental algorithm.

is applied to search for an optimal estimated path for the whole GPS trajectory on the road network. Finally, to reconstruct a complete footprint, we need to eliminate duplicates and add missing edges along the estimated path. In this paper, we focus on how to simplify a trajectory to remove noisy GPS points and to improve the performance of path inference.

4. TRAJECTORY SIMPLIFICATION

In this section, we propose three algorithms: *incremental simplification*, *sliding window simplification*, and *global simplification*. The general principle is that the weight of each point indicates its importance in the entire trajectory. We define the weight by combining a geometric property w_g and reliability w_r (indicating the noise degree), see Section 5. When a point is removed, its neighbor's weights need to be updated. Finally, when the size of a simplified trajectory equals to the number calculated by a given compression ratio, the simplification algorithm terminates.

4.1 Incremental Simplification

The main idea of *incremental simplification* is to use a growing window approach for the simplified trajectory: the algorithm removes points within this window. The algorithm starts with the first two GPS points. Then, it incrementally scans the remaining points adding a point to its existing simplified trajectory T' (Algorithm 1). Next, the algorithm updates the weights of the neighbor points for this new point. A decision is made whether or not to delete a point from the simplified trajectory. This is determined by the number of points that the simplified trajectory should keep, which in turn is specified by the compression ratio r. For example, if there are 100 points in the original trajectory and the compression ratio r is 70% then 70 points are to be removed in the final simplified trajectory.

In the *incremental simplification*, the size of existing simplified trajectory T' is based on the number of scanned points and their compression ratio r. If the size of T' exceeds the number of points to be kept, the algorithm removes a point from the simplified trajectory T'. Then, a point with the minimum weight is removed from T' and all of its neighbors weights are updated. Through processing such GPS point in turn and applying our proposed simplification method, a simplified GPS trajectory with a specific compression ratio can be achieved.

Algorithm 1: Incremental Trajectory Simplification

Input : $T = [p_{t_1}, p_{t_2}, ..., p_{t_N}]$ is a raw GPS trajectory that consists of measured GPS points and a compression rate r_{comp} also is given.
Output: T' is a subset of T, which is a simplified trajectory from T.

```
begin
    // Initialize the simplified trajectory
    T' = [p_{t_1}, p_{t_2}]
    numVisited ⟵ 2
    for p ∈ [p_{t_3}, p_{t_4}, ..., p_{t_N}] do
        Append p to T'
        Calculate weights for the new triangle
        numVisited ⟵ numVisited + 1
        // Calculate the number of points to be kept
        numKeep ⟵ numVisited ×(1 − r)
        if numKeep < size(T') then
            // Search for and delete a point from T'
            Find the point p' with minimum weight
            Remove p' from T'
            Update weights of p''s neighbors
```

An example of a simplification update of the incremental algorithm is illustrated in Figure 3. In the figure, there are three types of points, unvisited points (empty, like node 7), visited points (grey color, like node 3), and reserved points (black color, like nodes 1, 4, 5). A solid line is a part of the existing simplified trajectory T' and a dashed line is a segment of its original trajectory T. The example assumes that we have already visited nodes from 1 to 5 and we can only keep four nodes in T'. Now, node 6 is added to T', but T' can only maintain 4 nodes so that a node has to be deleted from T'. Every three consecutive nodes in T' can form a triangle. The triangle of nodes 1-2-4 has the lowest height compared to nodes 2-4-5 and nodes 4-5-6. Therefore, node 2 is removed from T'. Calculating triangle's height is one of the geometric weighting functions, which we will discuss later.

It is common to implement a priority queue (minimum heap) to achieve a time complexity $O(\log n)$ for deleting a point with the smallest weight from the current simplified trajectory. However, the difficulty is that we need to search for the deleted point's neighbors and update their weights. This requires an ordered index for the simplified trajectory to keep the complexity of the search process to be $O(1)$ (using a double linked list) or $O(\log n)$ (using binary search). If both deleting the minimum weighted point and searching its temporal neighbors is achieved in $O(\log n)$ time, then the time complexity of *incremental simplification* is $O(n \log n)$.

The *incremental simplification* can be directly used for online or stream trajectory simplification. However, it has two major drawbacks. First, when the algorithm deals with a real-time GPS trajectory, its execution time increases significantly because increasing the size of the window incurs a search for more points. Second, this algorithm only considers the historical GPS points that it has considered before. This is a greedy algorithm that always considers current simplification as the best strategy. Its early simplification can lead to a global non-optimal solution.

