Offline map matching using time-expanded graph for low-frequency data

Tanaka, Akira
Graduate School of Mathematics, Kyushu University

Tateiwa, Nariaki
Graduate School of Mathematics, Kyushu University

Hata, Nozomi
Graduate School of Mathematics, Kyushu University

Yoshida, Akihiro
Graduate School of Mathematics, Kyushu University

他

出版情報：Transportation Research Part C: Emerging Technologies. 130 (103265), 2021-09.
Elsevier
バージョン：
権利関係：

Highlights

Offline map matching using time-expanded graph for low-frequency data

Akira Tanaka, Nariaki Tateiwa, Nozomi Hata, Akihiro Yoshida, Takashi Wakamatsu, Shota Osafune, Katsuki Fujisawa

- Parameter-free time-dependent map-matching algorithm for low-frequency data
- Map matching based on the area between the predicted path and the vehicle trajectory
- Bottom-up segmentation and fractional cascading are applied for speedup

Offline map matching using time-expanded graph for low-frequency data

Akira Tanaka[a,*], Nariaki Tateiwa[a], Nozomi Hata[a], Akihiro Yoshida[a], Takashi Wakamatsu[b], Shota Osafune[c], Katsuki Fujisawa[d,e]

[a]*Graduate School of Mathematics, Kyushu University, 744 Motooka Nishi-ku Fukuoka, Japan*
[b]*YJ Card Corporation, 3-4-2, Hakataekimae Hakata-ku Fukuoka, Japan*
[c]*OkiDozen High School, 1403 Fukui, Oki District, Shimane, Japan*
[d]*Institute of Mathematics for Industry, Kyushu University, 744 Motooka Nishi-ku Fukuoka, Japan*
[e]*Artificial Intelligence Research Center, AIST, 2-3-36 Aomi Koto-ku Tokyo, Japan*

Abstract

Map matching is an essential preprocessing step for most trajectory-based intelligent transport system services. Due to device capability constraints and the lack of a high-performance model, map matching for low-sampling-rate trajectories is of particular interest. Therefore, we developed a time-expanded graph matching (TEG-matching) that has three advantages (1) high speed and accuracy, as it is robust for spatial measurement error and a pause such as at traffic lights; (2) being parameter-free, that is, our algorithm has no predetermined hyperparameters; and (3) only requiring ordered locations for map matching. Given a set of low-frequency GPS data, we construct a time-expanded graph (TEG) whose path from source to sink represents a candidate route. We find the shortest path on TEG to obtain the matching route with a small area between the vehicle trajectory. Additionally, we introduce two general speedup techniques (most map matching methods can apply) bottom-up segmentation and fractional cascading. Numerical experiments with worldwide vehicle trajectories in a public dataset show that TEG-matching outperforms state-of-the-art algorithms in terms of accuracy and speed, and we verify the effectiveness of the two general speedup techniques.

Keywords: Offline map matching, Time-expanded graph, Neighborhood search analysis, Fractional cascading, Bottom-up segmentation

1. Introduction

Map-matching algorithms determine the user or vehicle travel route by aligning the discrete positioning data to the road network and are driven by the ubiquity and improvement of positioning devices. Current map-matching solutions can be categorized into online and offline map-matching according to functional scenarios and applications. Online map-matching processes the current sample with a limited number of preceding or succeeding samples (Goh et al., 2012; Yin et al., 2018). The process is often fast and straightforward for interactive performance and is used for route guidance, autonomous cars, collision avoidance systems, lane departure warning, emergency response, and enhanced driver awareness systems (White et al., 2000; Toledo-Moreo et al., 2010; Sathiaseelan, 2011). In contrast, offline map matching is performed after the entire trajectory is obtained, aiming for an optimal matching route with fewer processing time constraints. Offline map matching is utilized for traffic flow analysis, road pricing, traffic surveillance, and transport operations (Velaga and Pangbourne, 2014).

To review the current status of map-matching and determine future research directions, Chao et al. (2020) and Huang et al. (2021) classified map-matching models based on the technical perspective (core matching model) and the sampling frequency of positioning data, respectively. Chao et al. (2020) focused on categorizing only competitive algorithms while including new models that appear after the last comprehensive survey. These algorithms are classified into four categories: *similarity*, *state-transition*, *candidate-evolving*, and *scoring* models.

*Corresponding author
Email address: akirat.tanaka@kyudai.jp (Akira Tanaka)

The *similarity* models return the vertices and arcs that are geometrically *closest* to the trajectory. Based on the definition of *closest*, the *similarity* models are divided into two subcategories. Distance-based models regard the spatial distance as the closeness between a trajectory and a matched path, and Fréchet distance (Alt et al., 2003; Wei et al., 2013) and the longest common subsequence (Zhu et al., 2017) are commonly used approaches. Meanwhile, the pattern-based algorithm assumes that people tend to travel on the same paths given a pair of origin and destination points. Historical map-matched data were utilized to determine similar travel patterns (Zheng et al., 2012).

The *state-transition* model builds a weighted topological graph whose vertices represent the possible *state* where the vehicle may be located at a particular moment. The arcs represent the *transitions* between states at different timestamps. The matched path is then obtained from the optimal path in the graph globally. There are three primary ways to build a graph and solve the optimal path problem: the hidden Markov model (HMM) (Newson and Krumm, 2009; Goh et al., 2012), conditional random field (CRF) (Hunter et al., 2014), and weighted graph technique (WGT) (Hsueh and Chen, 2018; Hu et al., 2017). In the HMM, each trajectory sample is regarded as the observation, while the actual location of the vehicle on the road, which is unknown, is the hidden state. The optimal path is obtained by the Viterbi algorithm, which is a dynamic programming approach. CRF is a statistical model and considers interactions among observations. HMM is also a statistical model and focuses only on the relationship between an observation and the state at the same stage.

The WGT enumerates candidate points for each positioning data and selects the most probable point sequence by creating a weighted candidate graph. A candidate point corresponds to any of the following things: (1) a road, (2) an endpoint of a road, and (3) a point on a road. For example, the spatio-temporal based matching algorithm (STD-matching) (Hsueh and Chen, 2018) and the AntMapper algorithm (Gong et al., 2018) use "(3) a point on a road" as a candidate point. Their algorithms calculate the shortest path problems between any pair of two consecutive candidate points to set the candidate graph's weight, but this incurs a high computational cost. Focusing on high-sampling-rate global positioning system (GPS) trajectories, Tang et al. (2016) introduced a time-dependent graph to address this problem. For each position fix, their algorithm utilizes the potential path area rather than the candidate points; hence, they do not need to solve the shortest path problems between two consecutive candidate points. Their algorithm finally produces a reasonable network-time path, representing the expected arc travel times and dwell times at possible intermediate stops. Although they succeeded in building both offline and real-time map-matching algorithms for high-sampling-rate trajectories, few studies have developed their model to consider low-sampling-rate trajectories. Therefore, we have developed a time-expanded graph matching (TEG-matching) for low-sampling-rate trajectories.

The *candidate-evolving* model holds a set of candidates (also known as particles or hypotheses) during map matching. The candidate set is initiated based on the first trajectory sample. It continues to evolve by adding new candidates propagated from the old ones close to the latest measurements while pruning the irrelevant ones. Compared to the *state-transition* model, the *candidate-evolving* type is more robust for off-track matching issues, and the particle filter (PF) (Wang and Ni, 2016; Bonnifait and Laneurit, 2009) and the multiple hypothesis technique (Taguchi et al., 2019; Knapen et al., 2018) are two representative solutions. The PF is a state estimation technique that combines Monte Carlo sampling methods with Bayesian inference. The PF model's general idea is to recursively estimate the probability density function (PDF) of the road network section around the observation as time advances. According to the moving status, the particles with higher weights are more likely to propagate to feed particles for the next cycle, while those with low weights are likely to die. The multiple hypothesis technique determines the scores to the candidate road edge (or point) rather than approximating the complicated PDF of the neighboring map area, which leads to a reduction in computation.

The *scoring* models (Quddus and Washington, 2015; Toledo-Moreo et al., 2010) assign a group of candidates to each trajectory segment (or location observation) and finds an arc from each group that maximizes the predefined scoring function. According to the working scenario, every timestamp's discovered segment is either returned or joined with other matched segments. Road network generation from crowdsourced trajectories is one of the related works of map-matching, and this was addressed by Yang et al. (2018), resulting in the successful automatic generation of lane-based intersection maps.

On the other hand, Huang et al. (2021) classified map-matching algorithms based on the sampling frequency and insisted that GPS information is desired to be recorded at a lower frequency to save energy consumption and communication cost produced by large-scale GPS devices. Yuan et al. (2010) collected positioning data of more than 10,000 taxis in Beijing and revealed that 66% occupied the low-frequency data (sampling frequency is more than one minute). Therefore, map-matching algorithms based on low-frequency sampling data have attracted much attention

in recent years (Huang et al., 2021; Chen et al., 2014; Yuan et al., 2010; Hsueh and Chen, 2018; Gong et al., 2018). Chen et al. (2014) proposed an online candidate-evolving map-matching algorithm for large-scale low-frequency data using multi-criteria dynamic programming (MDP). The MDP technique reduces the number of candidate routes when candidate routes stored from the previous position are extended to candidate routes at the current position. As for offline map-matching, (Yuan et al., 2010) developed an interactive voting-based map-matching algorithm. The authors consider spatial information, temporal information, and the mutual influence between matched points for neighboring GPS points. Recent works for tackling the same issue are STD-matching (Hsueh and Chen, 2018) and AntMapper algorithm (Gong et al., 2018), which utilizes WGT. STD-matching and the AntMapper algorithm enumerate candidate roads for each position data and use the longest path problem and an ant colony algorithm, respectively, to find the best combination of candidate roads.

Although these studies contribute to developing an offline map-matching algorithm for low-frequency data, there are the following shortcomings:

1. Few models achieve both high speed and accuracy.
2. Some methods require additional information such as velocity and angle, which are not obtained by some vehicles.
3. Some algorithms suffer from the tuning of hyperparameters.

Therefore, we utilized a concept from previous research (Tang et al., 2016) and developed a time-expanded graph matching (TEG-matching). We leverage the concept of TEG and formulate the map-matching problem as the shortest path problem, which achieves high speed and accuracy with only positioning data (Section 4). The elaborate weight on TEG makes the proposed model parameter-free (Section 4.2).

2. Problem setting

In this study, we handle a 2D road network containing two types *nodes*: *junctions* and *shape nodes*. A one-way road (hereinafter called *arc*) is represented by a polyline whose first and last points are *junctions*, and the other points are *shape nodes*. Let V be the set of *junctions* and *shape nodes*, A be the set of *arcs*, G be a *road network* that has *arc* set A. Then, *arc* a is represented as $a \coloneqq (v_1, v_2, \dots, v_m)$, $\{v_j\}_{j=1}^{m} \subset V$, and every two-way road is expressed as two one-way roads such as $(v_1, \dots, v_m) \in A$ and $(v_m, v_{m-1}, \dots, v_1) \in A$. A pair of two consecutive points of an *arc* is referred to as a *shape arc* and denoted by (v_i, v_{i+1}), where $v_i, v_{i+1} \in V$. A vehicle *trajectory* is a chronologically ordered position fixes $\boldsymbol{P} = (P_1, P_2, \dots, P_{n+1})$ produced by a GPS device mounted on the vehicle. For each time step i, P_i includes only the east and north coordinates. The proposed algorithm aims to restore the most likely path $(a_1, a_2, \dots, a_m)$ $(a_j \in A)$ of the vehicle under a given vehicle trajectory $\boldsymbol{P} = (P_i)_{i=1}^{n+1}$. The predicted path is referred to as the *matched path*, and the *arc* included in the *matched path* is called the *matched arc*. Fig. 1 represents an example of map-matching for a vehicle trajectory $\boldsymbol{P} = (P_1, P_2, \dots)$ and the *matched path* $((v_0, v_1), (v_1, v_4, v_5), (v_5, v_6), \dots)$. The *correct path* denotes the a finite sequence of A on which the vehicle actually travels, and the *arc* included in the *correct path* is called the *correct arc*.

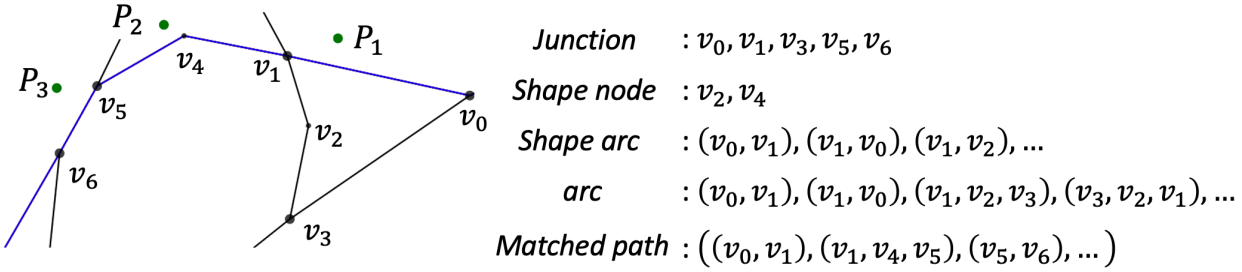

Figure 1: Example of a *road network*, a *vehicle trajectory*, and the corresponding *matched path*.

3. Abstract of the proposed model

3.1. Model description and contributions

Intuitively, if the area between the vehicle *trajectory* $(P_i)_{i=1}^{n+1}$ and the *matched path* is sufficiently small, the *match path* almost coincides with the *correct path*; hence, our algorithm finds the *matched path*, which reduces the area. We first specify a potential area $L(G, P_i)$, where the vehicle may pass through from time step i to $i + 1$. Then, each pair of two consecutive position fixes (P_i, P_{i+1}) $(i \in \{1, 2, \ldots, n\})$ is simultaneously matched to a path (referred to as a *partial path*) on $L(G, P_i)$ while maintaining the connectivity of the two consecutive *partial paths*. A selected *partial path* tends to have few abrupt direction changes and creates a small area with the two position fixes. We formulate the matching as the shortest path problem on a time-expanded graph (TEG). In summary, our contributions are listed as follows:

- We propose a parameter-free offline map-matching approach called TEG-matching that requires only the vehicle's ordered locations. TEG-matching is robust for spatial measurement errors and pauses such as in traffic lights. We experimentally achieved a higher Jaccard index of 0.098 and a 5.6x faster outcome than two state-of-the-art algorithms, namely, the STD-matching (Hsueh and Chen, 2018) and the AntMapper algorithm (Gong et al., 2018). TEG-matching can be easily extended to cooperate with various traffic information, such as speed limits and traffic volume.

- We apply fractional cascading (FC) to candidate *shape node* search and verify a 2.5x speedup compared with the k-d tree. To make use of FC, we also conduct a geometric analysis and answer "how big a square is needed to obtain all *arcs* within a radius r from a position fix."

- We apply the existing bottom-up segmentation to a road network and achieves a 64% reduction of shape nodes, resulting in a 1.78x speedup with only a 0.0074 accuracy drop for map-matching.

3.2. Procedure for the proposed map-matching algorithm

The steps involved in our proposed map-matching model are as follows:

1. Preprocess
   (a) Define *junctions* and *shape nodes*
   Some datasets do not distinguish between *junctions* and *shape nodes*; hence, nodes on the road network are classified into these two types according to the topology of the road network. We also split long *shape arcs* (add *shape nodes*) such that the length of each *shape arc* is less than or equal to the predefined parameter ℓ_{max}. This is because that we may overlook *arcs* close to a certain point if the *arc* contains some long *shape arcs*. This situation occurs when both endpoints of a long *shape arc* are far from the point because we report an *arc* if at least one *shape node* of the *arc* is in a square centered on the point. The road network that completes this process is referred to as the *processed road network*.
   (b) Fractional cascading (Section 6)
   The first step of map-matching is to obtain the *arcs* close to a vehicle *trajectory*, and FC data structure is utilized to this end. We want to report *arcs* within r meter from a certain point, and an elementary geometry determines whether or not *arcs* are in the circle. However, this process is computationally expensive if the road network has vast *arcs*. We speed up this step by utilizing FC that reports every *arc* that has at least one *shape node* belonging to a square centered on a certain point. Based on the Theorem 2 , if we set the side length of the square $c = \max\left(r, \frac{\ell_{max}+2r}{2\sqrt{2}}\right)$ if $\ell_{max} \leq 2\left(1 + \sqrt{2}\right) r$ $\left(\text{otherwise } c = \frac{\ell_{max}}{2}\right)$, we obtain either of the endpoints of the *shape arc* within a radius r meter from a certain point. This implies we acquire all *arcs* within r meter from the point. Before performing map-matching, we construct FC data structure with all the *shape nodes* of *processed road network*; and the FC is utilized for the step "2. TEG-matching >(a) Obtain neighborhood arcs" as mentioned below.
   (c) Bottom-up segmentation (Section 5)
   Processed road network have some redundant *shape nodes* to represent the shape of the road. Bottom-up segmentation reduces these nodes and contributes to memory reduction and speedup in map matching.

