Signature-Based Trajectory Similarity Join

Na Ta Guoliang Li Yongqing Xie Changqi Li Shuang Hao Jianhua Feng

Abstract—Emerging vehicular trajectory data have opened up opportunities to benefit many real-world applications, e.g., frequent trajectory based navigation systems, road planning, car pooling, etc. The similarity join is a key operation to enable such applications, which finds *similar* trajectory pairs from two large collections of trajectories. Existing similarity metrics on trajectories rely on aligning sampling points of two trajectories. However due to different sampling rates or different vehicular speeds, the sample points in similar trajectories may not be aligned. To address this problem, we propose a new bi-directional mapping similarity (`BDS`), which allows a sample point of a trajectory to align to the closest location (which may not be a sample point) on the other trajectory, and vice versa. Since it is expensive to enumerate every two trajectories and compute their similarity, we propose `Strain-Join`, a signature-based trajectory similarity join framework. `Strain-Join` first generates signatures for each trajectory such that if two trajectories do not share common signatures, they cannot be similar. In order to utilize this property to prune dissimilar pairs, we devise several techniques to generate high-quality signatures and propose an efficient filtering algorithm to prune dissimilar pairs. For the pairs not pruned by the filtering algorithm, we propose effective verification algorithms to verify whether they are similar. Experimental results on real datasets show that our algorithm outperforms state-of-the-art techniques in terms of both effectiveness and efficiency.

Index Terms—Trajectory, Similarity Join, Signature-Based Method, Trajectory Similarity, Filtering-Verification Framework

1 Introduction

The advancement of vehicle positioning techniques have produced high volume of trajectory data, where a vehicular trajectory is a sequence of discrete sample points (geo-locations) at sampling time of a running vehicle. For example, a taxi or a Uber car will generate a trajectory from the pick-up point to a drop-off point, which contains a set of discrete sample points on the driving route at every sampling time (e.g., every second). Trajectory data can benefit many real-world applications, such as frequent trajectory based navigation[22], taxi pick-up recommending system[27], trajectory interpreter[31], traffic condition analysis[3] and adaptive trajectory storage system[8].

Trajectory similarity join, which, given two large collections of trajectories, finds all similar trajectory pairs from the two collections, is an important operation in many applications. For example, in navigation systems, we can mine the frequent routes from taxi trajectories and utilize them to recommend better routes for ordinary users. In government road planning, we can dig out the 'heavy routes' that are covered by many trajectories and thus the government can plan to construct new roads to ease the burden of the heavy routes. Both applications require to compute the similarity of trajectories and identify the most similar trajectories.

Existing trajectory similarity functions rely on aligning the sample points between two trajectories, e.g., closest-pair distance[25], edit distance on real sequence[6], edit distance with real penalty[5], and one way distance[20]. However due to different sampling rates or different vehicular speeds, the sample points may not be well aligned. For example, in Figure 1, trajectories T_i and T_j are similar. However, if we align them based on closet sample point pairs as illustrated in Figure 1(a), their similarity is rather small, because they have different speeds in different time periods.

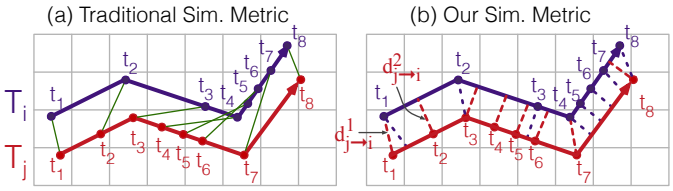


Fig. 1. Trajectory Similarity Functions.

To address this problem, we propose a new bi-directional mapping similarity (`BDS`), which allows a sample point of a trajectory to align to the closet location(which may not be a sample point) on the other trajectory, and vice versa. For example, in Figure 1(b), we align points of T_j (T_i) to their closest position on T_i (T_j) and get a larger similarity, resulting in a more practical similarity measurement.

It is expensive to enumerate every two trajectories and compute their similarity. To address this problem, we propose `Strain-Join`, a signature-based trajectory similarity join framework. `Strain-Join` includes two steps: the filtering step and the verification step. In the filter step, `Strain-Join` first generates signatures for each trajectory such that if two trajectories do not share common signatures, they cannot be similar, where a signature captures positional characteristics to uniquely identify a trajectory. We can utilize this property to prune large numbers of dissimilar pairs. To this end, we devise several filtering techniques to generate high-quality signatures and propose an efficient filtering algorithm to prune dissimilar pairs. In the verification step, `Strain-Join` verifies whether the pairs that are not pruned by the filtering algorithm are similar. It is expensive to directly verify each pair, and we propose to utilize the signatures to efficiently verify the pairs. We showcase how `Strain-Join` works using trajectories in Figure 2, where we simply take the grids having overlap with a trajectory as its signature. In the filtering step, T_1 in Figure 2(a) and T_2 in Figure 2(b) do not share common signatures, therefore they

- *Na Ta, Guoliang Li, Yongqing Xie, Shuang Hao and Jianhua Feng are with the Department of Computing Science, TNList, Tsinghua University, Beijing, China. {dan13,xyq14,haos13}@mails.tsinghua.edu.cn, {liguoliang, fengjh}@tsinghua.edu.cn. Guoliang Li is the corresponding author.*
- *Changqi Li is with the High School affiliated to Renmin University, Beijing, China. E-mail: lcq_1999@126.com.*

are not similar. T_3 in Figure 2(c) and T_4 in Figure 2(d) both have common signatures with T_1, so $\langle T_1, T_3\rangle$ and $\langle T_1, T_4\rangle$ are two candidate similar pairs. In the verification step, we compute the BDS similarity values and eliminate the false positives, and the final answer is $\{\langle T_1, T_3\rangle\}$.

To summarize, we make the following contributions.

(1) We propose a new similarity function and an efficient signature-based framework to solve the trajectory similarity join problem.

(2) We develop effective techniques to generate signatures. We propose grid-based signatures that utilize the grids to generate the signatures, and threshold-aware signatures that uses the threshold to generate signatures. We devise a cost-based model to select high-quality signatures. We propose an efficient filtering algorithm that utilizes the signatures to prune large numbers of dissimilar pairs.

(3) We devise efficient verification techniques. We propose a signature-based method to reduce the complexity and an expansion-based method to prune unnecessary signatures.

(4) We have conducted an extensive set of experiments on real-world datasets. Experimental results show that our algorithm achieves high quality and efficiency, and outperforms state-of-the-art approaches.

The rest of the paper is organized as follows. We formalize our problem in Section 2. Section 3 introduces the framework. We propose filtering algorithms in Section 4 and develop verification techniques in Section 5. Experimental results are provided in Section 6. We review related work in Section 7 and conclude the paper in Section 8.

2 Problem Formulation

Given two collections of trajectories, a similarity join aims to find all similar trajectory pairs from the two collections. Before we formulate the trajectory join problem, we first give a formal definition of a trajectory.

Definition 1 (Trajectory)**.** *A trajectory T is a sequence of sample points, i.e., $T = \{p^1, p^2, \cdots, p^{|T|}\}$, where p^k is a sample point (i.e., a geo-location) and $|T|$ is the number of sample points in T.*

Two adjacent sample points on T form a line and any location on such lines belongs to T. Thus T has $|T| - 1$ lines and infinite numbers of locations.

Given two trajectories T_i and T_j, a sample point p_i^k on T_i may be aligned to a location on a line of T_j, and vice versa. We want to align p_i^k to the closet location on T_j. Denote by $\texttt{Dist}_{\texttt{PT}}(p_i^k, T_j)$ the minimal distance from a sample point p_i^k of trajectory T_i to a trajectory T_j, i.e.,

$$\texttt{Dist}_{\texttt{PT}}(p_i^k, T_j) = \min_{l \in T_j} \texttt{Dist}_{\texttt{PL}}(p_i^k, l), \tag{1}$$

where l is any line on T_j and $\texttt{Dist}_{\texttt{PL}}(p_i^k, l)$ is the minimal distance from point p_i^k to line l. If the perpendicular line from p_i^k to l has an intersection point p^* with l, then $\texttt{Dist}_{\texttt{PL}}(p_i^k, l)$ is the distance between p_i^k and p^*; otherwise, $\texttt{Dist}_{\texttt{PL}}(p_i^k, l)$ is the shorter distance between p_i^k and the two end points of l.

Basically, the closer two trajectories are, the shorter distances (e.g., $\texttt{Dist}_{\texttt{PT}}(p_i^k, T_j)$) between each sample point of one trajectory (e.g., p_i^k) and the other trajectory (e.g., T_j) are. If $\texttt{Dist}_{\texttt{PT}}(p_i^k, T_j)$ is large, p_i^k will not be aligned to any location at T_j. To address this issue, we set a maximal distance bound $\texttt{D}_{\texttt{max}}$, and use it to compute a normalized distance $d_{i\to j}^k$, where

$$d_{i\to j}^k = \begin{cases} \frac{\texttt{Dist}_{\texttt{PT}}(p_i^k, T_j)}{\texttt{D}_{\texttt{max}}} & \text{if } \texttt{Dist}_{\texttt{PT}}(p_i^k, T_j) \le \texttt{D}_{\texttt{max}} \\ +\infty & \text{if } \texttt{Dist}_{\texttt{PT}}(p_i^k, T_j) > \texttt{D}_{\texttt{max}} \end{cases} \tag{2}$$

Accordingly, the smaller $d_{i\to j}^k$ ($d_{j\to i}^k$) is, the larger similarity T_i and T_j have. "Typically, the GPS nominal accuracy is about $15m$"[29]. If the distance of two points is larger than the nominal accuracy, the two points cannot be aligned. Thus we can set D_{max} based on the GPS sampling error.

Next we propose the bi-directional similarity.

Definition 2 (Bi-Directional Similarity)**.** *The Bi-Directional (*BD*) Similarity of trajectories T_i and T_j is*

$$\textsc{Sim}(T_i, T_j) = 1 - \frac{\sum_{k=1}^{|T_i|} d_{i\to j}^k + \sum_{k=1}^{|T_j|} d_{j\to i}^k}{|T_i| + |T_j|} \tag{3}$$

Algorithm To Compute BD-Similarity of Two Trajectories. Given two trajectories T_i and T_j, we first compute $d_{i\to j}^k$ for each sample point p_i^k in T_i by first computing the minimal distance from p_i^k to each line l in T_j and then selecting the minimal one. The complexity of computing the minimal distance from each sample point in T_i to a line in T_j is $\mathcal{O}(|T_j|)$ and the complexity of computing the minimal distance from all sample points in T_i to T_j is $\mathcal{O}(|T_i||T_j|)$. Next we compute $d_{j\to i}^k$ for each sample point in T_j to T_i. Thus the overall complexity of the algorithm is $\mathcal{O}(|T_i||T_j|)$.

For example, in Figure 1(b), $|T_i| = |T_j| = 8$. For point p_j^1, $d_{j\to i}^1$ is the minimal normalized distance from p_j^1 to its closest position on line $p_i^1 \to p_i^2$ of T_i, i.e., the first sample point p_i^1 of T_i. While for point p_j^2, $d_{j\to i}^2$ is computed using the minimal distance from p_j^2 to line $p_i^1 \to p_i^2$, which is on the perpendicular line from p_j^2 to $p_i^1 \to p_i^2$. For trajectory pair T_i and T_j, $\textsc{Sim}(T_i, T_j) = 1 - (\sum_{k=1}^{8} d_{i\to j}^k + \sum_{k=1}^{8} d_{j\to i}^k)/(8+8)$.