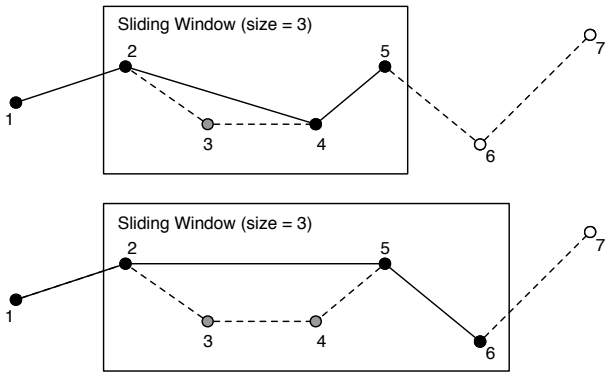


Figure 4: An example of the *sliding window simplification* process.

4.2 Sliding Window Simplification

To keep a constant execution time for handling with the online trajectory simplification, we present an algorithm based on the sliding window. The algorithm maintains a fixed-size window for its search instead of scanning all visited points. The visited points as for the sliding window approach are no longer checked.

An example illustrates the sliding window simplification (Figure 4). If the size of the sliding window is 3, only three points will be scanned when a simplification update is required. Node 1 is safe and will not be removed from the simplified trajectory. In this case, node 4 is removed from the simplified trajectory instead of node 2 that is deleted in the previous case of incremental algorithm (Figure 3).

Algorithm 1 can be modified to implement a sliding window algorithm. The algorithm still searches for a minimally weighted GPS point, but it only searches backward for k points (the size of the sliding window). Searching for a minimum and updating its neighbors can be achieved in $O(\log k)$ time, which leads to a time complexity of the sliding window simplification of $O(n \log k)$.

The advantage of using a sliding window is the speedup of execution time even dealing with a large real-time trajectory simplification. Its execution time does not grow with the increasing number of visited points, which is considerably faster than *incremental simplification*. It does not compute a simplified trajectory that is optimal, because instead of considering all visited points, it only considers closest k neighbors in past. The algorithm sacrifices optimality in return for a speedup.

4.3 Global Simplification

Finally, we propose a *global simplification* based on the S-DMin semantic line simplification [9] for map generalization. In this approach, we simplify noisy GPS trajectory by scanning the entire trajectory to search for the point with the minimum weight (Algorithm 2).

This method is an offline algorithm, which requires global knowledge of the trajectory. The main idea is to iteratively remove a point with the minimum weight until the desired compression ratio is achieved. In each step, the algorithm checks the entire remaining GPS trajectory to simplify the trajectory.

Two major considerations are considered in the *global simplification* algorithm. First, using always the entire trajectory, computing geometric and reliability weights for all points can be done at the very beginning. However, deleting a point requires an update for the weights of its neighbors. To keep the same topological consistency with its original trajectory, new self-intersections are not introduced and existing ones cannot be removed. Specifically, after initializing the weights, a list of existing self-intersections is created. Before removing a point from simplified trajectory, the list of existing self-intersections is checked to see whether it eliminates a self-intersection. In addition, new self-intersections cannot created after deleting a point.

Two critical parts determine the time complexity of the algorithm: (i) the search of a point with the minimum weight and the update of weights of its neighbors; (ii) the naive algorithm for self-intersection detection has a time complexity of $O(n^2)$ as it checks every pair of line segments for a trajectory. Bentley–Ottmann algorithm [2] is a line sweep algorithm to determine if n planar segments intersect. This algorithm can solve the problem in $O((n + k) \log n)$, where k is the number of intersections. The algorithm can be implemented by using a balanced binary search tree and a logarithmic-time priority queue.

Global simplification can provide a better simplification than *incremental simplification* and *sliding window simplification*. In addition, it can maintain the topological consistency without adding new self-intersections and removing existing ones.

However, the *global simplification* is an offline algorithm, which requires a global view of the trajectory. Furthermore, it is difficult to achieve a fast algorithm for the detection of the existing self-intersections, which limits the performance of *global simplification*. If it is not necessary to keep topological consistency, then self-intersections are eliminated during the simplification update. However, we still need to ensure that no new self-intersection are introduced in the simplified trajectory.

Algorithm 2: Global Trajectory Simplification

Input : $T = [p_{t_1}, p_{t_2}, ..., p_{t_N}]$ is a raw GPS trajectory that consists of measured GPS points; a compression rate r_{comp}.
Output: A simplified trajectory T' of T.
// Initialize weights
Calculate reliability weights Ω_r
for *GPS point* $p \in T$ **do**
  Compute geometric weight $\omega_g(p)$
  Retrieve reliability weight $\omega_r(p)$ from Ω_r
  Combined weight $\Omega(p) \longleftarrow \omega_g(p) \otimes \omega_r(p)$
Create a list I of all existing self-intersections in T
// Compute the number of points to be kept
$N_{remain} \longleftarrow N \times (1 - r_{comp})$
$T' \longleftarrow T$
while *length*$(T') > N_{remain}$ **do**
  Fetch the minimum weighted point p_k from Ω
  if $p_k \notin I$ *and not intersect*$(\overline{p_{k-1}, p_{k+1}}, T')$ **then**
    Remove point p_k from T'
    Update weights of p_{k-1} and p_{k+1} in T'
  else
    $\omega(p_k) \longleftarrow$ MAX_WEIGHT