2. TEG-matching
   (a) Obtain neighborhood *arcs*
   Given a vehicle *trajectory* $\boldsymbol{P} = (P_i)_{i=1}^{n+1}$, we first obtain *arcs*, where the vehicle may travel from time stamp i to $i+1$ $(i \in \{1, \ldots, n\})$. We assume that these *arcs* lie within $r' = d(P_i, P_{i+1})/2 + r_{\mathrm{GPS}}$ from the midpoint of P_i and P_{i+1} (hearinafter denoted by $P_{i,i+1}$), where r_{GPS} is the upper bound of the spatial measurement error, and $d(x, y)$ is the Euclidean distance between x and y. From the above discussion, if we set $\ell_{max} = 2(1+\sqrt{2})r_{\mathrm{GPS}} \left(\leq 2(1+\sqrt{2})r'\right)$ and $c_i = \max\left(r', \frac{\ell_{max}+2r'}{2\sqrt{2}}\right)$, we can obtain either of the endpoints of *shape arcs* within r' meter from $P_{i,i+1}$ and obtain all *arcs* that lie within r' from $P_{i,i+1}$. We speed up the query by utilizing FC.
   (b) Construct a time-expanded graph (TEG) and find the shortest path on TEG (Section 4)
   We construct a TEG that represents the space-time movement of the vehicle. To attain the most plausible *matched path*, we find the shortest path on the TEG and restore the *matched path* from the shortest path.

The remainder of this paper is organized as follows. We propose TEG-matching in Section 4, and a graph reduction method called bottom-up segmentation is described in Section 5. Section 6 presents a fractional cascading (FC) that is a speedup technique of finding *arcs* close to a vehicle *trajectory*. The effectiveness of TEG-matching, bottom-up segmentation, and FC is demonstrated through numerical experiments in Section 7. Our conclusions are presented in Section 8, and the proof relevant to the square query is presented in Section Appendix B.

4. Time-expanded graph

Given a time series of GPS recording, our model first builds the time-expanded graph (TEG) and obtains a *matched path* by solving the shortest path problem on the TEG. This section starts with the topology construction of the TEG in Section 4.1 and then explains three types of weights on TEG in Section 4.2.

We first introduce some notation used in this section. Given $X, Y \subset \mathbb{R}^2$ or $X, Y \in \mathbb{R}^2$, $d(X, Y)$, and $d_\infty(X, Y)$ are the Euclidean and Chebyshev distances between X and Y, respectively. For example, the distance between a *shape arc* (v_1, v_2) and a position fix P_i is denoted by $d((v_1, v_2), P_i)$. $d((v_1, v_2), P_i)$ is the perpendicular distance if the perpendicular distance is achieved on the segment (v_1, v_2); otherwise, $\min\{d((v_1, P_i), d((v_2, P_i)\}$. Similarly, if $a = (v_1, \ldots, v_m)$ is an *arc*, $d(a, P_i) = \min_{1 \leq j \leq m-1} d\left((v_j, v_{j+1}), P_i\right)$. The first and last *node* of an *arc* a are denoted by $\mathrm{tail}(a) := v_1$ and $\mathrm{head}(a) := v_m$, respectively.

4.1. Topology construction of the time-expanded graph

Given a *road network* G that has *arc* set A, we define the line graph (Ray-Chaudhuri, 1967) $L(G)$ of G as the directed graph whose vertex set $V(L(G)) := A$ and whose directed edge set $E(L(G)) := \{(a, \tilde{a}) \mid a, \tilde{a} \in A, \mathrm{head}(a) = \mathrm{tail}(\tilde{a})\}$. Given G and a vehicle *trajectory* $\boldsymbol{P} = (P_i)_{i=1}^{n+1}$, the corresponding time-expanded graph (TEG), referred to as $T(G, \boldsymbol{P}) = (V(T(G, \boldsymbol{P})), E(T(G, \boldsymbol{P})))$, consists of n subgraphs of $L(G)$ denoted by

$$L(G, P_i) = (V(L(G, P_i)), E(L(G, P_i))) \ (1 \leq i \leq n) \tag{1}$$

. The i-th *layer graph* $L(G, P_i)$ represents a partial road network where the vehicle may travel from time stamp i to $i+1$; hence $L(G, P_i)$ includes the area near P_i and P_{i+1}. Specifically, $V(L(G, P_i))$ includes all *arcs* that lie within $r' = d(P_i, P_{i+1})/2 + r_{\mathrm{GPS}}$ from the $P_{i,i+1}$ where r_{GPS} is the upper bound of measurement error, and $P_{i,i+1}$ is the midpoint of P_i and P_{i+1}. Based on the previous discussion in Section 3.2, we define

$$V(L(G, P_i)) := \left\{ a^i \mid a = (v_1, \ldots, v_m) \in A, \min_{1 \leq j \leq m} d_\infty(v_j, P_{i,i+1}) \leq c_i/2 \right\} \text{ and} \tag{2}$$

$$E(L(G, P_i)) := \left\{ \left(a^i, \tilde{a}^i\right) \in V(L(G, P_i)) \times V(L(G, P_i)) \mid \mathrm{head}(a) = \mathrm{tail}(\tilde{a}) \right\}, \tag{3}$$

where c_i is the positive number defined in Section 3.2. For any notation x, x^i is the copy of x related to the time stamp i, and the superscript i is used for distinguishing copies related to different time stamps. For instance, a^i exists in i-th *layer graph* and is the copy of the *arc* $a \in A$. $a^i \in V(L(G, P_i))$ and $(a^i, \tilde{a}^i) \in E(L(G, P_i))$ are referred to as a *layer*

vertex and *layer edge*, respectively. $(a^i, \tilde{a}^i)$ implies that the vehicle moves from the *arc* a to the *arc* $\tilde{a}$ between the time stamps i and $i+1$. To express the travel of the vehicle from time stamp i to $i+2$, we define the set of *layer-to-layer edges* $E_{\mathrm{LtL}}\left(L(G, P_i, P_{i+1})\right)$ as

$$E_{\mathrm{LtL}}\left(L(G, P_i, P_{i+1})\right) \coloneqq \left\{(a^i, a^{i+1}) \in V\left(L(G, P_i)\right) \times V\left(L(G, P_{i+1})\right) \mid d(a, P_{i+1}) \leq r_{\mathrm{GPS}}\right\} \tag{4}$$

for $i \in \{1, 2, \ldots, n-1\}$. (a^i, a^{i+1}) implies that the vehicle lies on *arc* a at the time stamp $i+1$. We also define *source* s and *sink* t so that we formulate the map-matching problem as a shortest path problem from s to t. The set of *source edges* $E_{\mathrm{source}}\left(L(G, \boldsymbol{P})\right)$ and the set of *sink edges* $E_{\mathrm{sink}}\left(L(G, \boldsymbol{P})\right)$ are defined as follows:

$$E_{\mathrm{source}}\left(L(G, \boldsymbol{P})\right) \coloneqq \left\{(s, a^1) \mid a^1 \in V\left(L(G, P_1)\right)\right\} \tag{5}$$

$$E_{\mathrm{sink}}\left(L(G, \boldsymbol{P})\right) \coloneqq \left\{(a^n, t) \mid a^n \in V\left(L(G, P_n)\right)\right\}. \tag{6}$$

In conclusion, the TEG $T(G, \boldsymbol{P}) = \left(V\left(T(G, \boldsymbol{P})\right), E\left(T(G, \boldsymbol{P})\right)\right)$ is defined as follows:

$$V\left(T(G, \boldsymbol{P})\right) \coloneqq \left(\bigcup_{i=1}^{n} V\left(L(G, P_i)\right)\right) \cup \{s, t\} \tag{7}$$

$$E\left(T(G, \boldsymbol{P})\right) \coloneqq \left(\bigcup_{i=1}^{n} E\left(L(G, P_i)\right)\right) \cup \left(\bigcup_{i=1}^{n-1} E_{\mathrm{LtL}}\left(L(G, P_i)\right)\right) \cup E_{\mathrm{source}}\left(L(G, \boldsymbol{P})\right) \cup E_{\mathrm{sink}}\left(L(G, \boldsymbol{P})\right) \tag{8}$$

A path from s to t represents the vehicle's travel from time stamp 1 to $n+1$, and the next subsection assigns weight to $L(G, \boldsymbol{P})$ to find the most reasonable space-time path. The following theorem insists that TEG has the path corresponding to the *correct path* under a certain condition.

Theorem 1. *Let G be a road network that has* arc *set A, $\boldsymbol{P} = (P_i)_{i=1}^{n+1}$ be a vehicle trajectory. If the vehicle lies within $r' = d(P_i, P_{i+1})/2 + r_{GPS}$ from the $P_{i,i+1}$ between the time stamp i and $i+1$ $(1 \leq \forall i \leq n)$, the TEG has the* path *from* source *s to* sink *t that corresponds to the* correct path.

Proof. Let $\left(a^i_1, a^i_2, \ldots, a^i_{L_i}\right)$ be the *correct path* (the sequence of *arcs* that the vehicle actually travels) between the time stamp i and $i+1$. Then, the equation $a^i_{L_i} = a^{i+1}_1$ $(1 \leq i \leq n-1)$ satisfies. As $V\left(L(G, P_i)\right)$ includes all *arcs* that lie within $r' = d(P_i, P_{i+1})/2 + r_{\mathrm{GPS}}$ from the $P_{i,i+1}$, especially $V\left(L(G, P_i)\right)$ includes all the *arcs* that lie within r_{GPS} from P_i and P_{i+1}. At the time step 1, as the vehicle lies within r_{GPS} from P_1, $a^1_1 \in V\left(L(G, P_1)\right)$, which implies $\left(s, a^1_1\right) \in E_{\mathrm{source}}\left(L(G, \boldsymbol{P})\right)$. For the same reason, $\left(a^n_{L_n}, t\right) \in E_{\mathrm{sink}}\left(L(G, \boldsymbol{P})\right)$. We also know $a^i_k \in V\left(L(G, P_i)\right)$ $(1 \leq \forall i \leq n, 1 \leq \forall k \leq L_i)$ from the assumption. From the characteristic of the *(correct) path*, $\mathrm{head}\left(a^i_k\right) = \mathrm{tail}\left(a^i_{k+1}\right)$ $(1 \leq \forall i \leq n, 1 \leq \forall k \leq L_i - 1)$ holds. Hence $\left(a^i_k, a^i_{k+1}\right) \in E\left(L(G, P_i)\right)$. If we are concern about $a^i_{L_i} \in V\left(L(G, P_i)\right)$, $a^{i+1}_1 \in V\left(L(G, P_{i+1})\right)$, $a^i_{L_i} = a^{i+1}_1$ and $d\left(a^{i+1}_1, P_{i+1}\right) \leq r_{\mathrm{GPS}}$, we have $\left(a^i_{L_i}, a^{i+1}_1\right) \in E_{\mathrm{LtL}}\left(L(G, P_i)\right)$ for $1 \leq \forall i \leq n-1$. In summary,

$$\left\{\left(s, a^1_1\right), \left(a^1_1, a^1_2\right), \ldots, \left(a^1_{L_i-1}, a^1_{L_i}\right), \left(a^1_{L_i}, a^2_1\right), \left(a^2_1, a^2_2\right), \ldots, \left(a^n_{L_n-1}, a^n_{L_n}\right), \left(a^n_{L_n}, t\right)\right\} \tag{9}$$

is the *path* from source s to sink t on TEG that corresponds to the *correct path*.

4.2. Weight of the time-expanded graph

The path from *source* s to sink t on the TEG represents the space-time travel of the vehicle, and we formulate map matching as a shortest path problem from s to t. To quantify the improbability of the path, we consider three points: (1) area between the path and the vehicle trajectory, (2) abrupt direction changes, and (3) spatial measurement error. The corresponding weight functions are *area weight* $w_{\mathrm{area}}(\cdot)$, *direction change weight* $w_d(\cdot)$, and *spatial weight* $w_s(\cdot)$, respectively. The following subsections explain the details of these weights and their motivations using mathematical expressions. The last subsection integrates these three weights into the weight of TEG.

We introduce some notations and definitions, which are illustrated in Fig. 2. Let $p, p', v_j \in \mathbb{R}^2$ $(j \in \{1, 2, \ldots\})$ be a two-dimensional point coordinate. For a polyline $a = (v_1, \ldots, v_m)$, the $l(a) \coloneqq \sum_{j=1}^{m-1} d(v_j, v_{j+1})$ denotes the

length of the polyline a. We define $\tilde{d}(p,(v_1,v_2))$ as the perpendicular distance between p and the segment (v_1,v_2), that is, $\tilde{d}(p,(v_1,v_2))$ is the Euclidean distance between p and the line containing the two points v_1 and v_2. The point on the "line" (not segment) that achieves the perpendicular distance is called the *perpendicular point* and is denoted by $\zeta_{p,(v_1,v_2)}$. If $\zeta_{p,(v_1,v_2)}$ lies on (v_1,v_2), we say that "p achieves perpendicular distance on (v_1,v_2)". The *projection* of p onto (v_1,v_2) is denoted by $\eta_{p,(v_1,v_2)}$. If p achieves a perpendicular distance on (v_1,v_2), $\zeta_{p,(v_1,v_2)} = \eta_{p,(v_1,v_2)}$ satisfies; otherwise, $\eta_{p,(v_1,v_2)} = \underset{v\in\{v_1,v_2\}}{\operatorname{argmin}} d(p,v)$. Similarly, we say that "p achieves perpendicular distance on a polyline $a=(v_1,\ldots,v_m)$" if at least one segment (v_j,v_{j+1}) $(1\le j\le m-1)$ achieves the perpendicular distance on (v_j,v_{j+1}). If p achieves perpendicular distance on a, the *perpendicular point* $\zeta_{p,a}$ is defined as the nearest *perpendicular point* for (v_j,v_{j+1}) $(1\le j\le m-1)$, that is, $\zeta_{p,a} \coloneqq \zeta_{p,(v_j,v_{j+1})}$ such that p achieves perpendicular distance on (v_j,v_{j+1}) and $\tilde{d}\left(p,\left(v_j,v_{j+1}\right)\right) \le \tilde{d}\left(p,\left(v_{j'},v_{j'+1}\right)\right)\ \forall j' \in \{j' \mid p \text{ achieves perpendicular distance on } (v_{j'},v_{j'+1})\}$. Otherwise, $\zeta_{p,a} \coloneqq$ nil, where "nil" implies that the position fix p does not have the *perpendicular point* on the polyline a. The tail and head of the segment on which the *perpendicular point* $\zeta_{p,a}$ lies on are denoted by $v_{j(\text{tail},p,a)}$ and $v_{j(\text{head},p,a)}$, respectively. Specifically, we define $v_{j(\text{tail},p,a)} \coloneqq v_j$ and $v_{j(\text{head},p,a)} \coloneqq v_{j+1}$ if $\zeta_{p,a} = \zeta_{p,(v_j,v_{j+1})}$; otherwise, if $\zeta_{p,a} =$ nil, $v_{j(\text{tail},p,a)} = v_{j(\text{head},p,a)} =$ nil. The *perpendicular distance* between p and a polyline a is denoted by $\tilde{d}(p,a)$ and is defined as

$$\tilde{d}(p,a) \coloneqq \min\left\{\tilde{d}\left(p,(v_j,v_{j+1})\right) \mid 1\le j\le m-1, p \text{ achieves } \textit{perpendicular distance} \text{ on } (v_j,v_{j+1})\right\} \tag{10}$$

if p achieves *perpendicular distance* on a. The *projection* of p onto a polyline a is denoted by

$$\eta_{p,a} \coloneqq \underset{\eta_{p,(v_j,v_{j+1})}}{\operatorname{argmin}} d\left(p,\eta_{p,(v_j,v_{j+1})}\right) \tag{11}$$

subject to $1\le j\le m-1$. Given two point coordinates $p,p'\in\mathbb{R}^2$, we say that the polyline $a=(v_1,\ldots,v_m)$ "lies on p' side" if $\zeta_{v_j,(p,p')} \in \{p'+(p'-p)\theta \mid \theta\ge 0\}$ for all $j\in\{1,\ldots,m\}$. The closest *perpendicular point* is called the *nearest perpendicular point* and denoted by $\zeta^*_{a,(p,p')} \coloneqq \underset{\zeta_{v_j,(p,p')}}{\operatorname{argmin}} d\left(\zeta_{v_j,(p,p')},(p,p')\right)$ subject to $j\in\{1,\ldots,m\}$, and the corresponding point on the polyline is denoted by $v^*_{a,(p,p')} \coloneqq \underset{v_j}{\operatorname{argmin}} d\left(\zeta_{v_j,(p,p')},(p,p')\right)$ subject to $j\in\{1,\ldots,m\}$. Fig. 2 visualizes these notations and the example of a polyline that "lies on p' side."