If the similarity of T_i and T_j is not less than a specified threshold τ, they are similar. Next we formalize the problem of trajectory similarity joins.

Definition 3 (Trajectory Similarity Joins)**.** *Given two sets of trajectories $\mathcal{T}$ and $\mathcal{S}$, a similarity threshold τ, find all similar trajectory pairs $\langle T \in \mathcal{T}, S \in \mathcal{S}\rangle$ such that* $\textsc{Sim}(T, S) \ge \tau$.

Without loss of generality, we first focus on self join, i.e., $\mathcal{T} = \mathcal{S}$. For example, given $\mathcal{T} = \{T_1, T_2, T_3, T_4, T_5\}$ in Figure 2, self similarity join returns $\langle T_1, T_3\rangle$ as the result. We discuss how to join two different sets ($\mathcal{T} \neq \mathcal{S}$) in Section 3.

Remark. Although each sample point in a trajectory has a sampling time, we focus on computing the similarity trajectories with similar shape (as illustrated in Figure 1) and do not consider the time information. Note that in many applications, due to different traffic condiations, the sampling points of similar trajectories may not be well aligned, but they have similar shape. In this paper, we focus on finding similar trajectories with similar shape.

Table 1 lists the notations we use in this paper.

3 Signature-Based Similarity Join

We propose a signature-based similarity join framework, which first generates the signatures for each trajectory, then utilizes the signature to prune dissimilar trajectory pairs that do not share any common signature, and finally verifies the pairs that are not pruned. We first present two methods

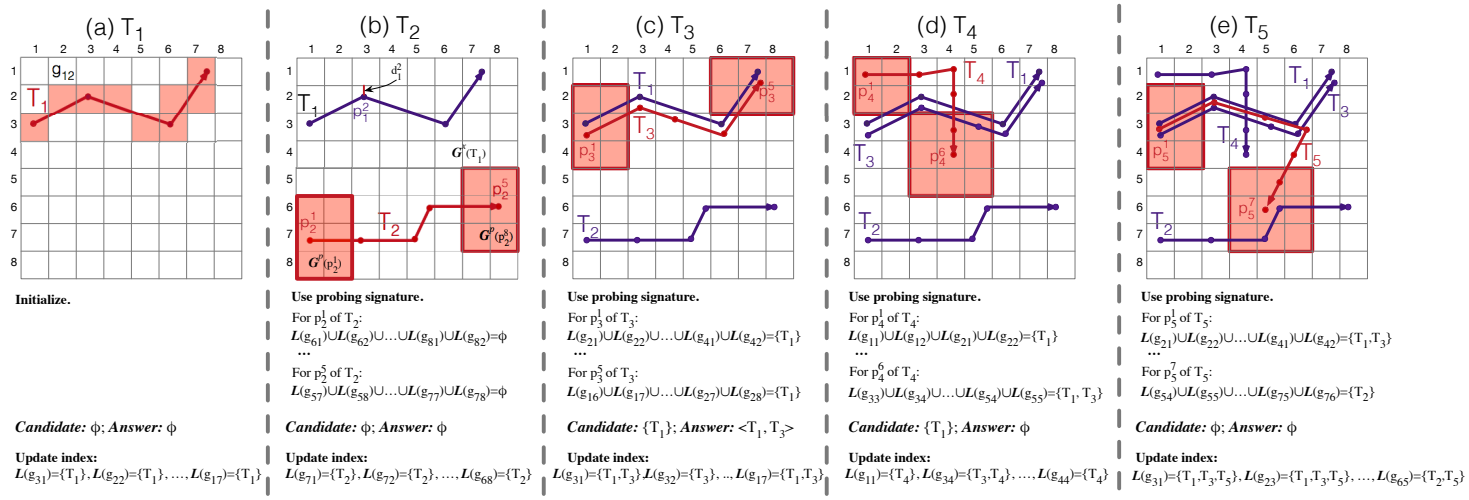


Fig. 2. An Example of the `Strain-Join` Framework.

TABLE 1
Notations

Notation	Description
T_i	A raw trajectory, $T_i = \{p_i^1, p_i^2, \cdots, p_i^{\lvert T_i \rvert}\}$
LEN (T_i)	The length of T_i
$\texttt{Dist}_{\texttt{PT}}(p_i^k, T_j)$	The minimal distance from p_i^k of T_i to T_j
$d_{i\to j}^k$	The normalized distance of $\texttt{Dist}_{\texttt{PT}}(p_i^k, T_j)$
$\texttt{D}_{\max}$	The maximal trajectory distance bound
$\textsc{Sim}(T_i, T_j)$	The Bi-Directional similarity of T_i and T_j
τ	The similarity threshold
$\mathcal{T}, \mathcal{S}$	The trajectory sets
$\mathcal{G}^p(p_j^k)$	$\{g \mid \textsc{MinDist}(p_j^k, g) \le \texttt{D}_{\max}\}$
$\mathcal{G}^x(T_i)$	$\{g \mid g \cap T_i \neq \phi\}$
$\mathcal{L}(g)$	Trajectories that take g as an indexing signature
d_j^k	$\min_l \texttt{Dist}_{\texttt{PL}}(p_j^k, l)/\texttt{D}_{\max}$
$\mathcal{G}^\tau(T_j)$	$\{g(p_j^k) \mid k \le x+1\}$
$\mathcal{C}_j^k$	The candidate set for point p_j^k
COR (p_j^k, p_j^{k+1})	Sample point correlation for p_j^k and p_j^{k+1}
$\hat{d}_{i\to j}^k[g_x]$	The minimal distance from p_i^k to lines of T_j in g_x
$\texttt{lb}_{i\to j}^k[g_x]$	The lower bound of $d_{i\to j}^k$
$\texttt{ub}_{i\to j}^k[g_x]$	The upper bound of $d_{i\to j}^k$

to generate the signatures (Sections 3.1 and 3.2) and then propose the similarity join framework (Section 3.3).

3.1 Grid-Based Signatures

Consider two trajectories T_i and T_j. For any sample point p_j^k in T_j, if the minimal distance from p_j^k to T_i is larger than $\texttt{D}_{\max}$, then $d_{j\to i}^k$ is rather large (Equation 2), and thus T_i and T_j cannot be similar (Equation 3). However it is rather expensive to compute the minimal distance $d_{j\to i}^k$ for every two trajectories. To address this issue, we propose a grid-based signature to check whether $d_{i\to j}^k$ is larger than $\texttt{D}_{\max}$.

Grid Index Structure. We build a grid index $\mathcal{G}$, where grid cell width is w (we will discuss how to set w in Section 3.3). In the sequel, we use 'grid' and 'grid cell' (denoted by g) interchangeably for ease of notation. Other spatial indices such as R-Tree are less applicable because: (1) the maintenance of an R-Tree is costly if the index structure needs frequent updates, (2) the leaf nodes of an R-Tree may overlap with each other, and (3) equal-sized grids can facilitate the computation of distances.

Grids Whose Minimal Distances to A Sample Point Are Not Larger Than $\texttt{D}_{\max}$. For each sample point p_j^k in T_j, we compute the set of grids, $\mathcal{G}^p(p_j^k)$, whose minimal distances to p_j^k are not larger than $\texttt{D}_{\max}$, i.e.,

$$\mathcal{G}^p(p_j^k) = \{g \mid \textsc{MinDist}(p_j^k, g) \le \texttt{D}_{\max}\}, \tag{4}$$

$$\textsc{MinDist}(p_j^k, g) = \begin{cases} 0 & \text{if } p_j^k \in g \\ \min_l \texttt{Dist}_{\texttt{PL}}(p_j^k, l) & \text{otherwise} \end{cases} \tag{5}$$

where l is any side of grid cell g, and $p_j^k \in g$ means that point p_j^k is located within g.

Without loss of generality, we set $w = \texttt{D}_{\max}$ in our examples throughout the paper. For point p_j^k in Figure 3(a), the red line from p_j^k to the lower-left vertex of grid g_{24} denotes $\textsc{MinDist}(p_j^k, g_{24})$; the nine shaded grids around p_j^k are within $\texttt{D}_{\max}$ distance to p_j^k and they compose $\mathcal{G}^p(p_j^k)$, i.e., $\mathcal{G}^p(p_j^k) = \{g_{22}, g_{23}, g_{24}, g_{32}, g_{33}, g_{34}, g_{42}, g_{43}, g_{44}\}$.

Grids Covering A Location on T_i. If any location at trajectory T_i does not fall in any grid in $\mathcal{G}^p(p_j^k)$, the minimal distance from p_j^k to T_i is larger than $\texttt{D}_{\max}$, therefore T_i cannot be similar to T_j. Next we discuss how to check whether T_i has a location falling in a grid in $\mathcal{G}^p(p_j^k)$. For T_i, we compute the set of grids that cover any location of T_i to assist the similarity check, i.e.,

$$\mathcal{G}^x(T_i) = \{g \mid g \cap T_i \neq \phi\}. \tag{6}$$

For example, in Figure 3(b), all the shaded grids along T_i, namely, $g_{31}, g_{22}, g_{23}, g_{24}, g_{35}, g_{36}, g_{26}, g_{27}, g_{17}$, compose $\mathcal{G}^x(T_i)$ for the given trajectory T_i.

Pruning Strategy. If $\mathcal{G}^p(p_j^k) \cap \mathcal{G}^x(T_i) = \phi$, the minimal distance from any point p_j^k to T_i is larger than $\texttt{D}_{\max}$, accordingly T_i and T_j cannot be similar as stated in Lemma 1.

Lemma 1. *Given two trajectories T_i and T_j, if $\mathcal{G}^p(p_j^k) \cap \mathcal{G}^x(T_i) = \phi$, then T_i and T_j cannot be similar.*

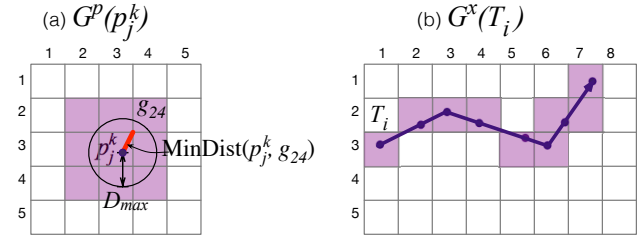


Fig. 3. Illstration of signatures ($w = \texttt{D}_{\max}$)

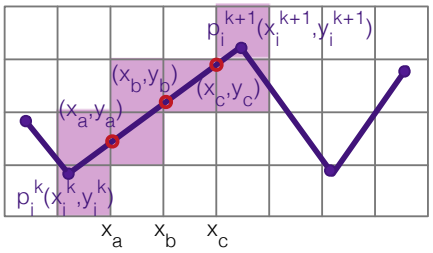


Fig. 4. Computing $\mathcal{G}^x(T_i)$.