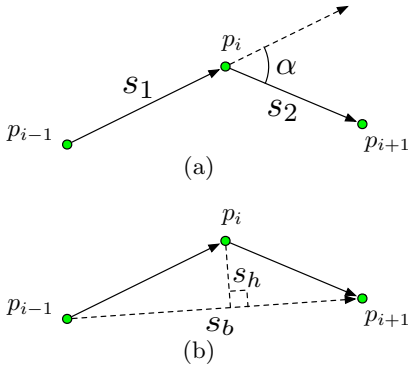


Figure 5: Examples of calculating geometric weights based on the length of segment, turning angle, and areas of triangles. p_{i-1}, p_i, and p_{i+1} are three consecutive points of a trajectory.

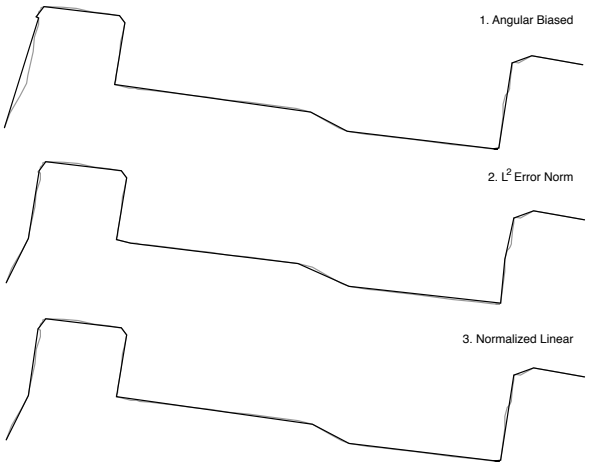


Figure 6: Trajectory simplification with applying three different geometric weights. Raw trace has 308 points (light grey) and simplified trace has 15 points (dark black).

5. ALGORITHM PARTICULARS

In this section, we discuss how to incorporate geometric weight ω_g and reliability weight ω_r into trajectory simplification for keeping the shape and reducing the noise.

5.1 Geometric Weight

The geometric weight ensures the simplified trajectory is close to its original trajectory. In general, the geometric weighting function decides how a point affects the shape of a trajectory. In other words, if the geometric weight of a point is small, then this point can be removed with a minimum change to the shape of the trajectory. The geometric weighting function takes into consideration its local situation, such as neighbor line segments, turning angle, area of triangle formed by two adjacent line segments, and perpendicular distance. Three weighting functions are presented:

- Angular biased: $f(s_1, s_2, \alpha) \mapsto s_1 \cdot s_2 \cdot \alpha^3$.
- L^2 error norm: $f(s_h, s_b) \mapsto \frac{1}{2} \cdot s_h \cdot s_b$.
- Normalised linear: $f(s_1, s_2, \alpha) \mapsto \frac{s_1 \cdot s_2 \cdot \alpha}{s_1 + s_2}$.

In the above definitions, s_1 and s_2 are the lengths of neighbor line segments of a point (Figure 5(a)). α is the turning angle at a point. s_h and s_b are the height and length of bottom of a triangle (Figure 5(b)), such that the L^2 error norm represents the size of its area. Due to the adaptability of defining a weighting function, different weights can also be attempted, e.g., only applying the triangle's height also returns a close shape. Furthermore, we tested with a length biased weight, $f(s_1, s_2) \mapsto \frac{s_1 \cdot s_2}{s_1 + s_2}$, which leads to a similar result to spatial subsampling that picks a GPS point within a certain distance.

Figure 6 illustrates the difference between giving different geometric weights. First, an angular biased weight is sensitive to points with a large turning angle, which is more accurate to keep turning corners of a trajectory. The L^2 error norm weight always removes the smallest triangle in terms of area. Triangles can be made up of any three consecutive points in a trajectory. Furthermore, normalized linear weight considers both length of line segments and turning angle, which gives a good approximation with respect to the geometry.

However, applying only geometric simplification for a GPS trajectory is not sufficient. All simplifying methods assume that the location of each point is completely correct, but this is not true for a noisy GPS trajectory. Due to the noisy representation of a GPS trajectory that contains outliers and stop points, we have to remove as many noisy points as possible.