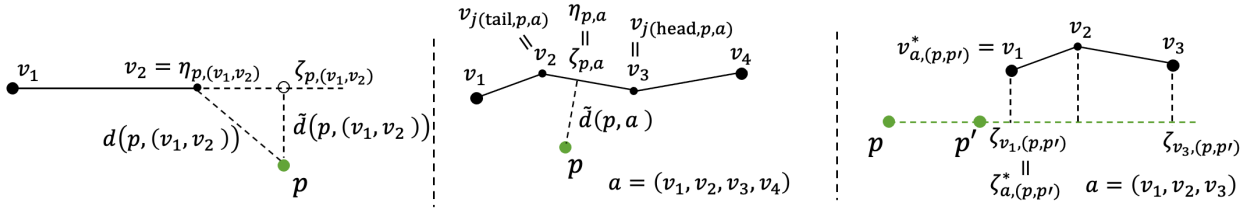

Figure 2: (Left and center) Visualization of a distance $d(\cdot)$, perpendicular distance $\tilde{d}(\cdot)$, a *projection* η, and the *perpendicular point* ζ. (Right) Example of a polyline that "lies on the p' side."

4.2.1. Area weight

If the area between a *vehicle trajectory* and the *matched path* is sufficiently small, the *matched path* is probably the same as the *correct path*. We attempt to obtain the path that reduces the area, but lining up a limited number of probable candidate paths is difficult. To address the issue, the *area weight* of an *arc* is defined as the area between the *arc* and a *vehicle trajectory*; then, the area between a path and a *vehicle trajectory* is defined as the total *area weight* overall arcs contained in the path. The *area weight* facilitates the acquisition of the path that has a small area with a *vehicle trajectory* by assigning the *area weight* to the TEG. This section defines the *area weight*, and the assignment is explained in Section 4.2.4. Fig. 3 illustrates the *area weights* as the sum of the green and yellow areas. We calculate the *area weights* in various ways according to the relative positions of the *arc* and position fixes, and the color corresponds to these situations. We categorize the relative positions and explain how to calculate the *area weight* for each case.

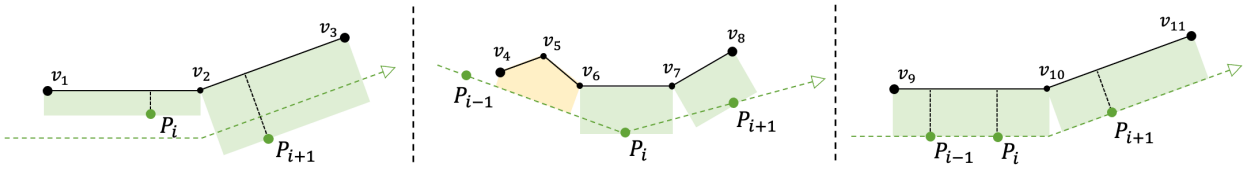

Figure 3: Examples of *area weight* $w_{\text{area}}(\cdot)$. *Area weight* is represented as the sum of the areas of green and yellow.

The *area weight* $w_{\text{area}}(\cdot)$ is defined using $w_{\text{area0}}(\cdot)$, $w_{\text{area1}}(\cdot)$, and $w_{\text{area2}}(\cdot)$, which are illustrated in Fig. 4; hence, we first provide these three definitions. Let $p, p', p_i, v_j \in \mathbb{R}^2$ $(i, j \in \{1, 2, \dots\})$ be a two-dimensional point coordinate. Suppose that (p, p') and $(p_1, \dots, p_n)$ are subsequences of a *vehicle trajectory*, and $a = (v_1, \dots, v_m)$ is a subsequence of an *arc*. Then, $w_{\text{area0}}((p_1, \dots, p_n), a)$ defines the area between $(p_1, \dots, p_n)$ and a under the assumption that the $\{p_i\}_{i=1}^n$ are obtained when the vehicle travels on a. Formally, if every p_i $(1 \le i \le n)$ achieves *perpendicular distance* on a, $w_{\text{area0}}((p_1, \dots, p_n), a)$ is defined as follows:

$$w_{\text{area0}}((p_1, \dots, p_n), a) \coloneqq \left(\frac{1}{n}\sum_{i=1}^{n} \tilde{d}(p_i, a)\right) l(a) = \left(\frac{1}{n}\sum_{i=1}^{n} \tilde{d}(p_i, a)\right)\left(\sum_{j=1}^{m-1} d(v_j, v_{j+1})\right) \tag{12}$$

$w_{\text{area1}}(p, p', a)$ defines the area between a and the segment (p, p') under the assumption that the position fix p (p') is acquired before (after) the vehicle travels on a. More precisely, $w_{\text{area1}}(p, p', a)$ is normally the area between a and the line containing p and p'. We define $w_{\text{area1}}(p, p', a)$ as the sum of the areas between the *shape arc* (v_j, v_{j+1}) $(j = 1, \dots, m-1)$ and the line containing p and p'; that is, $w_{\text{area1}}(p, p', a) \coloneqq \sum_{j=1}^{m-1} w_{\text{area1}}(p, p', v_j, v_{j+1})$, where $w_{\text{area1}}(p, p', v, v')$ is defined as follows. $w_{\text{area1}}(p, p', v, v')$ is the improbability estimation of the vehicle passing through the *shape arc* (v, v') during the interval of p and p'. $w_{\text{area1}}(p, p', v, v')$ becomes large if the angle difference between (v, v') and (p, p') is large, or if the (v, v') is far from the (p, p'). Normally, $w_{\text{area1}}(p, p', v, v')$ becomes the area between (v, v') and the line containing p and p'. However, the exact definition depends on the two spatial conditions: (1) whether the v, v' are separated by the line containing (p, p'); and (2) whether the angle between the (p, p') and (v, v') is less than $\pi/2$. The exact definition and the corresponding figures are summarized in Table 1 and Fig. 5, respectively.

$w_{\text{area2}}(p, p', a)$ denotes the penalty that the vehicle passes through a far from the two consecutive position fixes p and p' during the interval of p and p'. Formally, $w_{\text{area2}}(p, p', a)$ is defined as follows:

$$w_{\text{area2}}(p, p', a) \coloneqq \begin{cases} d\left(\zeta^*_{a,(p,p')}, (p, p')\right) d\left(v^*_{a,(p,p')}, \zeta^*_{a,(p,p')}\right) & \text{if } a \text{ lies on } p \text{ or } p' \text{ side} \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

, as shown in the right side on Fig. 4.

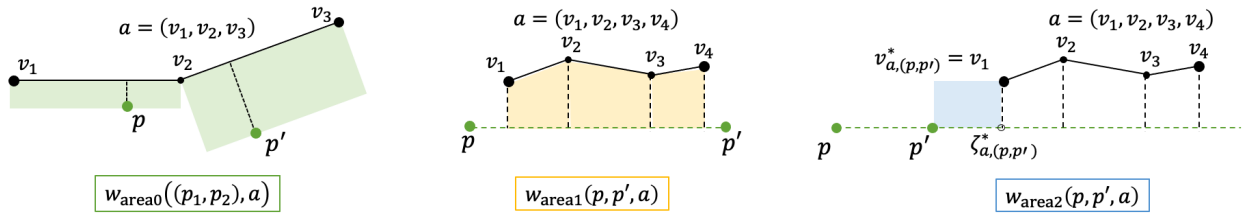

Figure 4: Visualization of $w_{\text{area0}}((p_1, p_2), a)$, $w_{\text{area1}}(p, p', a)$, and $w_{\text{area2}}(p, p', a)$.

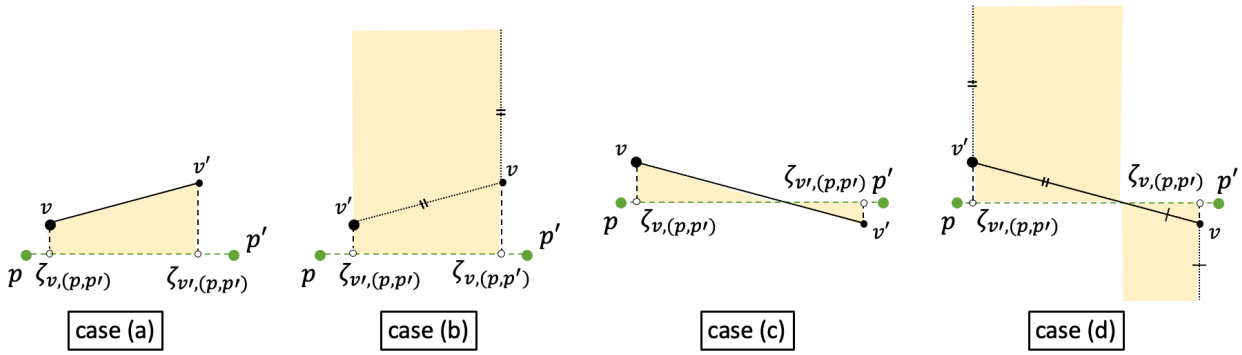

Figure 5: Visualization of $w_{\text{area1}}(p, p', v, v')$.

case	(1) v, v' are separated	(2) angle $< \pi/2$	$w_{\text{area1}}(p, p', v, v')$
(a)		✓	$\left(d\left(v, \zeta_{v,(p,p')}\right) + d\left(v', \zeta_{v',(p,p')}\right)\right) d\left(\zeta_{v,(p,p')}, \zeta_{v',(p,p')}\right)/2$
(b)			$d\left(\zeta_{v',(p,p')}, \zeta_{v,(p,p')}\right)\left(d\left(v, \zeta_{v,(p,p')}\right) + d(v', v)\right)$
(c)	✓	✓	$\frac{\left(d\left(v,\zeta_{v,(p,p')}\right)^2 + d\left(v',\zeta_{v',(p,p')}\right)^2\right) d\left(\zeta_{v',(p,p')},\zeta_{v,(p,p')}\right)}{2\left(d\left(v,\zeta_{v,(p,p')}\right) + d\left(v',\zeta_{v',(p,p')}\right)\right)}$
(d)	✓		$\frac{\left(d\left(v,\zeta_{v,(p,p')}\right) + d\left(v',\zeta_{v',(p,p')}\right) + d(v',v)\right)\left(d\left(v,\zeta_{v,(p,p')}\right)^2 + d\left(v',\zeta_{v',(p,p')}\right)^2\right) d\left(\zeta_{v',(p,p')},\zeta_{v,(p,p')}\right)}{\left(d\left(v,\zeta_{v,(p,p')}\right) + d\left(v',\zeta_{v',(p,p')}\right)\right)^2}$

Table 1: Definition of $w_{\text{area1}}(p, p', v, v')$

Consider the *area weight* $w_{\text{area}}(a^i)$ for the *layer vertex* $a^i \in V(L(G, P_i))$, where the vehicle may travel from the time step i to $i+1$. If we denote the *arc* by $a = (v_1, \ldots, v_m)$, $w_{\text{area}}(a^i)$ indicates the improbability that the vehicle travels on the *arc* a from the time step i to $i+1$. The improbability is basically expressed as the area between the arc a and the vehicle trajectory. However, the exact definition depends on the relative positions of the *arc* a, position fixes P_{i-1}, P_i, and P_{i+1}. Fig. 6 illustrates the $w_{\text{area}}(a^i)$, and Table 2 summarizes the definition of $w_{\text{area}}(a^i)$ by cases, and the detailed explanation is given below:

1. (case1) Both P_i and P_{i+1} achieve *perpendicular distance* on a. ($\zeta_{P_i,a} \neq$ nil, and $\zeta_{P_{i+1},a} \neq$ nil)
   We can suppose that the vehicle trajectory is $\left(\tilde{d}(P_i, a) + \tilde{d}(P_{i+1}, a)\right)/2$ away from the *arc* a on average; hence, we define $w_{\text{area}}(a^i) \coloneqq w_{\text{area0}}((P_i, P_{i+1}), a)$.
2. (case2) P_i achieves perpendicular distance on a, but P_{i+1} does not. ($\zeta_{P_i,a} \neq$ nil, and $\zeta_{P_{i+1},a} =$ nil)
   When a vehicle passes through the *shape arc* $(v_{j(\text{tail},P_i,a)}, v_{j(\text{head},P_i,a)})$, the vehicle trajectory is likely $\tilde{d}(P_i, a)$ away from the *shape arc*. Hence, we add $w_{\text{area0}}\left((P_i), (v_{j(\text{tail},P_i,a)}, v_{j(\text{head},P_i,a)})\right)$ to $w_{\text{area}}(a^i)$. Because P_{i+1} does not achieve a perpendicular distance on a, we can suppose that the vehicle passes through the polyline

   $$(v_{j(\text{head},P_i,a)}, v_{j(\text{head},P_i,a)+1}, \ldots, v_m)$$

   during the interval of P_i and P_{i+1}. This implies that we add

   $$w_{\text{area1}}\left(P_i, P_{i+1}, (v_{j(\text{head},P_i,a)}, v_{j(\text{head},P_i,a)+1}, \ldots, v_m)\right)$$

   to $w_{\text{area}}(a^i)$. For the polyline $\left(v_1, v_2, \ldots, v_{j(\text{tail},P_i,a)}\right)$, we consider two cases. If P_{i-1} achieves a perpendicular distance on a, we can suppose that the vehicle trajectory $\left(\tilde{d}(P_{i-1}, a) + \tilde{d}(P_i, a)\right)/2$ away from the polyline on average. Hence, we add

   $$w_{\text{area0}}\left((P_{i-1}, P_i), \left(v_1, \ldots, v_{j(\text{tail},P_i,a)}\right)\right)$$

   to $w_{\text{area}}(a^i)$. In the other case, that is, if P_{i-1} does not achieve a perpendicular distance on a, we suppose that the P_{i-1} is obtained before the vehicle passes through the a. Hence, we add

   $$w_{\text{area1}}\left(P_{i-1}, P_i, (v_1, \ldots, v_{j(\text{tail},P_i,a)})\right)$$

to $w_{\text{area}}(a^i)$.

3. (case3) P_i does not achieve perpendicular distance on a.
   Because $w_{\text{area}}(a^i)$ represents the improbability of the vehicle passing through the arc a from the time stamp i to $i+1$, we suppose that P_i is acquired before the vehicle travels on *arc* a. Hence, we add $w_{\text{area1}}(P_i, P_{i+1}, a)$ to $w_{\text{area}}(a^i)$. Moreover, we assign the penalty $w_{\text{area2}}(P_i, P_{i+1}, a)$ to $w_{\text{area}}(a^i)$ to address the situation in which the *arc* a is far from the segment (P_i, P_{i+1}).

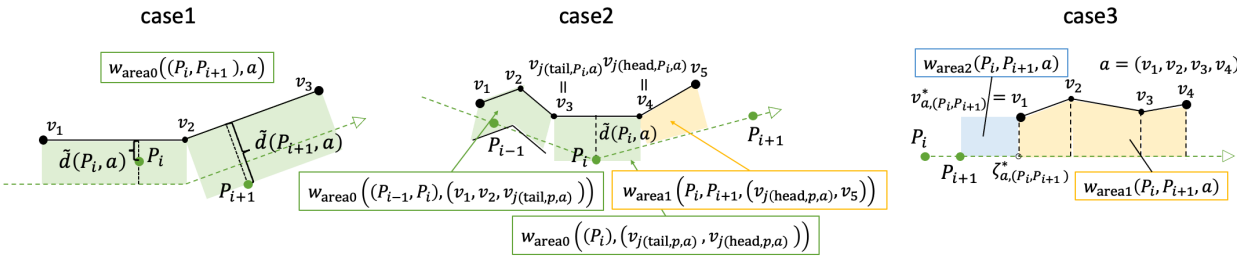

Figure 6: Visualization of $w_{\text{area}}(a^i)$ by cases.