For example, in Figure 2(a), all the shaded grids along T_1 compose $\mathcal{G}^x(T_1)$. For trajectory T_2 in Figure 2(b), none of $\mathcal{G}^p(p_2^k)(1 \leq k \leq |T_2|)$ overlaps $\mathcal{G}^x(T_1)$, such as $\mathcal{G}^p(p_2^1)$ and $\mathcal{G}^p(p_2^5)$ denoted by the shaded grids with bold-borders. Therefore, T_2 and T_1 are not similar.

Computing $\mathcal{G}^p(p_j^k)$. We first locate the grid where p_j^k resides in $\mathcal{O}(1)$ time. Then we check each of its eight surrounding neighbor grids. For each such grid g, if $\text{MINDIST}(p_j^k, g) \leq \texttt{D}_{\texttt{max}}$, we add it into $\mathcal{G}^p(p_j^k)$ and visit its neighbor grids; otherwise we skip the grid. Iteratively, we can compute $\mathcal{G}^p(p_j^k)$. The complexity is $\mathcal{O}(|\mathcal{G}^p(p_j^k)|)$. As the minimal distance from p_j^k to g is not larger than $\texttt{D}_{\texttt{max}}$, there are at most $2(\lceil \frac{\texttt{D}_{\texttt{max}}}{w} \rceil + 1)^2$ grids. Thus the complexity is $\mathcal{O}(\lceil \frac{\texttt{D}_{\texttt{max}}}{w} \rceil^2)$.

In Figure 3(a), sample point p_j^k is located in grid g_{33}, so initially $\mathcal{G}^p(p_j^k) = \{g_{33}\}$; then we check the eight neighbors of g_{33}: g_{22}, g_{23}, g_{24}, g_{32}, g_{34}, g_{42}, g_{43} and g_{44}; and they are added to $\mathcal{G}^p(p_j^k)$ as their MINDIST's to p_j^k are within $\texttt{D}_{\texttt{max}}$. Next, for each of these eight newly added grids, we check their neighbors, e.g., g_{22}'s neighbors g_{21}, g_{11} and g_{22}. This time, all the newly checked neighbor grids are more than $\texttt{D}_{\texttt{max}}$ away from p_j^k, so the final $\mathcal{G}^p(p_j^k)$ is the nine shaded grids around p_j^k in Figure 3(a).

Computing $\mathcal{G}^x(T_i)$. For each sample point p_i^k, we compute the grid that p_i^k is located in $\mathcal{O}(1)$ time, denoted by $g(p_i^k)$. Then for p_i^k and p_i^{k+1}, we find the grids between grids $g(p_i^k)$ and $g(p_i^{k+1})$. As illustrated in Figure 4, we find all the intersections of line $p_i^k \rightarrow p_i^{k+1}$ with the vertical lines of the grid index between the x-coordinates of p_i^k and p_i^{k+1} and get $(x_a, y_a), (x_b, y_b), (x_c, y_c)$, then the grids covered by coordinates ranges $[(x_i^k, y_i^k), (x_a, y_a)]$, $[(x_a, y_a), (x_b, y_b)]$, $[(x_b, y_b), (x_c, y_c)]$ and $[(x_c, y_c), (x_i^{k+1}, y_i^{k+1})]$ are the grids between grids $g(p_i^k)$ and $g(p_i^{k+1})$. The complexity is $\mathcal{O}(|\mathcal{G}^p(T_i)|)$. As the number of grids covering T_i is at most $\lceil \frac{\text{LEN}(T_i)}{w} \rceil + 1$, where $\text{LEN}(T_i)$ is the length of T_i, the complexity is $\mathcal{O}(\lceil \frac{\text{LEN}(T_i)}{w} \rceil)$.

SIGNATURE-BASED FRAMEWORK. We first define two signatures of a trajectory.

Definition 4 (Probing Signatures). *Given a trajectory T_j, for each sample point p_j^k of T_j, girds in $\mathcal{G}^p(p_j^k)$ are the probing signatures of p_j^k.*

Definition 5 (Indexing Signatures). *Given a trajectory T_i, grids in $\mathcal{G}^x(p_i^k)$ for each sample point p_i^k of T_i are the indexing signatures of T_i.*

Indexing. We maintain two types of indices to organize trajectories and signatures respectively. The first is the trajectory index, which keeps a set of grids for each trajectory that have overlap with the trajectory. It is utilized to get the grids for each trajectory. The second is the signature index, which keeps a set of trajectories for each indexing signature that contain the indexing signature. It is used to find the candidates. For each trajectory T_i, we find its similar candidate as follows. For each sample point p_i^k, we compute its probing signature $\mathcal{G}^p(p_i^k)$ and find the trajectories whose indexing signatures overlap with $\mathcal{G}^p(p_i^k)$. We utilize the signature index to store the indexing signatures, where the entries are indexing signatures and each indexing signature maintains an inverted list of trajectories that contain the indexing signature. We use $\mathcal{L}(g)$ to denote the set of trajectories taking grid g as an indexing signature.

Filtering. Suppose we construct the inverted index for all the indexing signatures of all trajectories. Then, given a trajectory T_j, considering a sample point p_j^k, we compute its probing signatures. For each grid g in the probing signature, we access the inverted list $\mathcal{L}(g)$. Trajectories in $\mathcal{L}(g)$ may be similar to T_j and we take them as candidates. Thus the candidate set is $\cup_{g \in \mathcal{G}^p(p_j^k)} \mathcal{L}(g)$. For each candidate T_i, we compute the similarity between T_i and T_j. If their similarity is not smaller than τ, we take $\langle T_i, T_j \rangle$ as an answer.

Incremental Indexing and Filtering. The above method may generate a candidate pair twice. For example, suppose $\langle T_i, T_j \rangle$ is an answer. When we use T_i to find the candidate, we identify this candidate pair. We will find it again when using T_j. To address this issue, we incrementally build the index. We visit the trajectories in ascending order of sample point numbers. When computing the answer of T_j, we only consider the trajectories before T_j (i.e., with less or equal numbers of sample points) and build the index for them. After processing T_j, we generate its indexing signatures and insert them into the index.

The framework functions as follows in Figure 2: we first access trajectory T_1, compute T_1's indexing signature and build the inverted lists: $\mathcal{L}(g_{31}) = \{T_1\}$, $\mathcal{L}(g_{22}) = \{T_1\}$, ... Then we move on to the second trajectory T_2. For p_2^1, $\mathcal{G}^p(p_2^1) = \{g_{61}, g_{62}, g_{71}, g_{72}, g_{81}, g_{82}\}$, we have $\mathcal{L}(g_{61}) = \phi$, $\mathcal{L}(g_{62}) = \phi, \ldots, \mathcal{L}(g_{82}) = \phi$, which means that the candidate set for point p_2^1 is empty; the rest of T_2's sample points also have empty candidate sets, thus there is no candidate for T_2 yet. We insert T_2 into inverted lists of corresponding grids: $\mathcal{L}(g_{71}) = \{T_2\}$, $\mathcal{L}(g_{72}) = \{T_2\}$, ... For the third trajectory T_3, we get a candidate $\{T_1\}$. As $\text{SIM}(T_1, T_3) > \tau$, $\langle T_1, T_3 \rangle$ is an answer. The index is again updated and the rest trajectories are processed in the same manner.

3.2 Threshold-Aware Signatures

It can be seen that a large threshold τ corresponds to a smaller number of answers. However the above method does not utilize τ to prune dissimilar pairs. To address this issue, we introduce a *threshold-aware signature* scheme, which fully utilizes the threshold to prune dissimilar pairs.

Given a trajectory T_j, for each sample point p_j^k and its corresponding grid $g(p_j^k)$, we compute the minimal distance, denoted by d_j^k, from p_j^k to a location outside $g(p_j^k)$,

$$d_j^k = \frac{\min_l \texttt{Dist}_{\texttt{PL}}(p_j^k, l)}{\texttt{D}_{\texttt{max}}}, \tag{7}$$

where l is a side of $g(p_j^k)$. Thus for any location outside $g(p_j^k)$, its distance to p_j^k must be larger than d_j^k.

Take Figure 2(b) as an example. The minimal distance d_1^2 from p_1^2 in grid $g(p_1^2)$, i.e., g_{23}, to any location outside g_{23}, is

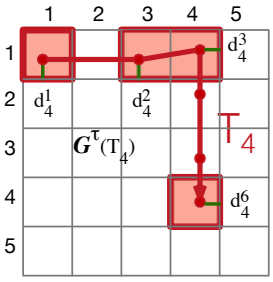


Fig. 5. Threshold-Aware Signature.

denoted by the perpendicular line from p_1^2 to the upper side of g_{23}; for any location outside g_{23}, its minimal distance to p_1^2 should be larger than d_1^2.

Given another trajectory T_i where $|T_i| \leq |T_j|$, based on Equation 3, if $\text{SIM}(T_i, T_j) = 1 - \frac{\sum_{k=1}^{|T_i|} d_{i\to j}^k + \sum_{k=1}^{|T_j|} d_{j\to i}^k}{|T_i|+|T_j|} \geq \tau$, then $\sum_{k=1}^{|T_i|} d_{i\to j}^k + \sum_{k=1}^{|T_j|} d_{j\to i}^k \leq (1-\tau)(|T_i| + |T_j|)$, and

$$\sum_{k=1}^{|T_j|} d_{j\to i}^k \leq \tau_j = 2(1-\tau)|T_j|. \tag{8}$$

If T_i does not have a location falling in $g(p_j^k)$, we have $d_{j\to i}^k \geq d_j^k$. Thus if $\sum_{k=1}^{|T_j|} d_j^k > \tau_j$, the similarity of T_i and T_j is smaller than τ and they cannot be similar as proved by Lemma 2.

Lemma 2. *If $\sum_{k=1}^{|T_j|} d_j^k > \tau_j$ for T_j, for any T_i having no location in any of $g(p_j^k)$, T_i and T_j cannot be similar.*

Using this feature, we generate a threshold-aware signature as follows.

Threshold-Aware Signature. For each trajectory T_j, if $\sum_{k=1}^{|T_j|} d_j^k > \tau_j$, we generate its threshold-aware probing signatures $\mathcal{G}^\tau(T_j)$ as follows. We sort $d_j^1, d_j^2, \cdots, d_j^{|T_j|}$ in descending order. Without loss of generality, suppose $d_j^1 \geq d_j^2 \geq \cdots \geq d_j^{|T_j|}$. We select the grids of the first $x+1$ sample points as the threshold-aware signatures, where $\sum_{k=1}^{x} d_j^k \leq \tau_j$ and $\sum_{k=1}^{x+1} d_j^k > \tau_j$. Thus

$$\mathcal{G}^\tau(T_j) = \{g(p_j^k) | k \leq x+1\} \tag{9}$$

where $\sum_{k=1}^{x} d_j^k \leq \tau_j$ and $\sum_{k=1}^{x+1} d_j^k > \tau_j$.

We can prove that given two trajectories T_i and T_j, if $\mathcal{G}^x(T_i) \cap \mathcal{G}^\tau(T_j) = \phi$, T_i and T_j cannot be similar. This is because T_i and T_j have no common signature, thus $d_{j\to i}^k \geq d_j^k$ for each k and $\sum_{k=1}^{|T_j|} d_{j\to i}^k > \tau_j$ as stated in Lemma 3.