5.2 Reliability

To remove noisy points, the definition of a weight is combined with the relative density as used in outlier detection [16] and the relative speeds of temporal neighbors. The motivation of reliability weight comes from an observation that an outlier always has a large distance and also an unusual speed comparing to its neighbors. Calculating density learns whether or not the location of a GPS point is far from its temporal neighbors through computing the average distance to them, and the speed of a GPS point is calculated as the average speed to its predecessors.

$$density(x,k) = \left(\frac{\sum_{y \in N(x,k)} distance(x,y)}{|N(x,k)|} \right)^{-1} \quad (1)$$

Eq.(1) calculates the density of a point, where k is the number of scanned neighbors, $N(x,k)$ returns a set of k nearest temporal neighbors of point x, and $|N(x,k)|$ returns the actual size of the set.

$$speed(x,k') = \frac{1}{|P(x,k')|} \cdot \sum_{y \in P(x,k')} \frac{distance(x,y)}{interval(x,y)} \quad (2)$$

Eq.(2) illustrates the computation of speed based on k' predecessors of point x, where $P(x,k')$ returns a set of k' predecessors and $interval(x,y)$ gives the time interval between point x and point y.

$$\omega(x,k) = \left| measure(x,k') - \sum_{y \in N(x,k)} \frac{measure(y,k'))}{|N(x,k)|} \right|^{-1} \quad (3)$$

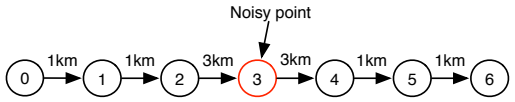


Figure 7: A simple example for calculating different of density and speed information from neighboring nodes. The numbers 0 to 6 refers to 7 GPS points in temporal order.

Algorithm 3: Calculating Reliability Weights

Input : A raw GPS trajectory $T = [p_{t_1}, p_{t_2}, ..., p_{t_N}]$, k is the number of nearest temporal neighbors, and k' is the number of predecessors.

Output: Ω_r is a list of reliability of weights for each point in T.

for *GPS point* $p \in T$ **do**
- Determine $N(p, k)$, the k-nearest neighbors of **p**
- Determine $P(p, k')$, the k' predecessors of **p**
- Compute $density(p, k)$
- Compute $speed(p, k')$

for *GPS point* $p \in T$ **do**
- // Compute the relative combined weight
- $\Omega_r(p) \longleftarrow \omega_d(x, k) \otimes \omega_s(x, k)$

where $\omega(x, k)$ is the similarity of x to its k nearest temporal neighbors. k' is used for calculating the *measure*, which is substituted by *density* (in Eq.(1)) or *speed* (in Eq.(2)).

Figure 7 shows an example that ignores directions of line segments and reduces the whole trajectory to a straight line. Take Point 3 as an example, given that $k = 4$ is the number of nearest temporal neighbors for calculating density, Point 3 has the distances of 4 km, 3 km, 3 km, 4 km to Point 1, 2, 4, and 5, respectively. Then, its density is calculated as $density(p_3, 4) = \left(\frac{4+3+3+4}{4}\right)^{-1} = \frac{2}{7}$ km^{-1}. Given that the number of predecessors for computing the speed is $k' = 1$ and the sampling rate is per minute, Point 3 has 3 km distance to Point 2 and then $speed(p_3, 1) = 3$ km/min. Similarly, applying to Point 1, 2, 4, 5 can get values of $\frac{1}{2}$, $\frac{1}{3}$, $\frac{1}{3}$, $\frac{1}{2}$ km in density and 1, 1, 3, 1 km/min in speed. To calculate the relative weight of Point 3, we need to compute the differences to its neighbors: $|\frac{2}{7} - (\frac{1}{2}+\frac{1}{3}+\frac{1}{3}+\frac{1}{2})\cdot\frac{1}{4}| = 0.131$ km^{-1} in density and $|3-(1+1+3+1)\cdot\frac{1}{4}| = 1.5$ km/min in speed. Both the density and the speed have the highest dissimilarities to its neighbors, which indicates that it has a high probability to be a noisy point. Finally, we get $\omega_d = 0.131^{-1} = 7.636$ and $\omega_s = 1.5^{-1} = 0.667$.

6. EXPERIMENTAL EVALUATION

In this section, we discuss the experiment setting, evaluation methodology, and experimental results. First, the section about the experiment setting presents the description of the used datasets and illustrates the evaluation procedure. In addition, we adopt the F_1 score as an error metric to calculate the distance between paths $Dist(P_1, P_2)$. Lastly, our experimental results show a dramatic improvement of map matching accuracy and execution time.

6.1 Experiment Setting

We evaluate the performance of our proposed algorithms on two benchmark datasets.