$\zeta_{P_i,a} \neq \text{nil}$	$\zeta_{P_{i+1},a} \neq \text{nil}$	$\zeta_{P_{i-1},a} \neq \text{nil}$	case	$w_{\text{area}}(a^i)$
✓	✓	✓	case1	$w_{\text{area0}}((P_i, P_{i+1}), a)$
✓	✓			
✓		✓	case2	$w_{\text{area0}}\left((P_{i-1}, P_i), \left(v_1, \ldots, v_{j(\text{tail},P_i,a)}\right)\right)$ $+w_{\text{area0}}\left((P_i), (v_{j(\text{tail},P_i,a)}, v_{j(\text{head},P_i,a)})\right)$ $+w_{\text{area1}}\left(P_i, P_{i+1}, (v_{j(\text{head},P_i,a)}, v_{j(\text{head},P_i,a)+1}, \ldots, v_m)\right)$
✓			case2	$w_{\text{area1}}\left(P_{i-1}, P_i, (v_1, \ldots, v_{j(\text{tail},P_i,a)})\right)$ $+w_{\text{area0}}\left((P_i), (v_{j(\text{tail},P_i,a)}, v_{j(\text{head},P_i,a)})\right)$ $+w_{\text{area1}}\left(P_i, P_{i+1}, (v_{j(\text{head},P_i,a)}, v_{j(\text{head},P_i,a)+1}, \ldots, v_m)\right)$
	✓	✓	case3	$w_{\text{area1}}(P_i, P_{i+1}, a) + w_{\text{area2}}(P_i, P_{i+1}, a)$
	✓			
		✓		

Table 2: Definition of $w_{\text{area}}(a^i)$

4.2.2. *Direction change weight*

We assume that the position fix is obtained every time the vehicle makes a significant direction change, and the *direction change weight* $w_d(\cdot)$ reflects this assumption. $w_d(\cdot)$ becomes large if there is no position fix near a significant direction change when the vehicle moves from one *arc* to another. For an *layer edge* $(a_1^i, a_2^i) \in A^i$, the $w_d\left((a_1^i, a_2^i)\right)$ is defined as the square of the distance between the common *node* of these two arcs (a_1 and a_2) and the segment whose endpoints are η_{P_i,a_1} and η_{P_i,a_2}.

$$w_d\left((a_1^i, a_2^i)\right) := \{d\left(\text{head}(a_1), (\eta_{P_i,a_1}, \eta_{P_i,a_2})\right)\}^2 \tag{14}$$

Fig. 7 shows two cases where the position fix is far and near from a large direction change at the intersection of two *arcs*. We assume that the vehicle travels $(\ldots, v_1, v_2, v_3, v_4, v_5, \ldots)$, and P_i is acquired when the vehicle passes through v_3.

4.2.3. *Spatial weight*

Spatial weight $w_s(\cdot)$ is the penalty when the vehicle lies on the *arc* far from the position fix P_{i+1} at the time stamp $i+1$. Because *layer-to-layer edge* (a^i, a^{i+1}) indicates that the vehicle lies on the *arc* a at the time step $i+1$, $w_s\left((a^i, a^{i+1})\right)$

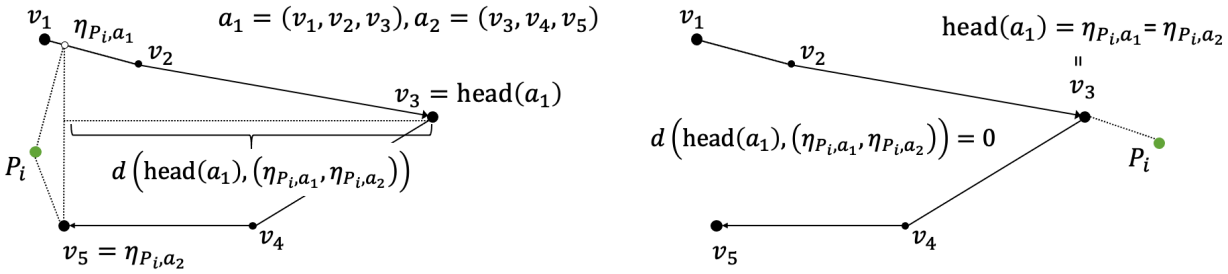

Figure 7: Visualization of $w_d\left((a_1^i, a_2^i)\right)$. The left (right) side is the case where the position fix is far (near) from a significant direction change at the intersection of two *arcs*.

is defined as the square of the distance between the position fix P_{i+1} and arc a:

$$w_s\left((a^i, a^{i+1})\right) := \{d(P_{i+1}, a)\}^2 \tag{15}$$

We define similar but large weights for the first and last position fix (P_1 and P_{n+1}) to prevent the *matched path* from being shorter than the *correct path*. The shorter path is caused by the property of the shortest path problem. Formally, for a *source arc* (s, a^1) and a *sink arc* (a^n, t), we define

$$w_s\left((s, a^1)\right) := d(P_1, a) l_{\text{mean}} \tag{16}$$

$$w_s\left((a^n, t)\right) := d(P_{n+1}, a) l_{\text{mean}} \tag{17}$$

, where l_{mean} is the average *arcs* length.

4.2.4. *Weight of the edge in TEG*

This section integrates *area*, *direction change*, and *spatial* weights into the weight of the TEG. Let G be a *road network*, $\boldsymbol{P} = (P_i)_{i=1}^{n+1}$ be the vehicle *trajectory*, $T(G, \boldsymbol{P}) = (V(T(G, \boldsymbol{P})), E(T(G, \boldsymbol{P})))$ be the corresponding TEG, where $V(T(G, \boldsymbol{P}))$ is the vertex set and $E(T(G, \boldsymbol{P}))$ is the edge set. We assign a weight to each edge of the TEG by transferring the vertex weight to the edge weight. Formally, for each edge $e = (e_{\text{tail}}, e_{\text{head}}) \in E(T(G, \boldsymbol{P}))$, *edge weight* is defined as follows:

$$w(e) := \begin{cases} w_{\text{area}}(e_{\text{tail}}) + w_d(e) & (\text{if } e \text{ is a } \textit{layer edge}) \\ w_s(e) & (\text{if } e \text{ is a } \textit{source edge} \text{ or } \textit{layer-to-layer edge}) \\ w_s(e) + w_{\text{area}}(e_{\text{tail}}) & (\text{if } e \text{ is a } \textit{sink edge}) \end{cases} \tag{18}$$

Our model solves the shortest path problem from *source* to *sink* on the TEG, thus obtaining the *matched path*.

5. Bottom-up segmentation

A segmentation method is applied to both (1) a *road network* to reduce the graph size and (2) vehicle *trajectory* to generate the evaluation data (low-sampling-rate trajectories). Keogh et al. (2002) undertook an extensive review and empirical comparison of several piecewise linear approximation techniques for the time series databases and proposed a new algorithm. Although such an approach does not initially develop for a directed graph, the approach is transferred to graph simplification while maintaining the topology of the graph. This section introduces three basic segmentation techniques, and we apply the most suitable one to the *road network* and the vehicle *trajectory*. We further visualized the *arcs* applied and did not apply the segmentation technique using different hyper-parameters.

Given a time series data $T_1, T_2, \ldots, T_n$, each of which includes only coordinates (spatial data), these algorithms aim to create a polyline similar to the time series data using a smaller number of line segments. According to Keogh et al. (2002), we essentially categorize time-series segmentation algorithms into three groups: (1) sliding windows (Koski

et al., 1995; Park et al., 2001), (2) top-down (DOUGLAS and PEUCKER, 1973; Park et al., 1999), and (3) bottom-up (Keogh and Pazzani, 1998; Luebke, 2001). The sliding window algorithm is a simple and intuitive online algorithm. It works by anchoring the first data point as the tail of a potential segment and then approximates the data with increasingly longer segments. At some point i, the potential segment's error is greater than the user-specified threshold; thus, the subsequence from the anchor to $i-1$ is transformed into a segment. The top-down algorithm works by considering every possible partitioning of the time series and splitting it at the best location. We then tested whether the approximation error of each subsection was below a user-specified threshold. A failed subsection was recursively split until all the segments had approximation errors below the threshold. The bottom-up segmentation is a natural complement to the top-down algorithm. The algorithm begins by creating the finest possible approximation of the time series. The merge cost of two adjacent segments is calculated, and the lowest cost pair is iteratively merged until a stopping criterion is met. Keogh et al. (2002) concluded that the sliding window algorithm shows a generally poor quality, and the bottom-up algorithm often significantly outperforms the other two algorithms. The properties of the three algorithms are summarized in Table 3.

Table 3: Feature summary for the three widely-known algorithms.

Algorithm	User can specify[1]	Online
Top-Down	E,ME,K	No
Bottom-Up	E,ME,K	No
Sliding Window	E	Yes

[1] E and ME are the maximum errors for a given segment and for an entire time series, respectively, where K represents the number of segments.

Because we apply a segmentation method to a *road network* and vehicle *trajectories* offline, bottom-up segmentation is the most suitable approach. We introduce some notation to describe the procedure of bottom-up segmentation. Let $T = (t_1, t_2, \dots)$ be a finite sequence of points, where t_i is a coordinate. $T[a : b] := (t_a, t_{a+1}, \dots, t_b)$ denotes the contiguous subsequence of T from a-th point to b-th point. A piecewise linear approximation of T is the output of bottom-up segmentation and is denoted by Seg_TS. Seg_TS is defined as the sequence of *approximate segments*, each of which is a two-element subsequence of T. An *approximate segment* "Seg" $(= (t_{\text{tail}}, t_{\text{head}}))$ approximates a contiguous subsequence of T (polyline), and tail(Seg) $:= t_{\text{tail}}$ and head(Seg) $:= t_{\text{head}}$ denote the endpoints of the "Seg." The *approximation error* of "Seg," denoted by calculate_error(Seg), is the maximum distance between the "Seg" and one of the approximated contiguous subsequence points. merge(Seg, Seg') := (tail(Seg), head(Seg')) is the rough *approximate segment* integrating two *approximate segments* Seg and Seg'. Hence, the corresponding two approximated contiguous subsequences are also merged.

Using these symbols, we show the pseudocode in Algorithm 1, and the right side of Fig. 8 shows how the algorithm works. The *max_error* is the parameter that determines the approximation accuracy, and the influence is visualized on the left side in Fig. 8. The figure indicates that too large a *max_error* destroys the road shape; thus, we have to choose an appropriate *max_error* based on the complexity and density of a *road network*.

6. Fractional cascading

All map-matching algorithms first restrict candidate nodes and arcs near positioning data. Because this operation is often repeated for each position fix, it can be a computational bottleneck. To speed up this operation, we introduced the fractional cascading (FC), proposed by Chazelle and Guibas (1986).

FC is a data structure for an orthogonal range query with a query time of $O(\log n + k)$ in the 2-dimensional space, where n is the total number of points in the data structure, and k is the number of the points lying in the orthogonal range. The range tree, an existing data structure, has a query time $O\left((\log n)^2 + k\right)$, and the k-d tree (Bentley, 1975) has a query time $O\left(n^{1/2} + k\right)$. Thus the FC is found to be an efficient algorithm, although it requires significant memory.

For 1-dimensional range queries, we commonly utilize the simple binary search tree. The set of points is split into two subsets of approximately equal sizes: one subset contains points smaller than or equal to the splitting value, while the other subset contains the points larger than the splitting value. The splitting value is stored at the root, and the two

Algorithm 1: *bottom-up segmentation*(T, max_error)

Input : a sequence of points $T = (t_1, t_2 \ldots, t_n)$, $n \geq 3$; max_error that decides the approximation accuracy
Output: A piecewise linear approximation of T, denoted by Seg_TS

```
Function calculate_error(Seg1, Seg2):
    merge_seg ← merge(Seg1,Seg2)
    merge_cost ← calculate_error(merge_seg)
    return merge_cost
Function bottom_up_segmentation(T,max_error):
    // Initialization
    Seg_TS ← [(t1, t2), (t2, t3), . . . , (tn−1, tn)]
    for i ← 1 to n − 2 do
        merge_costs[i] ← calculate_error(Seg_TS[i], Seg_TS[i+1])
    while min(merge_costs) < max_error and len(Seg_TS) > 1 do
        // Find the minimum pair to merge
        i ← arg min_i(merge_costs[i])
        Seg_TS[i] ← merge(Seg_TS[i], Seg_TS[i + 1])
        delete(Seg_TS[i + 1])
        delete(merge_costs[i])
        if 0 < i − 1 then
            merge_costs[i-1] ← calculate_error(Seg_TS[i-1], Seg_TS[i]))
        if i + 1 <len(Seg_TS)+1 then
            merge_costs[i] ← calculate_error(Seg_TS[i], Seg_TS[i+1])
    return Seg_TS
```

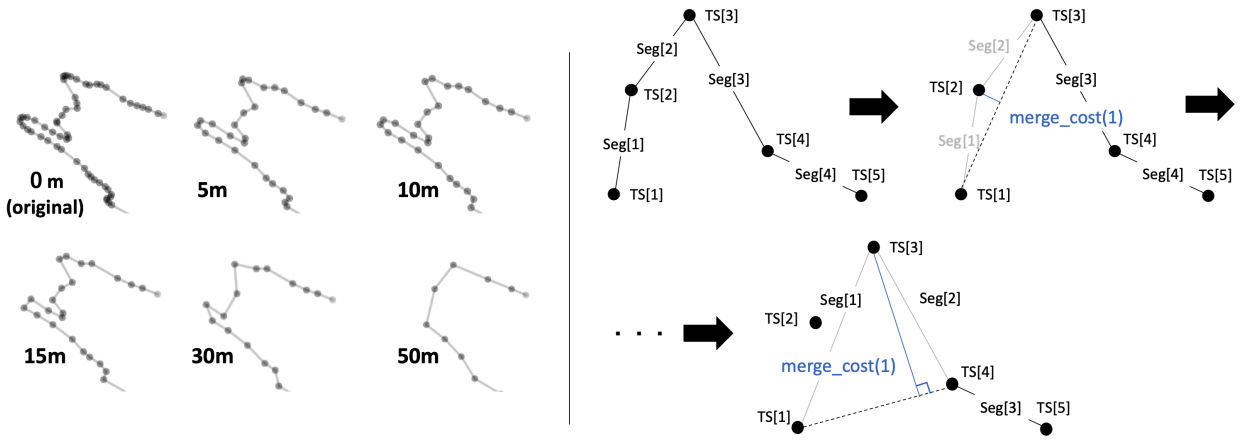

Figure 8: (Left) Influence of max_error on the bottom-up segmentation. The max_error is written at the lower left of each drawing. (Right) Procedure for the bottom-up segmentation.

subsets are stored recursively in the two subtrees. This structure can be expanded to higher-dimensional range queries using FC; in terms of 2-dimension, we can search query points along the x- and y-axes simultaneously.

The construction of the FC consists of two stages: *creating a binary tree* and *setting minmax and maxmin pointers*. *Creating a binary tree* is remarkably similar to the aforementioned one-dimensional case. The set of points is recursively split into two subsets of roughly equal size according to x-value. The only difference is that every vertex v of an FC tree contains not only the split point v_{split} but also a sorted list v_{list}, a sub-list of points in lexicographical order for

	Query time	Space complexity
Fractional Cascading	$O(\log n + k)$	$O(n \log n)$
range-tree	$O\left((\log n)^2 + k\right)$	$O(n \log n)$
kd-tree	$O\left(n^{1/2} + k\right)$	$O(n)$

Table 4: Complexities of each data structure for 2-dimensional data

(y, x). The pseudocode is described as "create_fctree" in Algorithm 2, where v_{left} and v_{right} represent the left and right children of vertex v, respectively. An example of an FC tree is illustrated in Fig. 9 along with the original data points. At the root vertex v_0, the lexicographical order for (x, y) is C, A, E, F, D, G, B; hence, F is selected as *split point*. All the points smaller than or equal to F (i.e., C, A, E, F) are held by the left child v_1 and sorted in lexicographical order for (y, x). The *minmax (maxmin) pointers* facilitate specifying points smaller (larger) than or equal to a specific point in lexicographical order for (y, x). Let $v_{\text{list}}[i]$ be the i-th point of v_{list} and v_{child} be the child of v. The *minmax (maxmin) pointer* $\text{minmax}(v_{\text{list}}[i], v_{\text{child}})$ ($\text{maxmin}(v_{\text{list}}[i], v_{\text{child}})$) is defined as the pointer pointing to the smallest (largest) element of v_{child} larger (smaller) than or equal to $v_{\text{list}}[i]$ in lexicographical order (y, x), respectively. If no element satisfies the condition, the pointer points to nil. For example, Fig. 9 shows that all the points of v_1 larger than or equal to B in lexicographical order (y, x) are C, E, F which are located on the right side of the head of the $\text{minmax}(B, v_1)$ (blue arc). The pseudocode of *setting minmax and maxmin pointer* is described as "set_pointer" in Algorithm 2, and the overall procedure of building FC tree is also represented as "main" in Algorithm 2.