Lemma 3. *Given two trajectories T_i and T_j, if $\mathcal{G}^x(T_i) \cap \mathcal{G}^\tau(T_j) = \phi$, T_i and T_j cannot be similar.*

In Figure 5, for trajectory T_4, we have $d_4^1 + d_4^2 + d_4^3 + d_4^6 > \tau_4$, thus T_4 has a threshold-aware probing signature, $\mathcal{G}^\tau(T_4) = \{g_{11}, g_{13}, g_{14}, g_{44}\}$, denoted by the four bordered grids. Since $\mathcal{G}^x(T_1) \cap \mathcal{G}^\tau(T_4) = \phi$, T_1 and T_4 are not similar. Similarly, $\mathcal{G}^x(T_2) \cap \mathcal{G}^\tau(T_4) = \phi$ and $\mathcal{G}^x(T_3) \cap \mathcal{G}^\tau(T_4) = \phi$, thus there is no similar trajectory for T_4 yet.

3.3 The Signature-Based Algorithm

Based on the signatures, we propose a filter-and-verification framework.

(1) **Sorting**: We sort the trajectories by the number of sample points in ascending order and create the trajectory index.

(2) **Filtering**: We access the trajectories in order. For each trajectory T_j, if $\sum_{k=1}^{|T_j|} d_j^k > \tau_j$, we generate its threshold-aware signature $\mathcal{G}^\tau(T_j)$. For each grid $g \in \mathcal{G}^\tau(T_j)$, we retrieve its inverted list $\mathcal{L}(g)$ and the trajectories on $\mathcal{L}(g)$ are candidates of T_j. Thus the candidate set based on threshold-aware signature is $\mathcal{C}_j^\tau = \cup_{g\in\mathcal{G}^\tau(T_j)}\mathcal{L}(g)$. Then we generate the probing signatures $\mathcal{G}^p(p_j^k)$ for each sample point p_j^k in T_j, and $\mathcal{C}_j^k = \cup_{g\in\mathcal{G}^p(p_j^k)}\mathcal{L}(g)$ is a candidate set ($1 \leq k \leq |T_j|$). Thus here we have $|T_j|+1$ candidate sets. We discuss how to effectively use them to generate candidates in Section 4. Here we compute their intersection as the candidate set, i.e.,

$$\mathcal{C} = \mathcal{C}_j^\tau \cap \bigcap_{1\leq k\leq |T_j|} \mathcal{C}_j^k. \tag{10}$$

(3) **Indexing**: We generate the indexing signature of T_j, $\mathcal{G}^x(T_j)$. For each indexing signature g of T_j, we insert $\langle g, T_j\rangle$ into the index, i.e., appending T_j to the inverted list of $\mathcal{L}(g)$.

(4) **Verification**: For each candidate T_i of T_j, we compute their real similarity. If the similarity is not smaller than τ, we return $\langle T_i, T_j\rangle$ as an answer.

Algorithm 1 shows the pseudo code of our framework and Figure 2 provides a running example. First we sort trajectories to make sure T_i is always processed before T_j if $|T_i| < |T_j|$. For trajectory T_1, inverted lists of grids in T_1's indexing signature are created and initialized to $\{T_1\}$, e.g., $\mathcal{L}(g_{31}) = \{T_1\}$. For T_2, it does not have threshold-aware signature, so we retrieve the inverted lists of grids contained by the probing signature of T_2's sample points, e.g., $\mathcal{L}(g_{61})$, $\mathcal{L}(g_{62})$, etc., and use their union set as T_2's candidate set, which is empty, meaning there is no candidate for T_2. Then we update the index to contain T_2, i.e., add T_2 to inverted lists of all grids in $\mathcal{G}^x$ (T_2). Trajectory T_3 has no threshold-aware signature either. We retrieve inverted lists of T_3's probing signature, e.g., $\mathcal{L}(g_{31})$, $\mathcal{L}(g_{32})$, etc., whose union is $\{T_1\}$, making T_1 a candidate for T_3. Since $\text{SIM}(T_1, T_3) > \tau$, $\langle T_1, T_3\rangle$ is an answer. Using probing signature, there is one candidate (T_1) for trajectory T_4, although $\langle T_1, T_4\rangle$ is not a qualified answer after verification. In fact, T_4 has a threshold-aware signature which can directly exclude T_1 as candidate, as alternatively suggested by Figure 5. Thus, we use the threshold-aware signature to avoid unnecessary verification. The rest trajectories are processed in the same manner util all trajectories are processed.

Complexity. We first analyze the space complexity. The maximum number of grids a trajectory can cover is $\mathcal{O}(\frac{\text{LEN}(T_i)}{w})$, where $\text{LEN}(T_i)$ is the length of T_i. This is exactly the number of indexing signature of T_i. As each indexing signature is stored once, the overall space complexity is $\mathcal{O}(\sum_{T_i\in\mathcal{T}} \frac{\text{LEN}(T_i)}{w})$.

Given a trajectory T_j, the cost of using the threshold-aware signature to find the candidate is $\mathcal{O}(\sum_{g\in\mathcal{G}^\tau(T_j)} |\mathcal{L}(g)|)$, and the cost of using the probing signature to find the candidate is $\mathcal{O}(\sum_{g\in\mathcal{G}^p(p_j^k)} |\mathcal{L}(g)|)$. The overall cost of filtering is $\mathcal{O}(\sum_{g\in\mathcal{G}^\tau(T_j)} |\mathcal{L}(g)| + \sum_{1\leq k\leq |T_j|} \sum_{g\in\mathcal{G}^p(p_j^k)} |\mathcal{L}(g)|)$.

Suppose there are $|\mathcal{C}|$ candidates. The verification cost is $\mathcal{O}(|\mathcal{C}||T_{max}|^2)$, where $|T_{max}|$ is the maximal number of sample points in a trajectory.

Discussion on Determining w. The larger w is, the larger the number of trajectories in a grid is, the larger the number of candidate is, and thus the larger the verification cost is.

Algorithm 1: Strain-Join $(\mathcal{T}, \tau, \mathtt{D_{max}})$

Input: $\mathcal{T}$: a trajectory set
τ: similarity threshold
$\mathtt{D_{max}}$: maximal trajectory distance
Output: $\mathcal{A} = \{\langle T_i \in \mathcal{T}, T_j \in \mathcal{T}\rangle | \text{SIM}(T_i, T_j) \geq \tau\}$

```
1  begin
2      Sort T by |T_i| in ascending order;
3      Construct grids with width w;
4      for T_j ∈ T do
5          C = CandidateSetGen(T_j, G, τ, D_max);
6          for T_i ∈ C do
7              Verification(T_i, T_j, τ);
8          UpdateIndex(T_j);
9      return A;
10 end
```

Function CandidateSetGen $(T_j, \mathcal{G}, \tau, \mathtt{D_{max}})$

Input: T_j: a trajectory; $\mathcal{G}$: the grid index;
τ: similarity threshold;
$\mathtt{D_{max}}$: maximal trajectory distance
Output: $\mathcal{C}$: candidate set for T_j

```
1  begin
2      if Σ_{k=1}^{|T_j|} d_j^k > τ_j then
3          Generate threshold-aware signature set G^τ(T_j);
4          Generate candidate set C_j^τ = ∪_{g∈G^τ(T_j)} L(g);
5      for p_j^k ∈ T_j do
6          Generate probing signature set G^p(p_j^k);
7          Generate candiate set C_j^k = ∪_{g∈G^p(p_j^k)} L(g);
8      C = C_j^τ ∩ ⋂_{1≤k≤|T_j|} C_j^k;
9      return C
10 end
```

Function UpdateIndex (T_j)

Input: T_j: a trajectory
Output: $\mathcal{L}$: Inverted Index

```
1  begin
2      Generate indexing signature set G^x(T_j);
3      for g ∈ G^x(T_j) do L(g) ← T_j;
4  end
```

Function Verification $(T_i, T_j, \tau, \mathcal{A})$

Input: T_i, T_j: candidate; τ: similarity threshold
Output: $\mathcal{A} = \{\langle T_i \in \mathcal{T}, T_j \in \mathcal{T}\rangle \mid \text{SIM}(T_i, T_j) \geq \tau\}$

```
1  begin
2      if SIM(T_i, T_j) ≥ τ then A ← ⟨T_i, T_j⟩;
3  end
```

Fig. 6. Strain-Join Algorithm.

On the other hand, the smaller w is, the larger the number of signatures is, the smaller the number of candidate is, and thus the larger the filtering cost is. From the experimental results in Section 6, we can see that w can be set as $\mathtt{D_{max}}$.

Discussion on $\mathcal{T} \neq \mathcal{S}$. We first generate indexing signatures and build indexes for one set (e.g., $\mathcal{T}$). Then for each trajectory in the other set (e.g., $\mathcal{S}$), we generate its probing signatures and threshold-aware signatures, and utilize the cost-based algorithm to find its similar pairs using indexes.

4 Cost-Based Filtering

To compute the candidates of a trajectory T_j, the framework requires to compute $\mathcal{C}_j^\tau$ and $\mathcal{C}_j^k$ for $1 \leq k \leq |T_j|$. The computation of these candidate sets rely on calculating the union of the inverted lists of threshold-aware signatures and probing signatures in T_j. It is expensive to compute the union if the candidate set is rather large. On the other hand, any candidate set in $\mathcal{C}_j^\tau$ and $\mathcal{C}_j^k$ can be used to generate the candidates. We can either use all of them or only utilize one set to compute the candidate. The trade-off is that the more candidate sets we use, the less candidates (by computing their intersection) and the less verification cost. On the contrary, the less candidate sets we use, the more candidates and the more verification cost. Thus we want to select high-quality candidate sets to achieve high overall performance. To address this problem, we first propose a context-based selection method (Section 4.1) and then present a cost-based algorithm (Section 4.2).

4.1 Context-Based Candidate Set Selection

The sample points have some correlations: given two sample points p_j^k and $p_j^{k'}$, their candidate sets $\mathcal{C}_j^k$ and $\mathcal{C}_j^{k'}$ may have large overlap, i..e, $\mathcal{C}_j^k$ and $\mathcal{C}_j^{k'}$ nearly have the same size with $\mathcal{C}_j^k \cap \mathcal{C}_j^{k'}$. Thus we do not want to use both $\mathcal{C}_j^k$ and $\mathcal{C}_j^{k'}$. To address this issue, we propose a context-based candidate set selection method.

In Figure 2(e), for trajectory T_3, we have $\mathcal{C}_3^4 = \{T_1\}$ for p_3^4, $\mathcal{C}_3^5 = \{T_1\}$ for p_3^5, and we can use one of them. For trajectory T_5, we have $\mathcal{C}_5^2 = \{T_1, T_3, T_4\}$ for p_5^2, and $\mathcal{C}_5^6 = \{T_2, T_4\}$ for p_5^6, the intersection of these two candidate sets only contains one element: $\{T_4\}$, and we can use both $\mathcal{C}_5^2$ and $\mathcal{C}_5^6$.