Table 1: Description of experimental datasets

Location	Seattle	Melbourne
Sampling	per second	per second
Number of points	7531	2512
Duration	about 2 hours	about 40 mins
Area	20.3km×11.9km	12.6km×11.6km
Error gaps	Yes	No
Ground truth	Yes	Yes

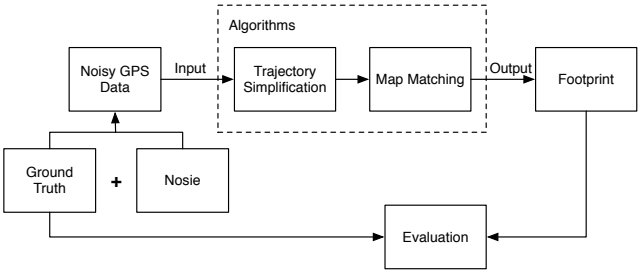


Figure 8: The procedure of testing the improvement of simplification on path inference.

Seattle Dataset. The dataset has been used in HMM experiments [13] and captures 2-hour driving in Seattle. It has 7531 GPS points and contains several time gaps without recorded points. The GPS trace covers a region between −122.357316° – −122.086483° longitude and 47.563166° – 47.671083° latitude. The ground truth is a sequence of edges in the map data.

Melbourne Dataset[1]. We recorded a trip in Melbourne of about 40 minutes to increase the diversity and the reliability of our experiments. The trace has 2512 GPS points. The road data is extracted from the OpenStreetMap[2], which covers 144.9316° – 145.0753° longitude and −37.8269° – −37.7221° latitude. The driver maps the raw trajectory afterwards to actual trip in the road network.

The procedure of our experiments is depicted in Figure 8. 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 precisely, if 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. Then, we apply the noisy data to our simplification and map matching algorithm to estimate true path. Next, we evaluate the closeness of the estimated trajectory to its ground truth. In our experiments, spatial sampling (SS) is our baseline algorithm, a common technique that has been applied to preprocess raw trajectory data [13].

6.2 Evaluation Methodology

F_1 score is commonly used in evaluation of classification

[1] http://people.eng.unimelb.edu.au/henli/projects/map-matching/

[2] http://www.openstreetmap.org/

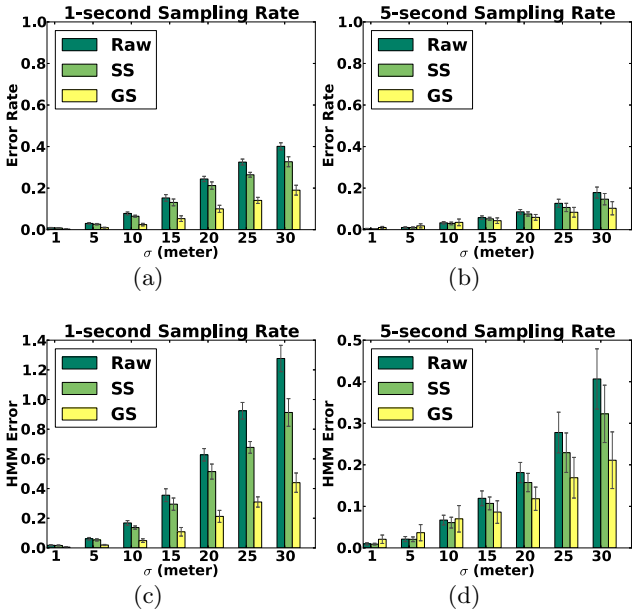


Figure 9: Comparisons of the error rates with different sampling rates in reference to the raw data (Raw), spatial sampling (SS), and *global simplification* (GS) with a 90% compression ratio on the Seattle dataset.

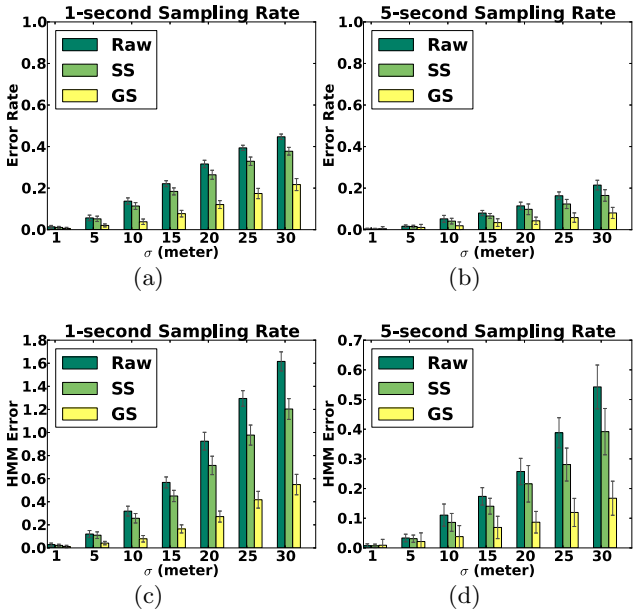


Figure 10: Comparisons of the error rates with different sampling rates in reference to the raw data (Raw), spatial sampling (SS), and *global simplification* (GS) with a 90% compression ratio on the Melbourne dataset.