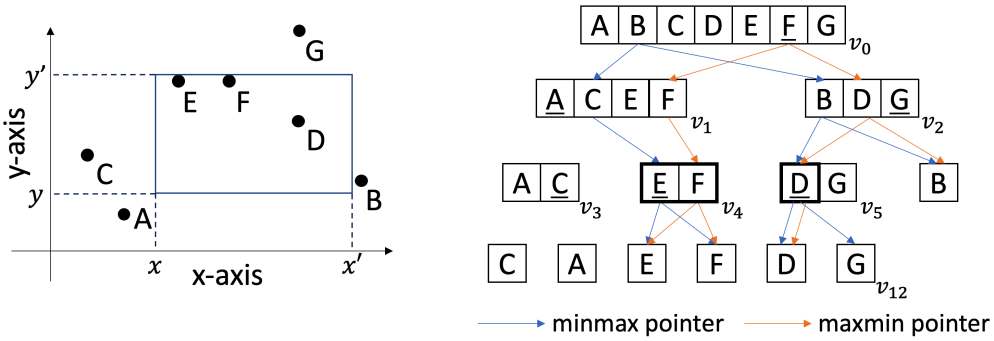

Figure 9: (Left) Two-dimensional point coordinates with a rectangular range query. (Right) Data structure for fractional cascading. The *minmax and maxmin pointers* used for the rectangular range search are only drawn, and the split points are underlined in each vertex. Reported points are bold.

We explain how the FC answers a two-dimensional orthogonal range query $[x, x'] \times [y, y']$. FC implements a binary search on the x-axis, whereas *minmax* and *maxmin pointers* automatically performs y-axis search. We obtain all the points lying in $[y, y']$ at only the root vertex, and *minmax and maxmin pointers* perform the y-axis check for other vertices automatically. Algorithm 3 presents the pseudocode of orthogonal range searching, and an example is drawn in Fig. 9. We first enumerate the candidate points in terms of y-axis at the root node v_0; that is, the points lying in $[y, y']$ are specified (B, C, D, E, and F remain). If the x-coordinate of $v_{0\text{split}} (= F)$ is greater than or equal to x, we continue searching for the left child of v_0 by following *minmax* and *minmax pointers*. Precisely, we follow the *minmax pointer* of the left-end point among the remaining points. In this case, we follow $\textit{minmax}(B, v_1)$ and reach C in v_1. We also follow *maxmin pointer* of the right-end point F in v_0 and reach F in v_1. Notably, each point in v_1 lies in $[y, y']$ if and only if the point is located between C and F (heads of the two pointers). This fact is directly deduced from the definitions of *minmax* and *maxmin pointers* and implies that the y-axis search is sufficient to be performed only at the root. The operation, deciding whether the child needs to be explored and following *minmax* and *minmax pointers*, is performed recursively until all the remaining points are included in $[x, x']$ in the current vertex. With such a situation, we report all these points. For example, all the points of v_4 lie in $[x, x']$; hence, E and F are reported without searching for children of v_4.

Algorithm 2: *constructing a fractional cascading tree*(S)

Input : a set of 2-dimensional data points S
Output: a data structure for fractional cascading

```
Function fctree(list):
    v_list ← list
    v_split ← median point of "list" in lexicographical order (x, y)
    leftlist ← sub-list consisting of all the points less than or equal to v_split in lexicographical order (x, y)
    rightlist ← sub-list consisting of all the points greater than v_split in lexicographical order (x, y)
    if len(leftlist) > 0 then
        v_left ← fctree(leftlist)
    if len(rightlist) > 0 then
        v_right ← fctree(rightlist)
    return v
Function create_fctree(S):
    list ← sorted array of points S in lexicographical order (y, x)
    v_root ← fctree(list)
    return v_root
Function set_pointer(v_root):
    V ← {v_root}
    while V ≠ ∅ do
        v ← V.pop()
        for side ∈ { left, right } do
            if there exists v_side then
                for p ∈ v_list do
                    minmax(p, v_side) ← the smallest element of v_side larger than or equal to p in
                      lexicographical order (y, x) if it exists; otherwise, nil
                    maxmin(p, v_side) ← the largest element of v_side smaller than or equal to p in
                      lexicographical order (y, x) if it exists; otherwise, nil
                V.add(v_side)
Function main(S):
    v_root ← create_fctree(S)
    set_pointer(v_root)
    return v_root
```

7. Experiments

This section first explains the dataset and introduces the parameters used in the experiment. We then compare our TEG-matching with two latest algorithms, namely, the STD-matching (Hsueh and Chen, 2018) and the AntMapper algorithm (Gong et al., 2018), using an open dataset.

7.1. Experiment settings

7.1.1. Dataset

In our experiments, we utilized worldwide vehicle trajectories in a public dataset (Kubička et al., 2015). This dataset includes 100 global GPS *trajectories* (ID = 0, 1, . . . , 99), each of which is associated with the *correct path* and the *road network* around the GPS *trajectory*. The GPS *trajectory* is a sequence of GPS points, each of which consists

Algorithm 3: Two-dimensional search for a rectangular range query using fractional cascading

Input : a query rectangle $[x, x'] \times [y, y']$; an FC tree v_{root}
Output: the set of all the points R which lie in $[x, x'] \times [y, y']$

$S \leftarrow \{v_{\text{root}}\}$
$R \leftarrow \emptyset$
if all points of v_{root} do not lie in $[y, y']$ **then**
    return R
$a_l(v_{\text{root}}) \leftarrow$ the smallest element of v_{root} lying in $[y, y']$
$a_r(v_{\text{root}}) \leftarrow$ the largest element of v_{root} lying in $[y, y']$
while $S \neq \emptyset$ **do**
    `// Pop out an element from a set` S
    $v \leftarrow S$.pop()
    if all points in v from $a_l(v)$ and $a_r(v)$ lies in $[x, x']$ **then**
        all points in v from $a_l(v)$ and $a_r(v)$ are added to R
        continue
    $C \leftarrow \emptyset$
    $sv \leftarrow$ the x coordinate of v_{split}
    if $x \leq sv$ **then**
        add v_{left} to C
    if $sv < x'$ **then**
        add v_{right} to C
    for $v_{child} \in C$ **do**
        $a_l(v_{\text{child}}) \leftarrow \text{minmax}(a_l(v), v_{\text{child}})$
        $a_r(v_{\text{child}}) \leftarrow \text{maxmin}(a_r(v), v_{\text{child}})$
        if $a_l(v_{child}) \neq$ nil and $a_r(v_{child}) \neq$ nil **then**
            `// The most important fact here is that a point of` v_{child} `lies in` $[y, y']$ `if and only if the point is between` $a_l(v_{\text{child}})$ `and` $a_r(v_{\text{child}})$
            add v_{child} to S
return R

of a timestamp and a longitude–latitude pair. The longitude–latitude pair is converted to 2D coordinates using a UTM-WGS84 converter[1]. We excluded data (ID=2,19,33,41,42,69,75,76,80,86, and 89) that the corresponding *correct path* $(e_1, \ldots, e_m)$ is not a path; that is, there exist consecutive edges e_i, e_{i+1} such that $\text{head}(e_i) \neq \text{tail}(e_{i+1})$. The average distance between two consecutive GPS points is 11 m, which is not suitable for evaluating map matching for low-frequency data. Hence, we sample the GPS points from each GPS trajectory by utilizing bottom-up segmentation with *max_error* = 7 m. The thinned-out GPS trajectories are used for evaluating map-matching algorithms. Bottom-up segmentation instead of constant interval sampling avoids overlooking significant direction changes, which is a critical assumption of our TEG-matching. The *max_error* = 7 provides an outline of the vehicle *trajectory* while removing unnecessary position fixes. The number of GPS points and distance between two consecutive GPS points applied and not applied to the bottom-up segmentation are summarized in Table 5.

7.1.2. Preprocess

This section explains the parameters and procedure of the map-matching preprocess explained in Section 3.2. We set the upper bound of the spatial measurement error as $r_{\text{GPS}} = 200$ m and the maximum length of the *shape arc* $\ell_{max} = 2(1 + \sqrt{2})r_{\text{GPS}}$ meter. *Shape nodes* and *junctions* (illustrated in Fig. 1) are identified based on the *road network*

[1] https://github.com/Turbo87/utm

max_error (m)	Original	Bottom-up segmentation			
		3	**7**	15	30
#GPS(%)	209,901 (100%)	17,723 (8.4%)	**10,041 (4.8%)**	6,351 (3.0%)	4,316 (2.1 %)
Distance interval (m)	11 ± 12	135 ± 257	**238 ± 427**	374 ± 572	549 ± 724

Table 5: The total number of GPS points (#GPS) and the distance (m) between two consecutive GPS points applied and not applied bottom-up segmentation over all *trajectories*. "Original" implies the GPS *trajectories* where bottom-up segmentation is not applied. The percentage of #GPS is the ratio to the original *road network*. The distance interval is written as (mean)±(standard deviation). The *trajectories* used for evaluating map-matching algorithms are **bold**.

topology because the original dataset does not distinguish them. We also split long *shape arcs* (add *shape nodes*) such that the length of each *shape arc* is less than or equal to ℓ_{max} to obtain all the *shape arcs* close to a position fix (see details in Section 3.2). For each area associated with a *vehicle trajectory*, we built a fractional cascading (FC) data structure and applied bottom-up segmentation.

7.1.3. Experimental platform

We used only one core of the PC server (2.30 GHz Intel Core E5-2670 with 24 cores and 512 GB of memory). Python 3.8.1 and NetworkX are utilized to calculate the shortest and longest paths. The calculation of the longest path is used for STD-matching that is compared with our TEG-matching.

7.1.4. Evaluation index

We evaluated map-matching algorithms using *arcs*, rather than *shape arcs*, to verify the effectiveness of bottom-up segmentation, which removes redundant *shape nodes* from the *road network*. We compared our approach with existing models in terms of accuracy and speed (#GPS/sec). In our experiment, the accuracy is the intersection over the union of the two multisets of *matched arcs* and *correct arcs*. More precisely, given two multisets of *correct arcs* C_i and *matched arcs* D_i for each *vehicle trajectory* $i \in \{1, \ldots, N\}$, the accuracy is defined as follows:

$$\text{Accuracy for i-th trajectory} = \frac{\#(C_i \cap D_i)}{\#(C_i \cup D_i)} \tag{19}$$

$$\text{Accuracy for dataset} = \frac{\sum_{i=1}^{N} \#(C_i \cap D_i)}{\sum_{i=1}^{N} \#(C_i \cup D_i)} \tag{20}$$

We do not use the average of $\frac{\#(C_i \cap D_i)}{\#(C_i \cup D_i)}$ because the number of position fixes varies per trajectory. For each *vehicle trajectory* $i \in \{1, \ldots, N\}$, let T_i be the processing time and $\#\text{GPS}_i$ be the number of remaining GPS points after bottom-up segmentation. Then, the speed (#GPS/sec) is defined as follows:

$$\text{Speed for i-th trajectory} = \frac{\#\text{GPS}_i}{T_i} \tag{21}$$

$$\text{Speed for dataset} = \frac{\sum_{i=1}^{N} \#\text{GPS}_i}{\sum_{i=1}^{N} T_i} \tag{22}$$

The processing time is defined as the duration from the end of candidate *node* and *arc* search to the output of a *matched path*. We leave out the candidate *node* and *arc* search from the speed because this process is inevitable for all map-matching algorithms.

7.2. Experimental results of fractional cascading

For each GPS point, almost all the matching algorithms must search the *nodes* and *arcs* close to the GPS point; hence, a fast searching speed (#GPS/sec) is vital for map matching. Fractional cascading (FC) accelerates the search for a given rectangular query, as explained in Section 6. We calculated the searching speed of the FC, k-d tree, and brute force for a square query with a side of $2r_{\text{GPS}} = 400$ m while changing the number of *shape nodes* in the data structure. Brute force checks individually whether the query square includes a *node*. Table 6 shows the speed of each

algorithm, using all GPS points. The FC searches the *shape nodes* 2.5x and 195x faster than the k-d tree and brute force search, respectively. The speed and memory usage for different numbers of *shape nodes* are illustrated in Fig. 10 (theoretical values are summarized in Table 4). The speed of FC is faster than that of the kd-tree and brute force for any number of *shape nodes*. However, the speed gradually decreases as the number of *shape nodes* increases, which is compatible with the theoretical query time. Besides, the memory usage of the FC is much larger than that of the kd-tree and brute force. Therefore, if we have enough memory, we should utilize FC; otherwise, the kd-tree is suitable for a candidate *node* search.

	Fractional Cascading	kd-tree	Brute Force
Speed(#GPS/sec)	10,143	3,990	52

Table 6: Speed (#GPS/sec) of fractional cascading, k-d tree, and brute force for square range query with a side of 400 m. This speed is calculated using all the GPS points.

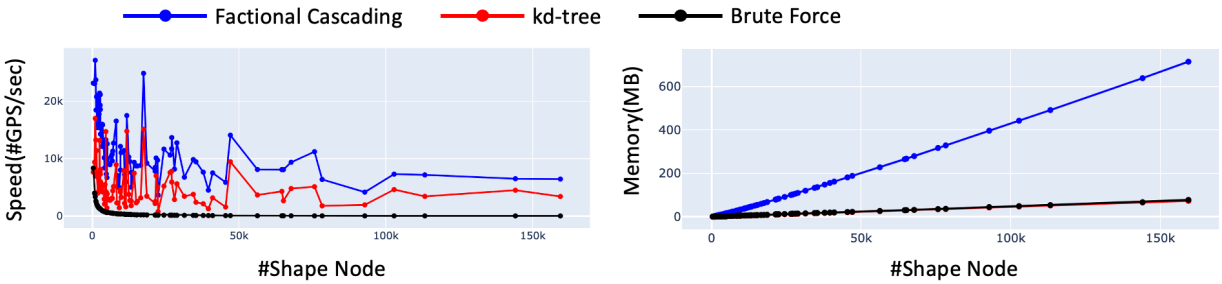

Figure 10: Speed (#GPS/sec) and memory of FC, k-d tree, and brute force for different numbers of *shape nodes*.

7.3. Map-matching models compared to our model

We compare our TEG-matching with STD-matching (Hsueh and Chen, 2018) and the AntMapper algorithm (Gong et al., 2018), and this section provides an overview of these models.

For each position fix, STD-matching lists the candidate *arcs* where a vehicle may be located, and the vehicle location is supposed to be the *projection* of the position fix onto the *arc*. To obtain a *matched path*, STD-matching constructs a graph whose nodes are *projections* and whose arcs are all two *projections* corresponding to two consecutive positions. The weight of each arc is determined based on two factors: (1) the ratio of the shortest path distance of the two *projections* to the distance of the corresponding two position fixes and (2) the distance between the position fix and the *arc*. STD-matching finally finds the most reasonable path from the *projection* of the first position fix to the *projection* of the last position fix.

In contrast, the AntMapper algorithm is non-deterministic; that is, the algorithm may produce a different *matched path* despite that the *trajectory* is the same. Similar to the case with STD-matching, the AntMapper algorithm matches each position fix to an *arc*. However, it computes both the global likelihood of the path and the local likelihood related to two consecutive *projections*. The AntMapper algorithm finally merges the local and global likelihoods, and the highest-value path is explored using the ant colony algorithm.

The parameters of STD-matching and AntMapper algorithms are the same as those used in the corresponding literature. Our dataset only includes timestamps and locations; thus, existing algorithms utilize only this information.

7.4. Comparison with all models

The accuracy, speed, and memory usage of our TEG-matching, STD-matching, and AntMapper algorithms are summarized in Table 7, Fig. 11, and Fig. 12. As shown in Table 7, our TEG-matching is 0.098 higher and 5.6 times faster than the existing models in terms of accuracy and speed, respectively. Significantly, our TEG-matching outperforms the existing models for almost all the data, as illustrated in Fig. 12. TEG-matching achieves sufficient accuracy for almost all data compared to the existing methods, whose accuracies are extremely low for some data.