Sample Point Correlation. Given two adjacent points p_j^k and p_j^{k+1} on a trajectory T_j, if most trajectories containing p_j^k also contain p_j^{k+1}, then p_j^k and p_j^{k+1} have high correlation. Formally, let $\text{COR}(p_j^k, p_j^{k+1})$ denote the correlation of p_j^k and p_j^{k+1}, which is defined as

$$\text{COR}(p_j^k, p_j^{k+1}) = \frac{\text{CNT}_{k,k+1}}{\text{CNT}_k}, \tag{11}$$

where CNT_k is the number of trajectories containing sample point p_j^k and $\text{CNT}_{k,k+1}$ is the number of trajectories adjacently containing both p_j^k and p_j^{k+1}. If $\text{COR}(p_j^k, p_j^{k+1})$ is large, then p_j^k and p_j^{k+1} are contextually correlated, and we do not need to select both of them.

In Figure 2(c), for trajectory T_3, $\text{COR}(p_3^4, p_3^5) = 1$, thus points p_3^4 and p_3^5 are highly correlated, there is no need to select both points for candidate set generation. For trajectory T_5 in Figure 2(e), $\text{COR}(p_5^4, p_5^7) = 1/3$, meaning if both points are selected, the candidate set can be reduced to one third.

Context-Aware Candidate Set Selection. Given a trajectory T_j, we group its sample points as follows. First, we generate the group with the first sample point p_j^1, i.e., $\mathcal{Z}_j^1 = \{p_j^1\}$. Next if $\text{COR}(p_j^1, p_j^2) \geq \theta$ (we will introduce how to set θ later), we add p_j^2 into $\mathcal{Z}_j^1$ and check whether p_j^3 can be added into $\mathcal{Z}_j^1$; otherwise we generate a new group $\mathcal{Z}_j^2 = \{p_j^2\}$ and check whether p_j^3 can be added into $\mathcal{Z}_j^2$. Iteratively we can group the sample points to different groups $\mathcal{Z}_j^1, \mathcal{Z}_j^2, \cdots, \mathcal{Z}_j^{|\mathcal{Z}_j|}$. From each group, we can select

the candidate set of any sample as a selected candidate set. As the smaller the size of the candidate set, the better, we want to select the candidate set with the minimal size. Thus the selected candidate set is $\mathcal{S} = \{\mathcal{S}_j^1, \mathcal{S}_j^2, \cdots, \mathcal{S}_j^{|\mathcal{S}|}\}$ where

$$\mathcal{S}_j^k = \arg \min_{\mathcal{C}_j^{k*} \in \mathcal{Z}_j^k} |\mathcal{C}_j^{k*}|. \quad (12)$$

Computing The Correlation $\text{COR}(p_j^k, p_j^{k+1})$. A brute-force method first computes the frequency of every sample point (CNT_k), and then enumerates every pair of two adjacent sample points in a trajectory and computes the frequency of each pair ($\text{CNT}_{k,k+1}$). However this method takes $\mathcal{O}(\sum_{T_i \in \mathcal{T}} |T_i|^2)$ space and time complexities. To address this issue, we propose a linear approach.

Given a trajectory T_j, for each sample point p_j^k, we check whether it is a *turning point*, i.e., whether there are more than three outgoing points from p_j^k. If yes, p_j^k is a turning point and p_j^k's outgoing points may have low correlation with p_j^k; otherwise, the trajectories passing p_j^k must pass its outgoing points, and p_j^k's outgoing points have high correlation with p_j^k. Thus we set $\theta = 1/2$ to indicate that if more than half of the trajectories pass both p_j^k and p_j^{k+1}, then these two points are highly correlated.

In road networks, each point has a limited number of outgoing points, say 2-5. As the trajectory should be aligned to the road network, the sample point also has a limited number of outgoing points. Due to the trajectories having measurement errors, we use eight directions around a sample point to keep the outgoing points as follows.

For each point p_j^k, we consider the 8 grids around p_j^k. We compute the number of p_j^k's subsequent sample points that fall in these grids. For trajectory T_j, p_j^{k+1} falls in one of these grids and suppose p_j^{k+1} falls in grid r. Let CNT_r denote the number of trajectories (visited before T_j) that contain both point p_j^k's corresponding grid and grid r. Thus we can utilize the ratio of sub-region's frequency (CNT_r) to the frequency of p_j^k (CNT_k) to estimate $\text{COR}(p_j^k, p_j^{k+1})$, i.e.,

$$\text{COR}(p_j^k, p_j^{k+1}) \approx \frac{\text{CNT}_r}{\text{CNT}_k}, \quad (13)$$

The time and space complexity is $\mathcal{O}(\sum_{T_i \in \mathcal{T}} |T_i|)$.

For example, consider T_5 in Figure 2(e). The sample points are grouped into $\mathcal{Z}_5^1 = \{p_5^1, p_5^2, p_5^3, p_5^4\}$ and $\mathcal{Z}_5^2 = \{p_5^5, p_5^6, p_5^7\}$ according to the COR. Then we get $\mathcal{S}_1^5 = \{T_1, T_3\}$ and $\mathcal{S}_2^5 = \{T_2\}$. As $\mathcal{S}_1^5 \cap \mathcal{S}_2^5 = \phi$, there is no candidate for T_5.

4.2 Cost-Based Candidate Set Selection

Given the threshold-aware candidate set $\mathcal{C}_j^\tau$ and the context-aware candidate set $\mathcal{S}$, we consider how to utilize them to generate the final candidate set. First, given a candidate set $\mathcal{C}_j^\tau$ or $\mathcal{S}_j^k$, we need to compute the union of the inverted lists of signatures. The complexities are respectively $\sum_{g \in \mathcal{C}_j^\tau} |g|$ and $\sum_{g \in \mathcal{S}_j^k} |g|$. We sort the complexity in ascending order. For simplicity, suppose the $|\mathcal{S}| + 1$ sorted candidate sets are $\mathcal{S}_j^0, \mathcal{S}_j^1, \cdots, \mathcal{S}_j^{|\mathcal{S}|}$. Note that we do not need to compute the union and we only need to compute the sum of the inverted-list size of grids in them. Thus it is efficient to get the order.

Then we propose a cost-based algorithm. We first compute the union set of $\mathcal{S}_j^0$ with the minimal union cost. Next we have two strategies to compute the final answer.

Algorithm 2: COSTBASEDFILTERING(T_j, $\mathcal{G}$, τ, $\mathtt{D_{max}}$)

Input: T_j: a trajectory; $\mathcal{G}$: the grid index;
τ: similarity threshold;
$\mathtt{D_{max}}$: maximal trajectory distance
Output: $\mathcal{C}$: candidate set for T_j

```
begin
    if Σ_{k=1}^{|T_j|} d_j^k > τ_j then
        Generate candidate set C_j^τ = ∪_{g∈G^τ(T_j)} L(g);
    Generate context-aware sets: S_j^1, ..., S_j^{|S_j|};
    Sort C_j^τ, S_j^k: |S_j^0| ≤ |S_j^1| ≤ ... ≤ |S_j^{|S_j|}|;
    k = 0; C = S_j^0;
    COST_V = |S_j^0||T_max|^2;
    COST_F = Σ_{g∈S_j^1} |L(g)| + |S_j^0 ∩ S_j^1||T_max|^2;
    while COST_V > COST_F do
        k = k + 1;
        C = C ∩ S_j^k;
        COST_V = |C||T_max|^2;
        COST_F = Σ_{g∈S_j^{k+1}} |L(g)| + |C ∩ S_j^{k+1}||T_max|^2;
    return C
end
```

Fig. 7. Cost-Based Filtering Algorithm.

Method 1: Direct Verification. We directly verify the candidates in $\mathcal{S}_j^0$, and the cost is

$$\text{COST}_\text{V} = |\mathcal{S}_j^0||T_{max}|^2. \quad (14)$$

Method 2: Further Filtering. We compute the candidate set $\mathcal{S}_j^1$ and intersect $\mathcal{S}_j^1$ with $\mathcal{S}_j^0$ to compute $\mathcal{S}_j^0 \cap \mathcal{S}_j^1$. Then we verify candidates in $\mathcal{S}_j^0 \cap \mathcal{S}_j^1$, and the cost is

$$\text{COST}_\text{F} = \sum_{g \in \mathcal{S}_j^1} |\mathcal{L}(g)| + |\mathcal{S}_j^0 \cap \mathcal{S}_j^1||T_{max}|^2. \quad (15)$$

In the equation, we can easily compute $\sum_{g \in \mathcal{S}_j^1} |\mathcal{L}(g)|$ but it is expensive to compute $|\mathcal{S}_j^0 \cap \mathcal{S}_j^1|$ and we need to estimate $|\mathcal{S}_j^0 \cap \mathcal{S}_j^1|$. To address this issue, we can utilize a sampling based method by (1) selecting a sample of $\mathcal{S}_j^1$ and (2) checking whether trajectories in the sample appear in $\mathcal{S}_j^0$; if so, $\mathcal{S}_j^0$ and $\mathcal{S}_j^1$ have high possibility to overlap each other and we do not have to compute their intersection set.

Cost-Based Method. If $\text{COST}_\text{V} \leq \text{COST}_\text{F}$, we use the first method and the algorithm terminates; otherwise, we employ the second method, and then iteratively check whether we use $\mathcal{S}_j^2$ by comparing the two methods. Finally, we compute all the answers. Algorithm 2 shows the pseudo code. For example, consider a new trajectory T_{10}. Suppose $\mathcal{S}_{10}^0 = \{T_1, T_5, T_8\}$, $\mathcal{S}_{10}^1 = \{\underline{T_2}, T_3, \underline{T_5}, T_8\}$, $\mathcal{S}_{10}^2 = \{\underline{T_5}, T_7, \underline{T_8}, T_9\}$ and $\mathcal{S}_{10}^3 = \{T_1, T_2, T_4, \underline{T_5}, T_8\}$. We use T_2 and T_5 as samples from $\mathcal{S}_{10}^1$. As only T_5 appears in $\mathcal{S}_{10}^0$, $\mathcal{S}_{10}^0$ and $\mathcal{S}_{10}^1$ do not overlap much, and COST_F is cheaper, then the candidate set $\mathcal{C} = \mathcal{S}_{10}^0 \cap \mathcal{S}_{10}^1 = \{T_5, T_8\}$. Next we check $\mathcal{S}_{10}^2$, we use T_5 and T_8 as samples and both are contained by $\mathcal{C}$. Thus COST_V is cheaper. We use $\mathcal{C} = \{T_5, T_8\}$ as the final candidate set.

5 EXPANSION-BASED VERIFICATION

To verify whether a candidate pair $\langle T_i, T_j \rangle$ is an answer, it is expensive to directly compute the similarity. To address

this issue, we first propose a signature-based method (Section 5.1), which utilizes the signature to verify the candidate pair. Since different signatures have different importances on the verification, we propose an expansion based method (Section 5.2), which only uses a subset of signatures to verify a candidate pair. Finally, we propose a bound-based method to further improve the performance (Section 5.3), which first estimates the upper bound and lower bound of the real similarity and then utilizes the two bounds to improve the verification cost.

5.1 Signature-Based Framework

Consider a candidate pair $\langle T_i, T_j \rangle$. Based on the similarity function (Equation 3), we only need to compute $d^k_{i\to j}$ and $d^k_{j\to i}$, and if

$$\sum_{k=1}^{|T_i|} d^k_{i\to j} + \sum_{k=1}^{|T_j|} d^k_{j\to i} \le (1-\tau)(|T_i|+|T_j|), \qquad (16)$$

then T_i and T_j are similar; dissimilar otherwise.