problems in data mining and machine learning. Here, we use F_1 score to test the accuracy of map matching, which takes into consideration both *precision* and *recall*. *Precision* refers to the ratio of comparing the length of matched edges to the length estimated path P'. *Recall* calculates the ratio of comparing the length of matched edges to the length of ground truth P. F_1 score is the harmonic mean of *precision* and *recall*. Then, we transform F_1 to the error rate.

$$Precision = \frac{length(P \cap P')}{length(P')} \quad (4)$$

$$Recall = \frac{length(P \cap P')}{length(P)} \quad (5)$$

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (6)$$

$$ErrorRate = 1 - F_1 \quad (7)$$

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

We also use the error metric used for HMM map matching in [13] to demonstrate the performance of our algorithms. The reported error is calculated:

$$d_- = length(P) - length(P \cap P') \quad (8)$$

$$d_+ = length(P') - length(P \cap P') \quad (9)$$

$$HMM\ Error = \frac{d_- + d_+}{length(P)} \quad (10)$$

where d_- is the length of edges in the ground truth P that are not matched to the predicted path P' and d_+ is the length of mismatched edges in P'.

6.3 Experimental Results

6.3.1 Map Matching Accuracy

The experiments show that our proposed algorithms enhance the map matching accuracy significantly compared to the baseline algorithm. Specifically, through changing noise σ and sampling rates, we create different situations, e.g., mapping a high noise and high density trajectory. The comparisons of the map matching accuracy on the Seattle and Melbourne data set are shown in Figure 9 and Figure 10, respectively. The x-axis represents the standard deviation σ (in meters) in a zero-mean Gaussian distribution. The error rate of F_1 score ranges from 0 to 1 (a low value means less errors), as shown in Figure 9(a)(b) and Figure 10(a)(b). The HMM error has no maximum, as shown in Figure 9(c)(d) and Figure 10(c)(d). When the HMM error is high, the sum of length of mismatched sections relative to the ground truth and the inferred path is larger than the length of the ground truth. That is, the inferred path has a high error rate.

Figure 9 and 10 show that under high noise conditions ($\sigma > 20$ meters), our algorithm improves about 20% map matching accuracy according to the error rate of the F_1 score. Similar results are achieved using the HMM error metric. With increasing noise, the simplified trajectory can successfully reduce the influence of high noise while reconstructing trajectory. Simplifying sparse trajectory data can decrease improvement, because too sparse data incurs more errors and further reduction of GPS points causes inaccurate results. For example, in comparison with per second sampling, the improvement of simplifying on 5-second sampling data decreases to around 10% (Figure 10(b)).

A comprehensive performance summary of our proposed simplification technique to enhance map matching is de-

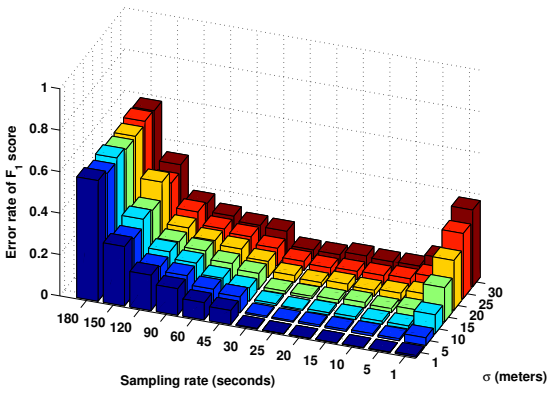


Figure 11: The error rates with different noisy levels and sampling rates after applying GS 70% on the Melbourne dataset. (Lower value has better performance.)

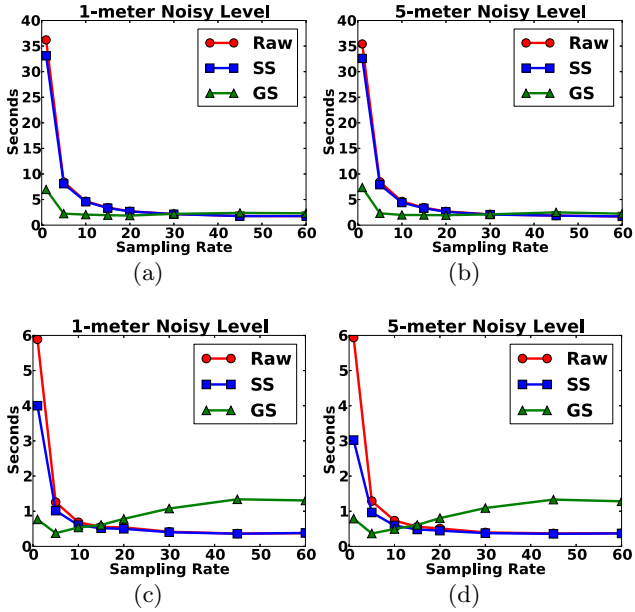


Figure 12: (a)(b) and (c)(d) show comparisons of map matching runtime with different sampling rates on the Seattle dataset and the Melbourne dataset, respectively.

picted in Figure 11. More precisely, it shows that when the sampling rate increases from 2 mins to 3 mins, the error rate soars from just under 20% to about 60%. In addition, for very high noise (30 meters σ) and very high sampling rates (1 second), the error goes up to about 40%. There are virtually no errors when sampling rates are 5-30 seconds and σ is less than 5 meters. We can see that both sparser data and higher noise can incur more errors for map matching. Furthermore, high sampling rates (1-5 seconds) do not always give the best results due to the interference from noise.