The prolonged speed of the AntMapper algorithm results from the ant colony algorithm used for computing the global likelihoods. As illustrated in Fig. 11, the peak memory usage of all algorithms has a weak correlation with the number of *shape nodes*, and the most memory-saved algorithm is difficult to determine. Our TEG-matching and the AntMapper algorithm use large amounts of memory for some specific trajectories.

	TEG	STD	AntMapper
Accuracy	**0.9645**	0.8672	0.3568
Speed (#GPS/sec)	**19.3**	3.4	0.2

Table 7: Accuracy and speed (#GPS/sec) of all algorithms for the dataset. The highest accuracy or speed is **bold**.

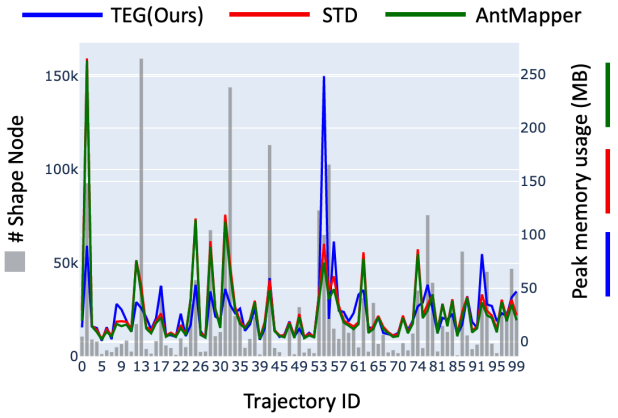

Figure 11: Peak memory usage of all algorithms and the number of *shape nodes* for each *trajectory*. The peak memory usage denotes the required memory except the data structure such as FC and kd-tree.

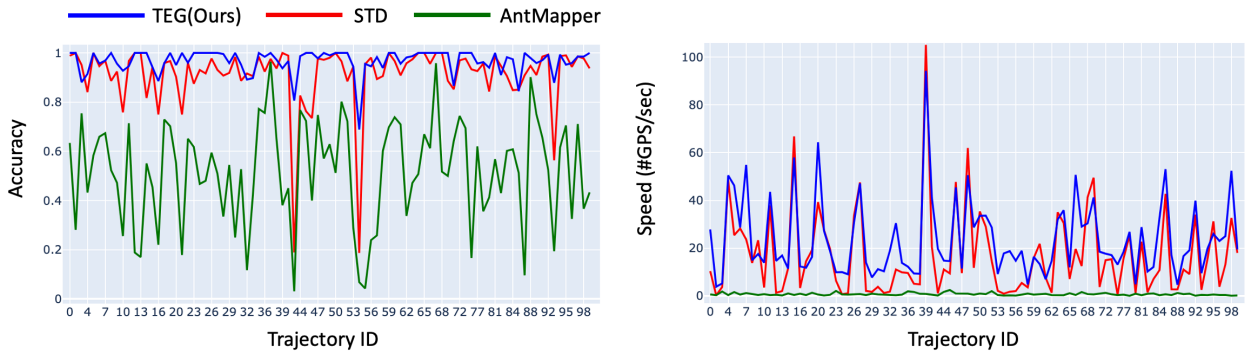

Figure 12: Accuracy and speed (#GPS/sec) of all algorithms for each *trajectory*.

We analyzed the *matched paths* and revealed the advantages of TEG-matching over existing algorithms. We have provided some trajectories for which existing algorithms are unsuccessful, but our TEG-matching is successful in its prediction. We have identified the situation in which our TEG-matching works well and why existing algorithms predict incorrect paths. Another trajectory in which our model fails to predict the *correct path* is also provided.

Area weight of TEG-matching contributes to a correct prediction in Fig. 13. Both STD-matching and AntMapper algorithm match P_{i+2} to *arc* (v_1, v_2) because of a spatial measurement error. In contrast, TEG-matching matches P_{i+2} to *arc* (v_1, v_3) by considering the area between the *trajectory* and a *matched path*. The area becomes small if we match P_{i+2} to *arc* (v_1, v_3) rather than match P_{i+2} to *arcs* (v_1, v_2) and (v_2, v_3). Moreover, the angle difference of segment between (P_{i+2}, P_{i+3}) and P_{i+2}'s matched *arc* becomes large if we match P_{i+2} to (v_1, v_2) and (v_2, v_3) compared to matching P_{i+2} to (v_1, v_3). Therefore, *area weight* of the *correct path* becomes smaller than that of an incorrect path (*matched path* of STD-matching or AntMapper), resulting in the accurate prediction of TEG-matching.

The trajectory (ID=21) shown in Fig. 14 has a spatial measurement error at the starting position (P_1 and P_2). The STD-matching and AntMapper predict the wrong U-turn paths because the direction between the correct *arc* and the segment (P_1, P_2) is the opposite. The *direction change weight* of our TEG-matching avoids U-turns and helps obtain the *correct path*.

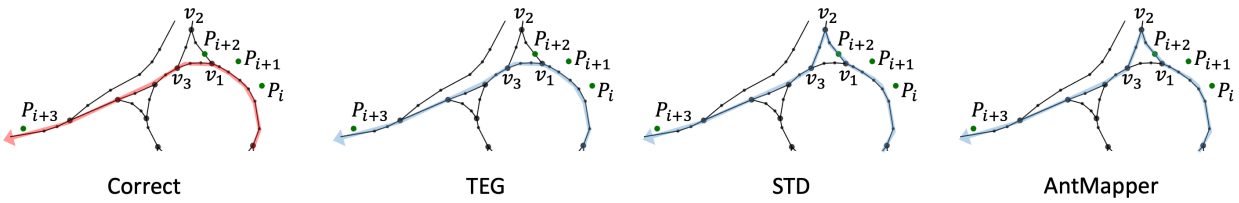

Figure 13: *Shape nodes* (small black dots), *junctions* (big black dots), a *vehicle trajectory* (green marks), and *matched paths* (sky blue) of TEG-matching, STD-matching, and AntMapper, as well as the correct path (red).

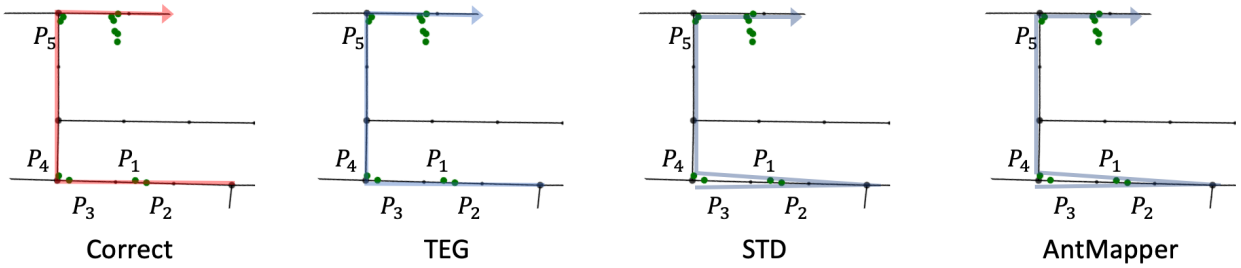

Figure 14: *Shape nodes* (small black dots), *junctions* (big black dots), a *vehicle trajectory* (green marks), and *matched paths* (sky blue) of TEG-matching, STD-matching, and AntMapper, as well as the correct path (red).

Fig. 15 (ID=21) is the example where the STD-matching and AntMapper present *matched paths* that go back and forth when the vehicle pauses. STD-matching matches two consecutive position fixes with two points such that the distance between the two position fixes is close to the shortest path distance of the corresponding two *projections*. $P_{i+1} \sim P_{i+6}$ are slightly different from each other, which causes an otiose detour for STD-matching. The inconsistent directions of two consecutive position fixes provide AntMapper with a round-trip path. The *direction change weight* of our TEG-matching helps remove these unnecessary detours.

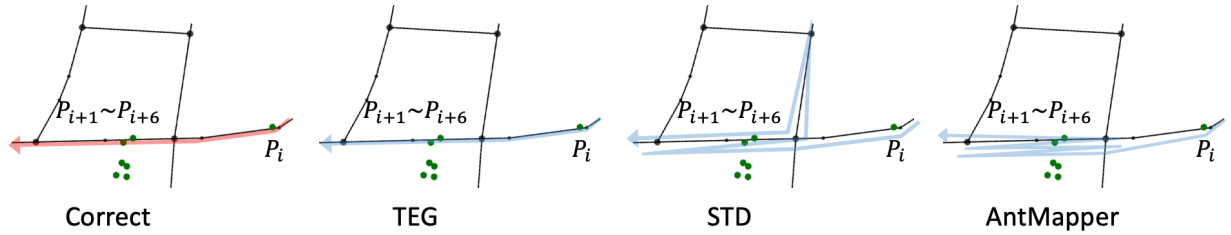

Figure 15: *Shape nodes* (small black dots), junctions (big black dots), vehicle trajectory (green marks) and *matched paths* (sky blue) of TEG-matching, STD-matching and AntMapper, as well as the correct path (red).

TEG-matching fails to predict the *matched path* for the trajectory (ID=95) shown in Fig. 16. The vehicle makes a U-turn in the middle of the long *arc* (v_1, v_2), and the "long" *arc* causes a mistake in the prediction. If our model matches (P_{i+2}, P_{i+3}) and (P_{i+3}, P_{i+4}) with *arcs* (v_1, v_2) and (v_2, v_1) respectively, the *direction change weight* $w_d\left((v_1, v_2)^{i+3}, (v_2, v_1)^{i+3}\right)$ becomes very large because P_{i+2}, P_{i+3} and P_{i+4} are far from v_2. Therefore our models selects the shortcut path $(\ldots, (v_1, v_3), (v_3, v_4), (v_4, v_3), \ldots)$. To avoid this situation, we need to split long *arcs* at direction change *shape nodes*. AntMapper is a non-deterministic algorithm that offers an incorrect path containing two round-trips, and STD-matching yields the correct *matched path*.

In summary, our TEG-matching is robust against spatial measurement errors and pauses, such as traffic lights. However, if the vehicle makes a U-turn in the middle of a long *arc*, the TEG-matching outputs an incorrect shortcut path. We can solve this problem by splitting long *arcs* at direction change *shape nodes*.

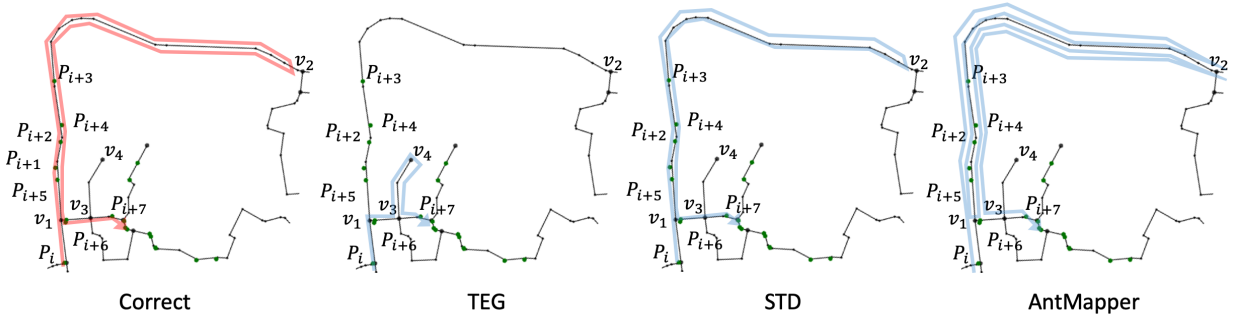

Figure 16: *Shape nodes* (small black dots), *junctions* (big black dots), a vehicle *trajectory* (green marks), and *matched paths* (sky blue) of TEG-matching, STD-matching, and AntMapper, as well as the correct path (red).

7.5. Impact of bottom-up segmentation

The bottom-up segmentation explained in Section 5 reduces the graph size while preserving the topology of a *road network*. Table 8 shows the number of *shape nodes*, speed, and accuracy of our TEG-matching applied and not applied bottom-up segmentation. In the table, "Original" implies not applying bottom-up segmentation to the *road network*. Surprisingly, a tiny parameter value (max_error = 3) achieves approximately 50% reduction in *shape nodes*, which implies that we succeed in significant node reduction while preserving the shape of the original *road network*. We achieved a 1.78x speed increase with only a 0.0074 accuracy drop at the max_error=10, where the accuracy 0.9571 is still higher than the existing models (STD:0.8672 and AntMapper:0.3568). The accuracy and speed for each *trajectory* are shown in Fig. 17 and Fig. 18, respectively. Bottom-up segmentation achieves an effective map-matching speedup with only a small accuracy drop.

max_error (meter)	Original	Bottom-up segmentation			
		3	5	10	30
#ShapeNode(%)	1,995,882 (100%)	1,045,355 (52%)	890,390 (45%)	724,726 (36%)	553,260 (28 %)
Accuracy	**0.9645**	0.9622	0.9619	0.9571	0.9469
Speed (#GPS/sec)	19.3	30.9	32.0	34.3	**44.20**

Table 8: Number of *shape nodes*, accuracy and speed of our TEG-matching applied and not applied *bottom-up segmentation*. The *shape nodes* covers areas associated with all GPS trajectories. "Original" implies that bottom-up segmentation is not applied to the *road network*. The percentage is the ratio of the *shape node* number to the original one. The highest accuracy and speed are **bold**.

Fig. 19 shows the vehicle *trajectory* whereby bottom-up segmentation has a negative influence on the accuracy. Bottom-up segmentation brings the *correct arc* away for the position fix P_i; hence, the *matched path* is wrong. The appropriate max_error is the key to balancing the accuracy and speed of map matching. In contrast, bottom-up segmentation occasionally improves the accuracy, as shown in Fig. 20. Our TEG-matching produces a shortcut path with a U-turn (yellow *arc*) for the original *road network*. The long purple *arc* is unlikely to be chosen because the long *arc* increases the *area weight*. Besides, P_{i+1} and P_{i+2} achieve perpendicular distances on the yellow *arc*, which reduces the *area weight* of the *arc*. After bottom-up segmentation, P_{i+2} does not achieve a perpendicular distance on the yellow *arc*; therefore, TEG-matching is successful in prediction.

8. Conclusion

We propose a parameter-free map-matching algorithm called TEG-matching. TEG-matching achieves a 0.098 accuracy improvement and 5.6x speedup than existing models. TEG-matching constructs a time-dependent graph and solves the shortest path problem on the graph to obtain the most plausible space-time path. Numerical experiments indicate that our TEG-matching is robust against spatial measurement errors and pauses such as traffic lights. However, an appropriate *arc* split is required for further accuracy improvement. We also performed a theoretical analysis and

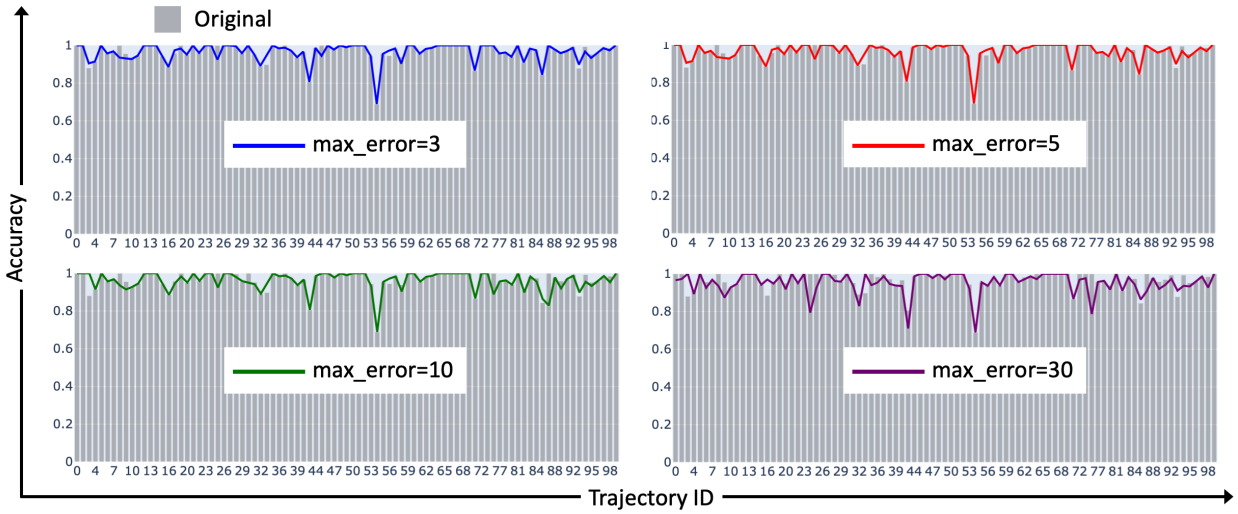

Figure 17: Accuracy of our TEG-matching for each *trajectory* while changing the max_error of the bottom-up segmentation. "Original" implies that bottom-up segmentation is not applied to the *road network*.