We first discuss how to compute $d^k_{i\to j}$ and the technique can be used to compute $d^k_{j\to i}$. We generate the indexing signature of T_j, $\mathcal{G}^x(T_j)$, and build a hash table for $\mathcal{G}^x(T_j)$. We then generate probing signature of each sample point p^k_i in T_i, $\mathcal{G}^p(p^k_i)$. Then we compute $d^k_{i\to j}$ for p^k_i.

For each grid g in $\mathcal{G}^p(p^k_i)$, if g is not in $\mathcal{G}^x(T_j)$, $d^k_{i\to j}$ is very large, and T_i and T_j are not similar. If g is in $\mathcal{G}^x(T_j)$, we compute the minimal distance from p^k_i to the lines of T_j falling in g, i.e.,

$$\texttt{Dist}_{\texttt{PT}}(p^k_i, T_j|g) = \min_{l\in T_j\cap g} \texttt{Dist}_{\texttt{PL}}(p^k_i, l). \qquad (17)$$

where l is a line in T_j falling in g.

If multiple grids appear in $\mathcal{G}^x(T_j)$, we compute the minimal distance of $\texttt{Dist}_{\texttt{PT}}(p^k_i, T_j|g)$ among grids in $\mathcal{G}^p(p^k_i)$, i.e.,

$$d^k_{i\to j} = \min_{g\in\mathcal{G}^p(p^k_i)\cap\mathcal{G}^x(T_j)} \frac{\texttt{Dist}_{\texttt{PL}}(p^k_i, T_j|g)}{\texttt{D}_{\texttt{max}}}. \qquad (18)$$

Thus we do not need to consider all the lines in T_j for p^k_i. Instead we only consider the lines falling in $\mathcal{G}^p(p^k_i)\cap\mathcal{G}^x(T_j)$.

For example, we want to compute $d^5_{4\to 1}$ from point p^5_4 of T_4 to T_1. As displayed in Figure 8(a), the nine grids are $\mathcal{G}^p(p^5_4)$ and the three shaded grids are $\mathcal{G}^p(p^5_4)\cap\mathcal{G}^x(T_1)$. As T_1 passes three of the nine grids of p^5_4's probing signature: g_{23}, g_{24} and g_{35}, we only need to compute three distances denoted by the three line from p^5_4 to T_1 in the figure: $\texttt{Dist}_{\texttt{PL}}(p^5_4, T_1|g_{23})$, $\texttt{Dist}_{\texttt{PL}}(p^5_4, T_1|g_{24})$(the minimum) and $\texttt{Dist}_{\texttt{PL}}(p^5_4, T_1|g_{35})$. Thus $d^5_{4\to 1} = \texttt{Dist}_{\texttt{PL}}(p^5_4, T_1|g_{24})/\texttt{D}_{\texttt{max}}$.

Complexity. The number of grids in $\mathcal{G}^p(p^k_i)$ is small, i.e., $|\mathcal{G}^p(p^k_i)| = (2\frac{\texttt{D}_{\texttt{max}}}{w}+1)^2$, which can be taken as an constant. Thus $\mathcal{G}^p(p^k_i)\cap\mathcal{G}^x(T_j)$ is a constant. If each grid has only one line in T_j, then the complexity to verify a candidate pair is $\mathcal{O}(|T_i|+|T_j|)$. Note that a trajectory will not appear many times in a grid, and thus this method is much faster than the straightforward method that directly computes $\texttt{Dist}_{\texttt{PT}}(p^k_i, T_j)$.

5.2 Expansion-Based Method

The signature-based method requires to enumerate every grid in $\mathcal{G}^p(p^k_i)\cap\mathcal{G}^x(T_j)$. However different grids have different importances to compute $d^k_{i\to j}$. In other words, for a grid, if its minimal distance to p^k_i is already very large, i.e., $\frac{\textsc{MinDist}(p^k_i, g)}{\texttt{D}_{\texttt{max}}} > d^k_{i\to j}$, we do not need to compute $\texttt{Dist}_{\texttt{PT}}(p^k_i, T_j|g)$. For example, in Figure 8(b), the minimal distance between grid g_{23} and point p^5_4 ($\textsc{MinDist}$ (p^5_4, g_{23})) is large and we can avoid the computation on such grids.

To this end, we can prioritize the girds based on the distance to p^k_i. We observe that the closer a grid is to p^k_i, the smaller the distances of lines of T_j in the grid are to p^k_i. Thus we can simply use the nearby grids and prune the far grids. Next we propose an expansion-based method.

Given a sample point p^k_i, we access its probing signatures in $\mathcal{G}^p(p^k_i)$ by $\textsc{MinDist}(p^k_i, g\in\mathcal{G}^p(p^k_i))$ in ascending order. Suppose the ordered grids are $g_1, g_2, \cdots, g_{|\mathcal{G}^p(p^k_i)|}$. If $g_1\in\mathcal{G}^x(T_j)$, we compute $\texttt{Dist}_{\texttt{PT}}(p^k_i, T_j|g_1)$.
(1) If $\texttt{Dist}_{\texttt{PT}}(p^k_i, T_j|g_1) \le \textsc{MinDist}(p^k_i, g_2)$,

$$d^k_{i\to j} = \frac{\texttt{Dist}_{\texttt{PT}}(p^k_i, T_j|g_1)}{\texttt{D}_{\texttt{max}}};$$

(2) Otherwise we compute $\texttt{Dist}_{\texttt{PT}}(p^k_i, T_j|g_2)$. (2.1) If $\min(\texttt{Dist}_{\texttt{PT}}(p^k_i, T_j|g_1), \texttt{Dist}_{\texttt{PT}}(p^k_i, T_j|g_2)) \le \textsc{MinDist}(p^k_i, g_3)$,

$$d^k_{i\to j} = \min(\frac{\texttt{Dist}_{\texttt{PT}}(p^k_i, T_j|g_1)}{\texttt{D}_{\texttt{max}}}, \frac{\texttt{Dist}_{\texttt{PT}}(p^k_i, T_j|g_2)}{\texttt{D}_{\texttt{max}}});$$

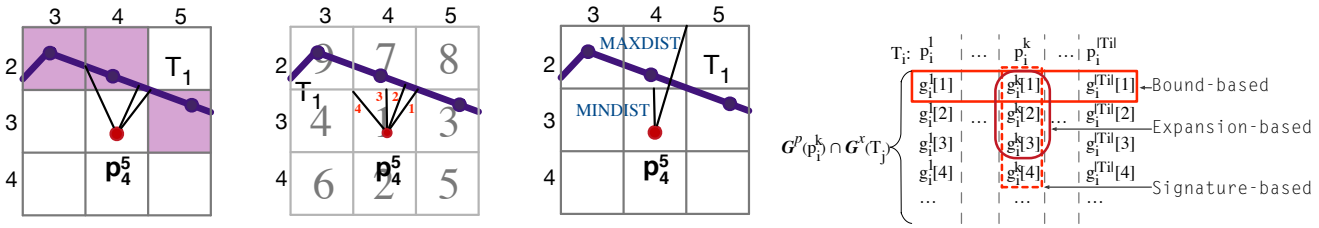
(2.2) otherwise we need to compute $\texttt{Dist}_{\texttt{PT}}(p^k_i, T_j|g_3)$.

Iteratively we can compute $\texttt{Dist}_{\texttt{PT}}(p^k_i, T_j)$ without needing to enumerate all grids. In Figure 8(b), grids are ordered by $\textsc{MinDist}$ to p^5_4 and labeled in the background accordingly, the grid in the center has the minimal $\textsc{MinDist}$ to p^5_4. T_1 goes through three of p^5_4's probing signature, i.e., g_{35}, g_{24} and g_{23} (the $3^{rd}, 7^{th}$ and 9^{th} nearest grids). The expansion-based method only uses two of them to compute $d^5_{4\to 1}$: we first check the 3^{rd} grid g_{35} and compute $\texttt{Dist}_{\texttt{PT}}(p^5_4, T_1|g_{35})$(line '1'), then we compare $\texttt{Dist}_{\texttt{PT}}(p^5_4, T_1|g_{35})$ and $\textsc{MinDist}(p^5_4, g_{24})$ (line '3'). As $\texttt{Dist}_{\texttt{PT}}(p^5_4, T_1|g_{35}) > \textsc{MinDist}(p^5_4, g_{24})$, we compute $\texttt{Dist}_{\texttt{PT}}(p^5_4, T_1|g_{24})$(line '2'), as it is smaller than $\texttt{Dist}_{\texttt{PT}}(p^5_4, T_1|g_{35})$, we use it to compare with $\textsc{MinDist}(p^5_4, g_{23})$ (line '4'). As $\texttt{Dist}_{\texttt{PT}}(p^5_4, T_1|g_{24}) < \textsc{MinDist}(p^5_4, g_{23})$, we get $d^5_{4\to 1} = \texttt{Dist}_{\texttt{PT}}(p^5_4, T_1|g_{24})/\texttt{D}_{\texttt{max}}$. $\texttt{Dist}_{\texttt{PT}}(p^5_4, T_1|g_{23})$ in grid g_{23} is thus avoided to calculate.

Get Ordred Grids. Given a point p^k_i, the first grid is $g(p^k_i)$. Then we can get the ordered grids by visiting its eight surrounding grids by computing the distance from p^k_i to the four sides and four end points of g (p^k_i), as shown in Figure 8. Then based on the distances to the eight grids, we can get next closest grids by visiting their neighbor grids.

5.3 Bound-Based Pruning

It is expensive to compute the real value of $d^k_{i\to j}$ as it involves complicated mathematical operations. Instead, we can estimate a lower bound of $d^k_{i\to j}$, denoted by $\texttt{lb}^k_{i\to j}$, and based on Equation 16, if the sum of the lower bound, i.e., $\sum_k \texttt{lb}^k_{i\to j} + \sum_k \texttt{lb}^k_{j\to i}$, is larger than $(1-\tau)(|T_i|+|T_j|)$, the pair cannot be similar and we can prune the pair. Similarly, we can estimate an upper bound of $d^k_{i\to j}$, denoted by $\texttt{ub}^k_{i\to j}$, and based on Equation 16, if the sum of the upper bound, i.e., $\sum_k \texttt{ub}^k_{i\to j} + \sum_k \texttt{ub}^k_{j\to i}$, is smaller than $(1-\tau)(|T_i|+|T_j|)$, the pair must be an answer and we do not need to compute the real similarity. Next we discuss how to estimate the lower bound and upper bound of $d^k_{i\to j}$.



(a) Signature-based (b) Expansion-based (c) Bound-based (d) Verification Comparison

Fig. 8. Signature-Based Verification.

Distance from p_i^k to lines of T_j in a grid g. Let $\hat{d}^k_{i\to j}[g_x]$ denote the minimal distance from p_i^k to lines of T_j in g_x, i.e,

$$\hat{d}^k_{i\to j}[g_x] = \texttt{Dist}_{\texttt{PT}}(p_i^k, T_j|g_x). \tag{19}$$

Estimating a lower bound of $d^k_{i\to j}$. Consider p_i^k. Suppose $g_1, g_2, \cdots, g_{|\mathcal{G}^x(T_j)\cap\mathcal{G}^p(p_i^k)|}$ denote the grids in $\mathcal{G}^x(T_j)\cap\mathcal{G}^p(p_i^k)$ sorted by MinDist(p_i^k, g) in an ascending order.