6.3.2 *Execution Time*

We can see from Figure 12 that with high sampling rates (1-10 seconds), *global simplification* reduces the execution time of map matching for 1 and 5 meters GPS noise.

Figure 12(a) and (b) show that the map matching with GS runs faster than using raw data and SS as long as the sampling rate is up to 20 seconds on the Seattle dataset. The reason is that GS reduces the number of points considerably which leads to a speedup for the search for the shortest path between two consecutive points when the trajectory data is dense. The shortest path search is a necessary step in HMM. Noisy and dense GPS data lead to an inaccurate result of shortest path search and longer execution time. For example, stop points lead to wrong estimated paths, which cause high execution times. Therefore, our simplification improves the running speed of HMM map matching.

However, on the Melbourne dataset, map matching using GS is only fast as long as the sampling rate is less than 10 seconds (Figure 12(c) and (d)). Since GS makes low sampling data more sparse, it increases the search time for shortest path computation between two consecutive points. For example, the Melbourne dataset has 2512 GPS points, but 252 points remain for the 10-second sampling rate. If we still use a simplification that reduces the dataset by a further 90%, then there are only around 26 points remaining. The search space of the simplified trajectory is considerably larger than the original trace. Hence, simplifying a sparse trajectory increases the execution time.

To validate the computational cost of our proposed methods, we measure and compare the execution time of three simplification algorithms (Figure 12). According to the figures, we can see that *sliding window simplification* (SWS) and *incremental simplification* (IS) is faster than *global simplification* (GS), because GS needs to scan the whole trajectory to find the point with minimum weight. SWS is faster than IS since the search space of IS becomes large with the growing window size. SWS keeps a fixed number of window to search the local minimum point, which does not increase if considering more GPS points.

6.3.3 *Compression Ratios*

Our experiments show that the compression ratio affects the map matching accuracy significantly. Comparisons of three different ratios, 30%, 60%, and 90%, are depicted in Figure 14. First, according to the figures, we can see that a high simplification ratio enhances the map matching accuracy when there are many GPS points for a trajectory. The error rate of 90% GS is always the lowest in Figure 14(a)(b)(c). We also point out the significant decrease of error rate from 60% to 90%, which means that a high simplification ratio can improve the accuracy substantially with the high sampling data.

On the other hand, when the data is sparse and there are not many GPS points for a trajectory, simplification can increase the error rate of map matching. Simplification is a lossy algorithm and an over-simplified trajectory loses necessary information required for map matching. For example, in Figure 14(d), when the sampling rate is 15-second, GS with 90% compression ratio causes higher error rates than for 30% and 60%. Therefore, the compression ratio of simplification should not be set too high, if there are not many GPS points in the trajectory. Simplification should be used for high sampling and noisy trajectory data rather than sparse data.

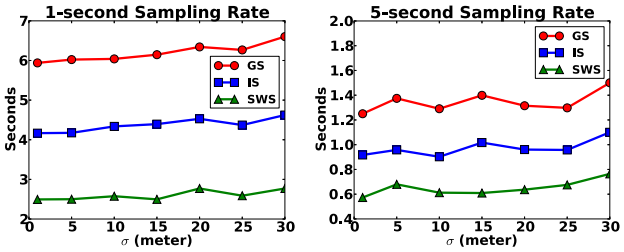


Figure 13: Comparisons of the execution time in reference to *global simplification* (GS), *incremental simplification* (IS), and *sliding window simplification* (SWS) (80% compression ratios) on the Melbourne dataset.