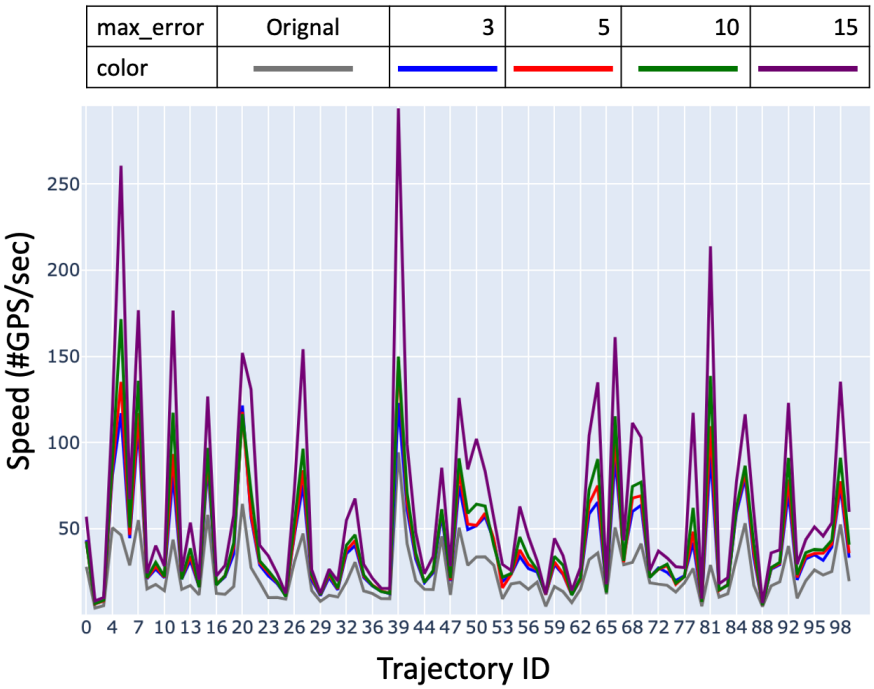

Figure 18: Speed (#GPS/sec) of our TEG-matching for each *trajectory* while changing the max_error of bottom-up segmentation. "Original" implies that bottom-up segmentation is not applied to the *road network*.

determined how large a square is needed to obtain all the *arcs* within a radius r from a certain point. Moreover, we utilized the data structure FC to achieve a high-speed neighborhood search. Bottom-up segmentation also achieves a 64% reduction in *shape nodes*, resulting in a 1.78x speed increase with only 0.0074 accuracy reduction for map matching.

The proposed algorithm is promising for offline usage in ITS, such as traffic dynamics analysis and urban planning to alleviate traffic congestion. The traffic dynamics analysis applies data mining methods to understand spatial and

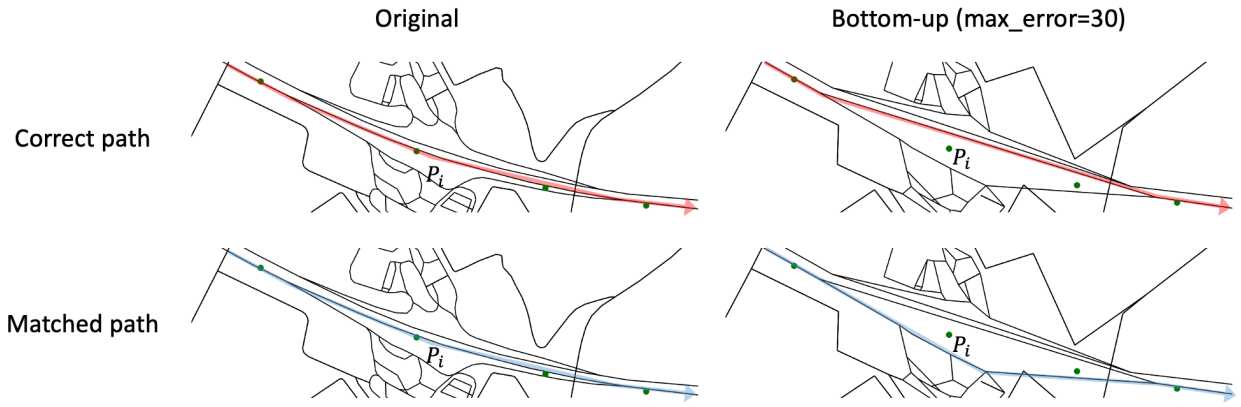

Figure 19: Vehicle trajectory (green marks), *correct path* (red), and *matched paths* (sky blue) of TEG-matching applied and not applied bottom-up segmentation, as well as the *correct path* (red).

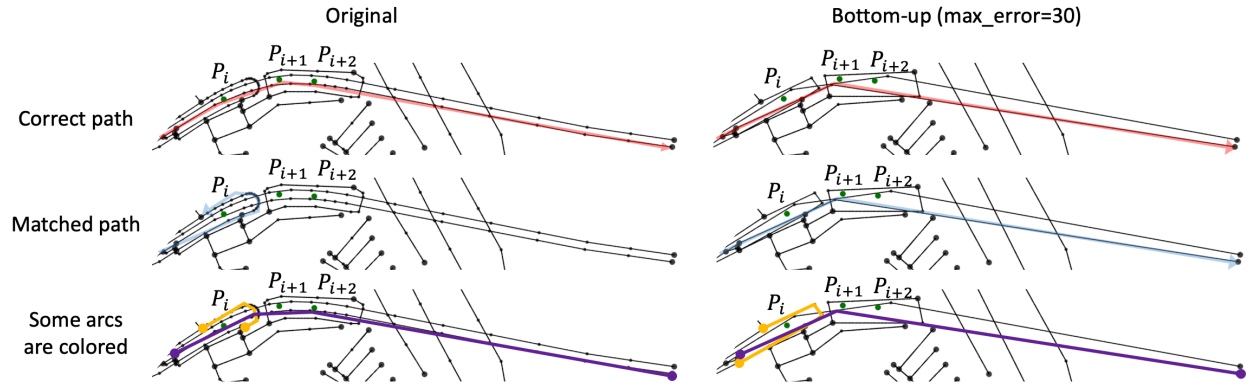

Figure 20: Vehicle trajectory (green marks), *correct path* (red), and *matched paths* (sky blue) of TEG-matching applied and not applied bottom-up segmentation, as well as some colored *arcs*.

social behavior such as travelers' route choice, accessibility patterns, and commercial center attractiveness. Advantageous properties of our algorithm are (1) high speed and accuracy for low-frequency data, (2) being parameter-free, and (3) only requiring ordered locations for map matching, which are highly beneficial to a practical case that requires high performance and reduces the cost of data transmission and tuning hyperparameters.

Acknowledgment

This research project was supported by the Japan Science and Technology Agency (JST), Core Research of Evolutionary Science and Technology (CREST), the Center of Innovation Science and Technology based Radical Innovation and Entrepreneurship Program (COI Program), JSPS KAKENHI Grant No. JP 16H01707.

Appendix A. Symbols used

The symbols for which the scope is not limited to a small context (a particular sentence, paragraph, or non-subdivided section) are summarized in Table A.9.

Appendix B. Proof

We utilize square query to find all *arcs* near a vehicle *trajectory*. To this end, we repeatedly obtain all *shape arcs* contained in a square. This section answers how large the square is required to obtain either of the endpoints of the

Symbol	Definition
V	The set of *junctions* and *shape nodes* (see Fig. 1)
A	The set of *arcs* (see Fig. 1))
G	The *road network*
$\boldsymbol{P} = (P_i)_{i=1}^{n+1}$	Vehicle *trajectory* from time stamp i to $i+1$
$T(G, \boldsymbol{P})$	Time-expanded graph (TEG) corresponding to vehicle *trajectory* $\boldsymbol{P}$
$d(x, y)$	Euclidean distance between x and y
$\#A$	Number of elements in set A
$\min\{x_1, x_2, \ldots, x_n\}$	The minimum value of $x_1, \ldots, x_n$ $(x_i \in \mathbb{R})$
∂A	Boundary of set $A \subset \mathbb{R}^n$
a^T, A^T	Transpose of row vector a, matrix A
$\|x\|_p$	ℓ^p norm of vector $x = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$, that is, $\|x\|_p := \left(\sum_{i=1}^n \|x_i\|^p\right)^{1/p}$
$\|x\|_\infty$	The maximum norm of vector $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, that is, $\|x\|_\infty := \max\{\|x_1\|, \|x_2\| \ldots, \|x_n\|\}$

Table A.9: Notations

shape arc located within a radius r meter from a point. Theorem 2 presents the side length of the square, and both Lemma 1 and Lemma 2 are used to prove the Theorem 2 .

Lemma 1. *Let* $0 < r \le s$, $\partial C := \left\{\boldsymbol{x} \in \mathbb{R}^2 \mid \|\boldsymbol{x}\|_2 = r\right\}$, *and* $\partial L := \left\{\boldsymbol{x} \in \mathbb{R}^2 \mid \|\mathbf{x}\|_\infty = s\right\}$. *Then we have*

$$\min_{(x,y)\in\partial C} l(x, y) = \min\left\{2\left(\sqrt{2}s - r\right), 2s\right\}$$

where $l(x, y)$ *is the distance between the two intersection points between* ∂L *and the tangent line at* (x, y) *of* ∂C. *If* $s = r$ *and the tangent line is either* $x = r$, $x = -r$, $y = r$, *or* $y = -r$, *the intersection points are infinite. In this case, we define* $l(x, y) = 2s$, *which is compatible with this lemma.*

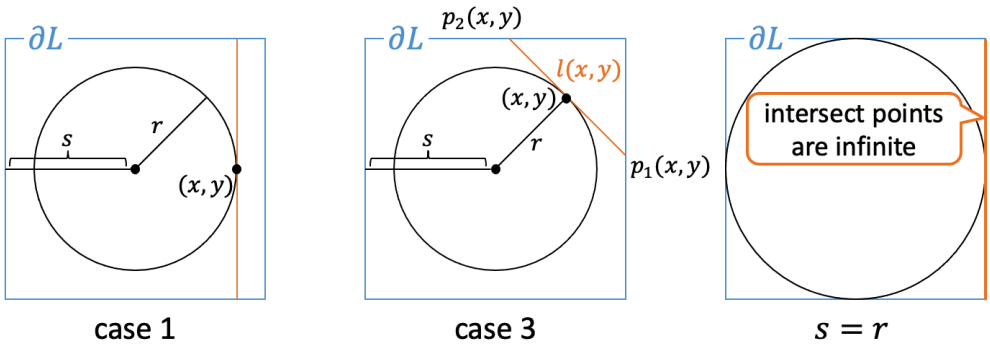

Figure B.21: Visualization of the symbols used in the lemma.

Proof. By symmetry, we only consider the point of tangency at $0 \le x \le r$, $0 \le y \le r$. Given the circle with the center at the origin and radius of r, the tangent line at $(x_0, y_0)^T \in \mathbb{R}^2$ is given by the following equation:

$$x_0 x + y_0 y = r^2 \tag{B.1}$$

Hence, the intersection conforms to any of the following three cases:
(1) The tangent line intersects ∂L at $y = s$ and $y = -s$.
(2) The tangent line intersects ∂L at $x = s$ and $x = -s$.
(3) The tangent line intersects ∂L at $x = s$ and $y = s$.

In cases (1) and (2), $l(x, y)$ is parallel to either the x-axis or y-axis and $l(x, y) = 2s$; hence, we only consider case (3) in the following proof. Because the tangent point is neither $x = 0$ nor $y = 0$, the tangent line at $(x, y)^T$ $(0 < x, y < r)$ intersects ∂L at the following two points:

$$p_1(x, y) = \left(s, \frac{r^2 - sx}{y}\right)^T, p_2(x, y) = \left(\frac{r^2 - sy}{x}, s\right)^T \tag{B.2}$$

By using polar coordinates, the distance between $p_1(x, y)$ and $p_2(x, y)$, denoted by $l(x, y)$, is expressed by

$$\begin{aligned}
(l(x, y))^2 &= \|p_1(x, y) - p_2(x, y)\|_2^2 && \text{(B.3)}\\
&= \left(\frac{r^2 - s(x + y)}{x}\right)^2 + \left(\frac{r^2 - s(x + y)}{y}\right)^2 && \text{(B.4)}\\
&= \frac{(r^2 - s(x + y))^2(x^2 + y^2)}{x^2y^2} && \text{(B.5)}\\
&= \frac{r^2(r^2 - sr(\cos\theta + \sin\theta))^2}{r^4 \sin^2\theta \cos^2\theta} && \text{(B.6)}\\
&= \frac{(r - s(\cos\theta + \sin\theta))^2}{\sin^2\theta \cos^2\theta} && \text{(B.7)}\\
&\equiv f(\theta), && \text{(B.8)}
\end{aligned}$$

where

$$x = r\cos\theta, y = r\sin\theta \tag{B.9}$$

Because $l(x, y)$ is non-negative, the problem is equivalent to finding the minimum value of $f(\theta)$ on $\theta \in \left(0, \frac{\pi}{2}\right)$. We perform the first derivative test and find the global minimum value. Because

$$(\sin^2\theta \cos^2\theta)' = 2\sin\theta\cos\theta(\cos^2\theta - \sin^2\theta), \tag{B.10}$$

the derivative of f, denoted by f', agrees with the following equation:

$$\begin{aligned}
f'(\theta) &= \frac{2\sin\theta\cos\theta(r - s(\sin\theta + \cos\theta))(\cos\theta - \sin\theta)\{-s\sin\theta\cos\theta - (r - s(\sin\theta + \cos\theta))(\cos\theta + \sin\theta)\}}{\sin^4\theta\cos^4\theta} && \text{(B.11)}\\
&= \frac{2(r - s(\sin\theta + \cos\theta))\,(s(1 + \sin\theta\cos\theta) - r(\sin\theta + \cos\theta))\,(\cos\theta - \sin\theta)}{\sin^3\theta\cos^3\theta}. && \text{(B.12)}
\end{aligned}$$

We first evaluate $r - s(\sin\theta + \cos\theta)$. Because $\sin\theta + \cos\theta = \sqrt{2}\sin\left(\theta + \frac{\pi}{4}\right)$ and $0 < \theta < \frac{\pi}{2}$,

$$1 < \sin\theta + \cos\theta \leq \sqrt{2} \quad \left(\frac{\pi}{4} < \theta + \frac{\pi}{4} < \frac{3}{4}\pi\right) \tag{B.13}$$

Hence,

$$r - s(\sin\theta + \cos\theta) < r - s \leq 0 \tag{B.14}$$

Next, we handle $s(1 + \sin\theta\cos\theta) - r(\sin\theta + \cos\theta)$ by transforming the equation as follows:

$$s(1 + \sin\theta\cos\theta) - r(\sin\theta + \cos\theta) \geq s\,(1 + \sin\theta\cos\theta - (\sin\theta + \cos\theta)) = s(1 - \sin\theta)(1 - \cos\theta) \geq 0 \tag{B.15}$$

We finally consider $\cos\theta - \sin\theta$. Because $\cos\theta - \sin\theta = -\sqrt{2}\sin\left(\theta - \frac{\pi}{4}\right)$ and $-\frac{\pi}{4} < \theta - \frac{\pi}{4} < \frac{\pi}{4}$ when $0 < \theta < \frac{\pi}{2}$, we have

$$-\frac{\pi}{4} < \theta - \frac{\pi}{4} < 0 \quad \left(0 < \theta < \frac{\pi}{4}\right) \Rightarrow \cos\theta - \sin\theta > 0 \tag{B.16}$$

$$0 \leq \theta - \frac{\pi}{4} < \frac{\pi}{4} \quad \left(\frac{\pi}{4} \leq \theta < \frac{\pi}{2}\right) \Rightarrow \cos\theta - \sin\theta \leq 0 \tag{B.17}$$

From the above discussions, we conclude as follows:

$$0 < \theta < \frac{\pi}{4} \Rightarrow f'(\theta) \leq 0 \tag{B.18}$$

$$\frac{\pi}{4} \leq \theta < \frac{\pi}{2} \Rightarrow f'(\theta) \geq 0 \tag{B.19}$$

Therefore, $f(\theta)$ has a minimum value $4\left(\sqrt{2}s - r\right)^2$ at $\theta = \frac{\pi}{4}$ on $\left(0, \frac{\pi}{2}\right)$, implying that $l(x, y)$ finds the minimum value $\sqrt{4\left(\sqrt{2}s - r\right)^2} = 2\left(\sqrt{2}s - r\right)$ at $|x| = |y| = \frac{r}{\sqrt{2}}$.