As MinDist$(p_i^k, g_1) \le$ MinDist$(p_i^k, g_x) \le \hat{d}^k_{i\to j}[g_x]$ for any $x > 1$, we can get a lower bound

$$\texttt{lb}^k_{i\to j}[g_1] = \frac{\text{MinDist}(p_i^k, g_1)}{\texttt{D}_{\max}}. \tag{20}$$

Next for g_2, we get a lower bound,

$$\texttt{lb}^k_{i\to j}[g_2] = \min(\hat{d}^k_{i\to j}[g_1], \frac{\text{MinDist}(p_i^k, g_2)}{\texttt{D}_{\max}}); \tag{21}$$

If $\hat{d}^k_{i\to j}[g_1] \le \frac{\text{MinDist}(p_i^k, g_2)}{\texttt{D}_{\max}}$, $d^k_{i\to j} = \hat{d}^k_{i\to j}[g_1]$ and we prune grids after g_2 and do not need to estimate a lower bound.

Let $\hat{d}^k_{i\to j}[g_1\cdots g_t] = \min_{1\le x\le t}\frac{\text{MinDist}(p_i^k, g_x)}{\texttt{D}_{\max}}$ denote the minimal distance from lines in the first t grids to p_i^k. We can estimate a lower bound,

$$\texttt{lb}^k_{i\to j}[g_{t+1}] = \min(\hat{d}^k_{i\to j}[g_1\cdots g_t], \frac{\text{MinDist}(p_i^k, g_{t+1})}{\texttt{D}_{\max}}). \tag{22}$$

In Figure 8(c), we estimate the lower bound $\texttt{lb}^5_{4\to 1}$ for point p_4^5 of trajectory T_4 to the line of T_1 in g_{24}. The line from point p_4^5 to the lower horizontal side of g_{24} is MinDist(p_4^5, g_{24}). We compare $\hat{d}^5_{4\to 1}[g_{24}]$ and MinDist(p_4^5, g_{24}). As MinDist$(p_4^3, g_{24})/\texttt{D}_{\max}$ is smaller, it is assigned to $\texttt{lb}^5_{4\to 1}[g_{24}]$. Iteratively, we can compute the lower bound for every grid.

Estimating an upper bound of $d^k_{i\to j}$. For any grid g, we can estimate an upper bound, MaxDist(p_i^k, g), from p_i^k to any location in g, where

$$\text{MaxDist}(p_j^k, g) = \max_v \text{Dist}(p_j^k, v), \tag{23}$$

where v is a vertex of g (g has four vertices).

Moreover, for any t, we can estimate a tighter upper bound,

$$\texttt{ub}^k_{i\to j}[g_1, \cdots, g_t] = \min_{1\le x\le t}\text{MaxDist}(p_j^k, g_x). \tag{24}$$

Thus for g_1, we can get an upper bound

$$\texttt{ub}^k_{i\to j}[g_1] = \frac{\text{MaxDist}(p_i^k, g_1)}{\texttt{D}_{\max}}. \tag{25}$$

Next for g_2, we get an upper bound,

$$\texttt{ub}^k_{i\to j}[g_2] = \min(\hat{d}^k_{i\to j}[g_1], \frac{\text{MaxDist}(p_i^k, g_2)}{\texttt{D}_{\max}}); \tag{26}$$

Algorithm 3: SignatureVerification$(T_i, T_j, \tau, \mathcal{A})$

Input: T_i, T_j: candidate; τ: similarity threshold
Output: $\mathcal{A} = \{\langle T_i \in \mathcal{T}, T_j \in \mathcal{T}\rangle \mid \text{Sim}(T_i, T_j) \ge \tau\}$

```
begin
    Compute G^x(T_i) and G^x(T_j);
    Get ordered grids in G^p(p_i^k) ∩ G^x(T_j) by MinDist;
    Get ordered grids in G^p(p_j^k) ∩ G^x(T_i) by MinDist;
    for 1 ≤ t ≤ max_k |G^p(p_i^k) ∩ G^x(T_j)| do
        for 1 ≤ k ≤ max(|T_i|, |T_j|) do
            if Σ_k lb^k_{i→j}[g_t] + lb^k_{j→i}[g_t] > (1−τ)(|T_i|+|T_j|)
            then T_i and T_j cannot be similar, return; if
            Σ_k ub^k_{i→j}[g_i] + ub^k_{i→j}[g_i] ≤ (1−τ)(|T_i|+|T_j|)
            then A = A ∪ {⟨T_i, T_j⟩}, return;
    Sim(T_i, T_j) = 1 − (Σ_k d̂^k_{i→j}[g_t] + d̂^k_{i→j}[g_t]) / (|T_i|+|T_j|);
    if Sim(T_i, T_j) ≥ τ then
        A = A ∪ {⟨T_i, T_j⟩}, return;
end
```

Fig. 9. Signature-based Verification Algorithm.

Then for g_{t+1}, we can estimate an upper bound,

$$\texttt{ub}^k_{i\to j}[g_{t+1}] = \min(\hat{d}^k_{i\to j}[g_1\cdots g_t], \frac{\text{MaxDist}(p_i^k, g_{t+1})}{\texttt{D}_{\max}}). \tag{27}$$

In Figure 8(c), we estimate the upper bound $\texttt{ub}^5_{4\to 1}$ for point p_4^5 of T_4 to the line of T_1 in grid g_{24}. The line from p_4^5 to the upper-right vertex of g_{24} represents MaxDist(p_4^5, g_{24}), and the upper bound $\texttt{ub}^5_{4\to 1}[g_{24}]$ is directly computed using Equation 25. For grid g_{23}, since $\hat{d}^5_{4\to 1}[g_{24}]$ is smaller than MaxDist$(p_4^5, g_{23})/\texttt{D}_{\max}$, we can get a tighter upper bound: $\texttt{ub}^5_{4\to 1}[g_{23}] = \hat{d}^k_{i\to j}[g_{24}]$. Iteratively, we can compute the upper bound for every grid.

Bound-Based Algorithm. Based on the lower bounds and upper bounds, we propose an expansion-based method. For any point p_i^k, we access its nearby grids in order by MinDist(p_i^k, g) and remove the grids that are not in $\mathcal{G}^x(T_j)$. Suppose the ordered grids are $g_1, g_2, \cdots$.

Then we compute $\texttt{lb}^k_{i\to j}[g_1]$ and $\texttt{ub}^k_{i\to j}[g_1]$.

(1) **Pruning**: If $\sum_k \texttt{lb}^k_{i\to j}[g_1] + \sum_k \texttt{lb}^k_{j\to i}[g_1] > (1-\tau)(|T_i|+|T_j|)$, $\langle T_i, T_j\rangle$ cannot be similar and we prune it.

(2) **Early Termination**: If $\sum_k \texttt{ub}^k_{i\to j}[g_1] + \sum_k \texttt{ub}^k_{i\to j}[g_1] \le (1-\tau)(|T_i|+|T_j|)$, $\langle T_i, T_j\rangle$ is similar, we can early terminate.

If we can neither early terminate nor prune, we compute $\hat{d}^k_{i\to j}[g_1]$. Next we compute $\texttt{lb}^k_{i\to j}[g_2]$ and $\texttt{ub}^k_{i\to j}[g_2]$, and repeat above steps. Algorithm 3 shows the pseudo code. For example, we verify candidate $\langle T_1, T_3\rangle$ in Figure 2. We first prepare $\mathcal{G}^x(T_1)$ and $\mathcal{G}^x(T_3)$ and get grids in $\mathcal{G}^p(p_1^k)\cap\mathcal{G}^x(T_3)$ for $p_1^k \in T_1$ and $\mathcal{G}^p(p_3^k)\cap\mathcal{G}^x(T_1)$ for $p_3^k \in T_3$ ordered

by MINDIST. Then, we estimate upper bound for each point's nearest grid, as $\sum_k \mathtt{ub}^k_{1\to3}[g_1] + \sum_k \mathtt{ub}^k_{3\to1}[g_1]$ is already smaller than $(1-\tau)(|T_1|+|T_3|)$, T_1 and T_3 must be similar, and we can early terminate. For candidate $\langle T_1, T_4\rangle$, we estimate lower bound for each point's nearest grids in ascending order. Suppose $\sum_k \mathtt{lb}^k_{1\to4}[g_2] + \sum_k \mathtt{lb}^k_{4\to1}[g_2]$ is larger than $(1-\tau)(|T_1|+|T_4|)$, we can safely prune $\langle T_1, T_4\rangle$.

Comparison of three verification techniques. Figure 8(d) illustrates the three verification techniques. We use $g_i^k[t]$ to denote the t^{th} nearest grids to p_i^k. For candidate pair $\langle T_i, T_j\rangle$, the signature-based method computes distances from each point p_i^k of T_i to T_j in any grids in $\mathcal{G}^p(p_i^k) \cap \mathcal{G}^x(T_j)$. The expansion-based method can prune some unnecessary grids. Both techniques work in a 'vertical' style, as they examine sample points one by one. The bound-based verification operates in a 'horizontal' style, and it uses grids $g_i^k[t]$ for every k to get an overall upper/lower bound to decide pruning or early termination.

6 EXPERIMENTS

We have implemented our method and conducted an extensive set of experimental studies, in order to (1) verify our proposed techniques; and (2) compare our method with state-of-the-art studies.

Datasets. We use two real vehicular datasets: Beijing Taxi (`Taxi`, www.datatang.com/data/45888) and Shenzhou Zhuanche (`UCar`, like Uber, zhuanche.zuche.com). Table 2 shows the statistics of the two datasets. `Taxi` contains trajectories generated by more than 8,000 public taxicabs in Beijing; `UCar` contains trajectories generated by nearly 2,000 cars within one week in Beijing. We also use the Australian Sign Language (`ASL`) dataset, where 98 different signs such as "all", "go", are expressed by hand movement trajectories.

Experimental Setting. All of the algorithms were implemented in C++. All the experiments were conducted in a machine with 2.10 GHz Intel Xeon CPU E5-2620, 64 GB RAM, running Ubuntu 13.4.

6.1 Evaluating Filtering

In this section, we evaluate candidate set generation techniques. We implement four strategies for `Strain-Join`: (1) Use the probing signature of one sample point for each trajectory (`One-Sig`). (2) Use probing signatures of all sample points for each trajectory (`All-Sig`). (3) Use the threshold-aware signature (τ`-Sig`). (4) Use the cost-based candidate set selection method (`Cost-Sig`). We compare candidate set size and running time by varying the three parameters: threshold τ, maximal distance $\mathtt{D_{max}}$ and grid width w. Figures 10(a)-10(f) show the results on `Taxi` dataset and Figures 11(a)-11(f) show the results on the `UCar` dataset. Note that we use the best verification algorithm (bound-based algorithm) to verify the candidate to report the elapsed time.