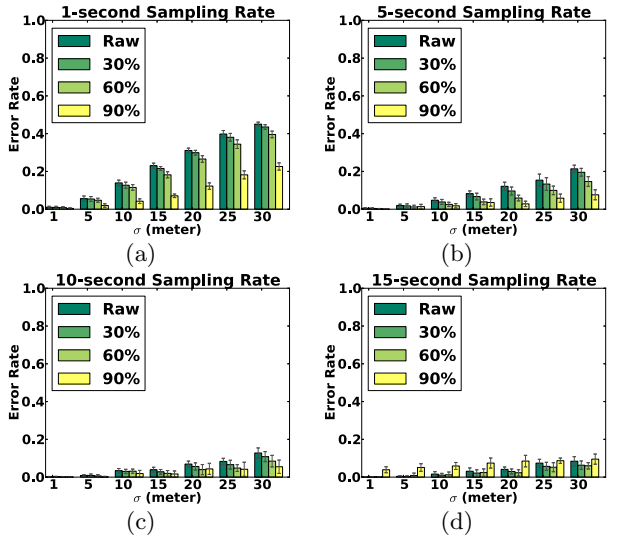


Figure 14: Comparisons of the error rates with different sampling rates in reference to *global simplification* (GS) with 30%, 60%, 90% compression ratio on the Melbourne dataset.

7. CONCLUSIONS AND FUTURE WORK

To conclude, we proposed three simplification algorithms to enhance the accuracy of map matching, which can deal with online and offline trajectory data. Furthermore, we use weighting functions to integrate spatial knowledge into trajectory simplification and we measure the noise degree of a GPS point. In addition to testing with an existing dataset, we collect our own GPS data and build ground truth. We conduct comprehensive experiments on real data. Experimental results show that under high sampling rates and noisy conditions, our algorithms improve the accuracy and computational cost of map matching considerably.

A trajectory simplification with adaptive compression ratio could be an important future direction. It learns the density of road network. We keep few points in sparse area to map GPS points, but remain more points in dense area.

8. ACKNOWLEDGMENTS

This research was supported under Australian Research Council's Discovery Projects funding scheme (project number DP130103705).

9. REFERENCES

[1] D. Agrawal, W. G. Aref, C.-T. Lu, M. F. Mokbel, P. Scheuermann, C. Shahabi, and O. Wolfson, editors. *17th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2009, November 4-6, 2009, Seattle, Washington, USA, Proceedings*. ACM, 2009.

[2] J. L. Bentley and T. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Computers*, 28(9):643–647, 1979.

[3] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. In K. Böhm, C. S. Jensen, L. M. Haas, M. L. Kersten, P.-Å. Larson, and B. C. Ooi, editors, *VLDB*, pages 853–864. ACM, 2005.

[4] H. Cao, O. Wolfson, and G. Trajcevski. Spatio-temporal data reduction with deterministic error bounds. *VLDB J.*, 15(3):211–228, 2006.

[5] Y. Chen, K. Jiang, Y. Zheng, C. Li, and N. Yu. Trajectory simplification method for location-based social networking services. In X. Zhou and X. Xie, editors, *GIS-LBSN*, pages 33–40. ACM, 2009.

[6] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[7] J. Gudmundsson, J. Katajainen, D. Merrick, C. Ong, and T. Wolle. Compressing spatio-temporal trajectories. *Comput. Geom.*, 42(9):825–841, 2009.

[8] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen. Discovery of convoys in trajectory databases. *PVLDB*, 1(1):1068–1080, 2008.

[9] L. Kulik, M. Duckham, and M. J. Egenhofer. Ontology-driven map generalization. *J. Vis. Lang. Comput.*, 16(3):245–267, 2005.

[10] J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: a partition-and-group framework. In C. Y. Chan, B. C. Ooi, and A. Zhou, editors, *SIGMOD Conference*, pages 593–604. ACM, 2007.

[11] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang. Map-matching for low-sampling-rate gps trajectories. In Agrawal et al. [1], pages 352–361.

[12] N. Meratnia and R. A. de By. Spatiotemporal compression techniques for moving point objects. In E. Bertino, S. Christodoulakis, D. Plexousakis, V. Christophides, M. Koubarakis, K. Böhm, and E. Ferrari, editors, *EDBT*, volume 2992 of *Lecture Notes in Computer Science*, pages 765–782. Springer, 2004.

[13] P. Newson and J. Krumm. Hidden markov map matching through noise and sparseness. In Agrawal et al. [1], pages 336–343.

[14] M. Potamias, K. Patroumpas, and T. K. Sellis. Sampling trajectory streams with spatiotemporal criteria. In *SSDBM*, pages 275–284. IEEE Computer Society, 2006.

[15] R. Song, W. Sun, B. Zheng, and Y. Zheng. PRESS: A novel framework of trajectory compression in road networks. *PVLDB*, 7(9):661–672, 2014.

[16] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.

[17] C. E. White, D. Bernstein, and A. L. Kornhauser. Some map matching algorithms for personal navigation assistants. *Transportation Research Part C: Emerging Technologies*, 8(1):91–108, 2000.

[18] J. Yuan, Y. Zheng, C. Zhang, X. Xie, and G. Sun. An interactive-voting based map matching algorithm. In T. Hara, C. S. Jensen, V. Kumar, S. Madria, and D. Zeinalipour-Yazti, editors, *Mobile Data Management*, pages 43–52. IEEE Computer Society, 2010.