Lemma 2. *Let $0 < r \leq s$ and $L \coloneqq \{\boldsymbol{x} \in \mathbb{R}^2 \mid \|\boldsymbol{x}\|_\infty \leq s\}$. Any line intersecting $C \coloneqq \{\boldsymbol{x} \in \mathbb{R}^2 \mid \|\boldsymbol{x}\|_2 \leq r\}$ has exactly two intersection points with ∂L, and the distance between the two intersection points has the minimum value $\min\{2(\sqrt{2}s - r), 2s\}$. If $s = r$ and the line is either $x = r$, $x = -r$, $y = r$, or $y = -r$, the intersection points are infinite. In this case, we define the distance as $2s$, which is compatible with this lemma.*

Proof. We divide this problem into three cases according to the conditions of the line. Here, we define r' as the distance between the line and the origin.
(1) $r' = 0$
The distance has a minimum value of $2s$ when the line is parallel to the x-axis or y-axis.
(2) $0 < r' \leq r$
Any line not going through the origin is regarded as the tangent line of the origin-centered circle with a radius of r'; hence, we denote the tangent point by $(x, y)^T$ ($\|(x, y)^T\|_2 = r'$). Let $l(x, y)$ be the distance between the two intersection points between the tangent line and ∂L. From Lemma 1, $l(x, y)$ has a minimum value of $\min\{2(\sqrt{2}s - r'), 2s\}$.
From (1) and (2), we can conclude that the distance has a minimum value of $\min\{2(\sqrt{2}s - r), 2s\}$.

Theorem 2. *Consider the road network whose shape arc is represented as a straight segment. Let $0 < r \leq s$, $\ell_{\max} > 0$ be the maximum length of the shape arc that satisfies $\ell_{\max} \leq \min\{2\left(\sqrt{2}s - r\right), 2s\}$. Then, for the shape arc (u, v) and any point P, we have*

$$d\left((u, v), P\right) \leq r \Rightarrow \min\left\{\|u - P\|_\infty, \|v - P\|_\infty\right\} \leq s$$

. Moreover, if $\ell_{\max} \leq 2\left(1 + \sqrt{2}\right)r$, then $\ell_{\max} \leq \min\left\{2\left(\sqrt{2}s - r\right), 2s\right\}$ is equivalent to $\frac{\ell_{\max} + 2r}{2\sqrt{2}} \leq s$. Otherwise, $\frac{\ell_{\max}}{2} \leq s$.

Proof. Without losing generality, we may assume that P is located at the origin by shifting the road network in parallel. Except for the equivalent condition, the claim is directly induced from Lemma 2; hence, we prove the equivalent condition by considering the following two cases:
(1) $2\left(\sqrt{2}s - r\right) \leq 2s$
(2) $2\left(\sqrt{2}s - r\right) \geq 2s$
In case (1), as $\min\left\{2\left(\sqrt{2}s - r\right), 2s\right\} = 2\left(\sqrt{2}s - r\right)$,

$$\ell_{\max} \leq \min\left\{2\left(\sqrt{2}s - r\right), 2s\right\} \Leftrightarrow \frac{\ell_{\max} + 2r}{2\sqrt{2}} \leq s \tag{B.20}$$

. In case (2), as $\min\left\{2\left(\sqrt{2}s - r\right), 2s\right\} = 2s$,

$$\ell_{\max} \leq \min\left\{2\left(\sqrt{2}s - r\right), 2s\right\} \Leftrightarrow \frac{\ell_{\max}}{2} \leq s \tag{B.21}$$

On the other hand, performing a simple transformation, we have

$$2\left(\sqrt{2}s - r\right) \leq 2s \Leftrightarrow s \leq \left(\sqrt{2} + 1\right)r \tag{B.22}$$

Therefore

$$\min\left\{2\left(\sqrt{2}s - r\right), 2s\right\} \Leftrightarrow \left(\frac{\ell_{\max} + 2r}{2\sqrt{2}} \leq s \leq \left(\sqrt{2} + 1\right)r\right) \vee \left(\max\left\{\frac{\ell_{\max}}{2}, \left(\sqrt{2} + 1\right)r\right\} \leq s\right) \tag{B.23}$$

If $\ell_{\max} \leq 2\left(\sqrt{2}+1\right)r$, $\max\left\{\frac{\ell_{\max}}{2}, \left(\sqrt{2}+1\right)r\right\} = \left(\sqrt{2}+1\right)r$. Therefore, if we are concerned about

$$\frac{\ell_{\max}+2r}{2\sqrt{2}} \leq \left(\sqrt{2}+1\right)r \Leftrightarrow \ell_{\max} \leq 2\left(\sqrt{2}+1\right)r \tag{B.24}$$

, we have

$$\left(\frac{\ell_{\max}+2r}{2\sqrt{2}} \leq s \leq \left(\sqrt{2}+1\right)r\right) \vee \left(\max\left\{\frac{\ell_{\max}}{2}, \left(\sqrt{2}+1\right)r\right\} \leq s\right) \tag{B.25}$$

$$\Leftrightarrow \left(\frac{\ell_{\max}+2r}{2\sqrt{2}} \leq s \leq \left(\sqrt{2}+1\right)r\right) \vee \left(\left(\sqrt{2}+1\right)r \leq s\right) \tag{B.26}$$

$$\Leftrightarrow \left(\frac{\ell_{\max}+2r}{2\sqrt{2}} \leq s \vee \left(\sqrt{2}+1\right)r \leq s\right) \wedge \left(s \leq \left(\sqrt{2}+1\right)r \vee \left(\sqrt{2}+1\right)r \leq s\right) \tag{B.27}$$

$$\Leftrightarrow \min\left\{\frac{\ell_{\max}+2r}{2\sqrt{2}}, \left(\sqrt{2}+1\right)r\right\} \leq s \tag{B.28}$$

$$\Leftrightarrow \frac{\ell_{\max}+2r}{2\sqrt{2}} \leq s \tag{B.29}$$

. Otherwise, if $\ell_{\max} > 2\left(\sqrt{2}+1\right)r$, we have $\frac{\ell_{\max}+2r}{2\sqrt{2}} > \left(\sqrt{2}+1\right)r$ from Eq.B.24. Therefore, we have

$$\left(\frac{\ell_{\max}+2r}{2\sqrt{2}} \leq s \leq \left(\sqrt{2}+1\right)r\right) \vee \left(\max\left\{\frac{\ell_{\max}}{2}, \left(\sqrt{2}+1\right)r\right\} \leq s\right) \tag{B.30}$$

$$\Leftrightarrow \max\left\{\frac{\ell_{\max}}{2}, \left(\sqrt{2}+1\right)r\right\} \leq s \tag{B.31}$$

$$\Leftrightarrow \frac{\ell_{\max}}{2} \leq s \tag{B.32}$$

.

References

Alt, H., Efrat, A., Rote, G., Wenk, C., 2003. Matching planar maps. J. Algorithms 49, 262–283. doi:10.1016/S0196-6774(03)00085-3.

Bentley, J.L., 1975. Multidimensional Binary Search Trees Used for Associative Searching. Commun. ACM 18, 509–517. doi:10.1145/361002.361007.

Bonnifait, P., Laneurit, J., 2009. Multi-hypothesis Map-Matching using Particle Filtering To cite this version :, in: 16th World Congr. ITS Syst. Serv. HAL, pp. pp. 1–8.

Chao, P., Xu, Y., Hua, W., Zhou, X., 2020. A Survey on Map-Matching Algorithms, in: Borovica-Gajic, R., Qi, J., Wang, W. (Eds.), Databases Theory Appl., Springer International Publishing, Cham. pp. 121–133.

Chazelle, B., Guibas, L.J., 1986. Fractional cascading: I. A data structuring technique. Algorithmica 1, 133–162. URL: https://doi.org/10.1007/BF01840440, doi:10.1007/BF01840440.

Chen, B.Y., Yuan, H., Li, Q., Lam, W.H., Shaw, S.L., Yan, K., 2014. Map-matching algorithm for large-scale low-frequency floating car data. Int. J. Geogr. Inf. Sci. 28, 22–38. URL: http://dx.doi.org/10.1080/13658816.2013.816427, doi:10.1080/13658816.2013.816427.

DOUGLAS, D.H., PEUCKER, T.K., 1973. ALGORITHMS FOR THE REDUCTION OF THE NUMBER OF POINTS REQUIRED TO REPRESENT A DIGITIZED LINE OR ITS CARICATURE. Cartogr. Int. J. Geogr. Inf. Geovisualization 10, 112–122. URL: https://utpjournals.press/doi/10.3138/FM57-6770-U75U-7727, doi:10.3138/FM57-6770-U75U-7727.

Goh, C.Y., Dauwels, J., Mitrovic, N., Asif, M.T., Oran, A., Jaillet, P., 2012. Online map-matching based on hidden Markov model for real-time traffic sensing applications. IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC 117543, 776–781. doi:10.1109/ITSC.2012.6338627.

Gong, Y.J., Chen, E., Zhang, X., Ni, L.M., Zhang, J., 2018. AntMapper: An Ant Colony-Based Map Matching Approach for Trajectory-Based Applications. IEEE Trans. Intell. Transp. Syst. 19, 390–401. doi:10.1109/TITS.2017.2697439.

Hsueh, Y.L., Chen, H.C., 2018. Map matching for low-sampling-rate GPS trajectories by exploring real-time moving directions. Inf. Sci. (Ny). 433-434, 55–69. URL: https://doi.org/10.1016/j.ins.2017.12.031, doi:10.1016/j.ins.2017.12.031.

Hu, G., Shao, J., Liu, F., Wang, Y., Shen, H.T., 2017. IF-Matching: Towards Accurate Map-Matching with Information Fusion. IEEE Trans. Knowl. Data Eng. 29, 114–127. doi:10.1109/TKDE.2016.2617326.

Huang, Z., Qiao, S., Han, N., an Yuan, C., Song, X., Xiao, Y., 2021. Survey on vehicle map matching techniques. CAAI Trans. Intell. Technol. 6, 55–71. doi:10.1049/cit2.12030.

Hunter, T., Abbeel, P., Bayen, A., 2014. The path inference filter: Model-based low-latency map matching of probe vehicle data. IEEE Trans. Intell. Transp. Syst. 15, 507–529. doi:10.1109/TITS.2013.2282352, arXiv:1109.1966.

Keogh, E., Chu, S., Hart, D., Pazzani, M., 2002. An online algorithm for segmenting time series , 289–296doi:10.1109/icdm.2001.989531.
Keogh, E., Pazzani, M., 1998. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. Kdd 98, 239–243. URL: http://www.aaai.org/Papers/KDD/1998/KDD98-041.pdf, doi:10.1.1.42.1358.
Knapen, L., Bellemans, T., Janssens, D., Wets, G., 2018. Likelihood-based offline map matching of GPS recordings using global trace information. Transp. Res. Part C Emerg. Technol. 93, 13–35. URL: https://doi.org/10.1016/j.trc.2018.05.014, doi:10.1016/j.trc.2018.05.014.
Koski, A., Juhola, M., Meriste, M., 1995. Syntactic recognition of ECG signals by attributed finite automata. Pattern Recognit. 28, 1927–1940. doi:10.1016/0031-3203(95)00052-6.
Kubička, M., Cela, A., Moulin, P., Mounier, H., Niculescu, S.I., 2015. Dataset for testing and training of map-matching algorithms, in: 2015 IEEE Intell. Veh. Symp., IEEE. pp. 1088–1093.
Luebke, D.P., 2001. A Survey of Polygonal Simplification Algorithms. Comput. Graph. Appl. IEEE 21, 24–35. doi:10.1109/38.920624.
Newson, P., Krumm, J., 2009. Hidden Markov map matching through noise and sparseness. GIS Proc. ACM Int. Symp. Adv. Geogr. Inf. Syst. , 336–343doi:10.1145/1653771.1653818.
Park, S., Kim, S.W., Chu, W.W., 2001. Segment-based Approach for Subsequence Searches in Sequence Databases, in: Proc. 2001 ACM Symp. Appl. Comput., ACM, New York, NY, USA. pp. 248–252. URL: http://doi.acm.org/10.1145/372202.372334, doi:10.1145/372202.372334.
Park, S., Lee, D., Chu, W.W., 1999. Fast retrieval of similar subsequences in long sequence databases, in: Proc. 1999 Work. Knowl. Data Eng. Exch. (Cat. No.PR00453), pp. 60–67. doi:10.1109/KDEX.1999.836610.
Quddus, M., Washington, S., 2015. Shortest path and vehicle trajectory aided map-matching for low frequency GPS data. Transp. Res. Part C Emerg. Technol. 55, 328–339. URL: http://dx.doi.org/10.1016/j.trc.2015.02.017, doi:10.1016/j.trc.2015.02.017.
Ray-Chaudhuri, D.K., 1967. Characterization of line graphs. J. Comb. Theory 3, 201–214. doi:10.1016/S0021-9800(67)80068-1.
Sathiaseelan, A., 2011. The Role of Location Based Technologies in Intelligent Transportation Systems. Asian J. Inf. Technol. 10, 227–233. doi:10.3923/ajit.2011.227.233.
Taguchi, S., Koide, S., Yoshimura, T., 2019. Online Map Matching with Route Prediction. IEEE Trans. Intell. Transp. Syst. 20, 338–347. doi:10.1109/TITS.2018.2812147.
Tang, J., Song, Y., Miller, H.J., Zhou, X., 2016. Estimating the most likely space–time paths, dwell times and path uncertainties from vehicle trajectory data: A time geographic method. Transp. Res. Part C Emerg. Technol. 66, 176–194. URL: https://www.sciencedirect.com/science/article/pii/S0968090X15003150, doi:10.1016/J.TRC.2015.08.014.
Toledo-Moreo, R., Betaille, D., Peyret, F., 2010. Lane-Level Integrity Provision for Navigation and Map Matching With GNSS, Dead Reckoning, and Enhanced Maps. IEEE Trans. Intell. Transp. Syst. 11, 100–112. doi:10.1109/TITS.2009.2031625.
Velaga, N.R., Pangbourne, K., 2014. Achieving genuinely dynamic road user charging: Issues with a GNSS-based approach. J. Transp. Geogr. 34, 243–253. URL: http://dx.doi.org/10.1016/j.jtrangeo.2013.09.013, doi:10.1016/j.jtrangeo.2013.09.013.
Wang, X., Ni, W., 2016. An improved particle filter and its application to an INS/GPS integrated navigation system in a serious noisy scenario. Meas. Sci. Technol. 27. doi:10.1088/0957-0233/27/9/095005.
Wei, H., Wang, Y., Forman, G., Zhu, Y., 2013. Map matching by Fréchet distance and global weight optimization. Dep. Comput. Sci. . . . .
White, C.E., Bernstein, D., Kornhauser, A.L., 2000. Some map matching algorithms for personal navigation assistants. Transp. Res. Part C Emerg. Technol. 8, 91–108. doi:10.1016/S0968-090X(00)00026-7.
Yang, X., Tang, L., Niu, L., Zhang, X., Li, Q., 2018. Generating lane-based intersection maps from crowdsourcing big trace data. Transp. Res. Part C Emerg. Technol. 89, 168–187. URL: https://doi.org/10.1016/j.trc.2018.02.007, doi:10.1016/j.trc.2018.02.007.
Yin, Y., Shah, R.R., Wang, G., Zimmermann, R., 2018. Feature-based Map Matching for Low-Sampling-Rate GPS Trajectories. ACM Trans. Spat. Algorithms Syst. 4, 1–24. doi:10.1145/3266430.
Yuan, J., Zheng, Y., Zhang, C., Xie, X., Sun, G.Z., 2010. An Interactive-Voting based Map Matching algorithm. Proc. - IEEE Int. Conf. Mob. Data Manag. , 43–52doi:10.1109/MDM.2010.14.
Zheng, K., Zheng, Y., Xie, X., Zhou, X., 2012. Reducing uncertainty of low-sampling-rate trajectories. Proc. - Int. Conf. Data Eng. , 1144–1155doi:10.1109/ICDE.2012.42.
Zhu, L., Holden, J.R., Gonder, J.D., 2017. Trajectory segmentation map-matching approach for large-scale, high-resolution GPS data. Transp. Res. Rec. 2645, 67–75.