We have the following observations. Firstly, `One-Sig` generates the largest number of candidates and `All-Sig` generates the smallest number of candidates. `Cost-Sig` generates smaller number of candidates than τ`-Sig` (sub-figures 10(a)-10(c) and 11(a)-11(c)). The results are consistent with our theoretical analysis, because `All-Sig` utilizes all signatures, `One-Sig` utilizes only one signature, and τ`-Sig` and `Cost-Sig` selects high-quality signatures. `Cost-Sig` is better than τ`-Sig` because it uses cost-based method to select the best signatures.

TABLE 2
Trajectory Data Sets

Data Set	# of Traj.	Avg Point #	Max Point #	Min Point #
`Taxi`	200,000	27	50	5
`UCar`	120,000	16	20	3
`ASL`	6757	58	4494	1

Secondly, `One-Sig` takes the longest time, `Cost-Sig` takes the shortest time, and τ`-Sig` is better than `All-Sig` (sub-figures 10(d)-10(f) and 11(d)-11(f)). This is because `One-Sig` generates huge number of candidates and it is expensive to verify the candidates. `All-Sig` takes long time as it is expensive to use all signatures and it is costly to compute the union set of all signatures. `Cost-Sig` outperforms others as it utilizes the cost model to trade-off the filtering and verification time.

Thirdly, with the increase of threshold τ, the number of candidates and elapsed time decrease (sub-figure(a) of Figures 10-11), because for a larger threshold there are smaller numbers of similar pairs. It is easier to find answers for a larger threshold. Note with the increase of τ, the gap between `Cost-Sig` and τ`-Sig` becomes smaller, because for a larger threshold, τ`-Sig` could utilize the threshold to select high-quality signatures. The larger τ, the better quality of the selected signatures.

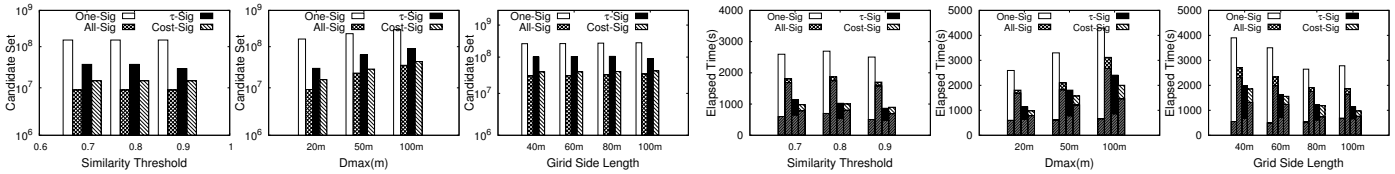
Fourthly, with the increase of $\mathtt{D_{max}}$, the number of candidates and elapsed time increase (sub-figure(b) of Figures 10-11), because there are more similar pairs for a large threshold as we allow matching in a large region. `Cost-Sig` still achieves the best performance.

Fifthly, with the increase of w, the number of candidates increases (sub figure(c) of Figures 10-11), because the fine-grained grids could reduce the number of candidates while coarse-grained grids would increased the number of candidates (as a large grid covered more trajectories and had larger possibilities to include more false positives). However with the increase of w, the elapsed time first increases and then decreases. This is consistent with our theoretical analysis: the larger grids, the more candidates but smaller filtering time; while the smaller grids, the less candidates but larger filtering time. We could see when w is 80m or 100m, it achieves the best performance.

In all, `Cost-Sig` achieves the best performance and we set $w = \mathtt{D_{max}}$. If we prefer recall, we set large $\mathtt{D_{max}}$ and small τ; if we prefer efficiency, we set small $\mathtt{D_{max}}$ and large τ.

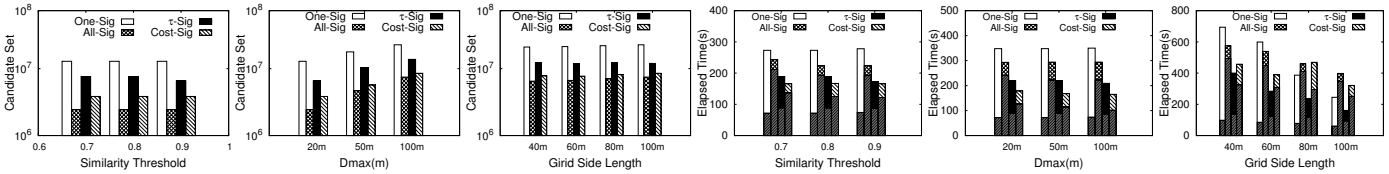
6.2 Evaluating Verification

In this section, we evaluate our verification techniques. We implement four methods: (1) The naive method (`Naive`), which directly computes the similarity for each candidate pair. (2) Signature-based verification (`Signature`), which uses signature grids to compute similarity. (3) Expansion-based verification (`Expansion`), which prunes distant signature grids to avoid unnecessary computation. (4) Bound-based verification (`Bound`), which uses upper and lower bounds to prune dissimilar pairs plus early termination technique. We compare running time by varying the three parameters: threshold τ, maximal distance $\mathtt{D_{max}}$ and grid width w. Figure 12 shows the results on `Taxi` dataset, `UCar` has similar trends, and is omitted due to space constraint.



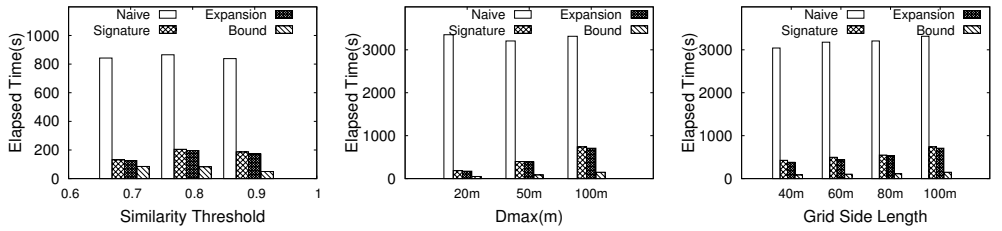
(a) $\mathtt{D_{max}} = w = 20m$ (b) τ=0.7, w=20m (c) $\mathtt{D_{max}}$=100m, τ=0.7 (d) $\mathtt{D_{max}} = w = 20m$ (e) τ=0.7, w=20m (f) $\mathtt{D_{max}}$=100m, τ=0.7

Fig. 10. Evaluating Filtering: Candidate Set Size & Elapsed Time (Taxi).



(a) $\mathtt{D_{max}} = w = 20m$ (b) $\tau = 0.9, w = 20m$ (c) $\mathtt{D_{max}} = 100m, \tau = 0.9$ (d) $\mathtt{D_{max}} = w = 20m$ (e) τ=0.9,w=20m (f) $\mathtt{D_{max}}$=100m, τ=0.9

Fig. 11. Evaluating Filtering: Candidate Set Size & Elapsed Time (UCar).



(a) $\mathtt{D_{max}} = w = 20m$ (b) $\tau = 0.9, w = 100m$ (c) $\mathtt{D_{max}} = 100m, \tau = 0.9$

Fig. 12. Evaluating Verification: Elapsed Time (Taxi)

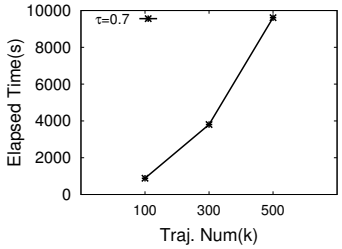
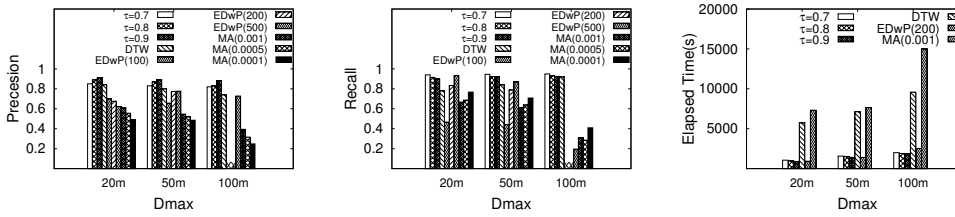
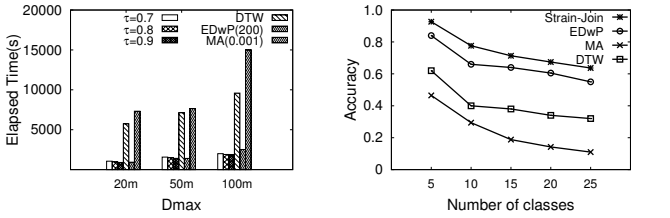


Fig. 13. Scalability($\mathtt{D_{max}}$=w=100m).



(a) Precision (b) Recall



(c) Elapsed Time(s) (d) Accuracy

Fig. 14. Comparison with State-of-the-art Methods.

We have the following observations. Firstly, the Naive method has the worst performance, Bound is most efficient, and Expansion is better than Signature. This is because Naive needs to compute many unnecessary similarity values. Signature is faster than Naive, because Signature decreases the complexity from $\mathcal{O}(|T_{max}|^2)$ to $\mathcal{O}(|T_i| + |T_j|)$ as only grids in $\mathcal{G}^p(p_i^k) \cap \mathcal{G}^x(T_j)$ that contains lines of T_j are used to compute $\text{SIM}(T_i, T_j)$. Expansion runs faster than Signature, because Expansion could avoid the calculation of distance between relatively far grids in $\mathcal{G}^p(p_i^k) \cap \mathcal{G}^x(T_j)$ to each sample point p_i^k. Bound achieves the best performance because it uses upper/lower bounds to prune a great portion of candidate pairs.

Secondly, with the increase of threshold τ, the verification time is reduced (Figure 12(a)), because larger thresholds only allows matching in a small region and thus produces smaller answer sets.

Thirdly, with the increase of $\mathtt{D_{max}}$, the verification costs less time (Figure 12(b)), because larger $\mathtt{D_{max}}$ allows matching in a large region and thus produces larger answer sets.

Fourthly, with the increase of grid size, the verification consumes less time (Figure 12(c)), because bigger grids introduce more trajectories as candidates.

In all, Bound is the best verification technique.

6.3 Comparison with State-of-the-art Methods

In this section, we compare effectiveness of Strain-Join with state-of-the-art approaches: DTW[34], EDwP[28] and MA[30], on precision, recall, runtime efficiency, classification accuracy, as well as the query performance.

Precision and recall. We randomly selected 1000 vehicular trajectories from our dataset and manually labeled similar pairs as the **ground truth**. We compare *precision*, which is the portion of real similar trajectories (as indicated by the ground truth) in all similar trajectories found by a method; and *recall*, which is the ratio of the number of real similar trajectories in the results reported by a method to the number of real similar trajectories in the ground truth. For EDwP, two trajectories are considered similar if their EDwP score is less than a certain value s. We varied $\mathtt{D_{max}}$ from 20m to 100m. For $\mathtt{D_{max}} = 20m$ or $\mathtt{D_{max}} = 50m$, we present the precision and recall results for $s = 100, 200, 500$. For $\mathtt{D_{max}} = 100m$, trajectories are more apart as $100m$ is a rather large threshold, therefore, smaller s values are too tight to generate similar trajectory pairs under the EDwP measurement. Accordingly, we can find similar trajectories for $s = 500$ but not for $s = 100$ or 200. An "x" symbol is used to indicate that there is no such setting in our figures. For MA, the four parameters are set as suggested in [30], and the similarity threshold values tested are 0.001, 0.0005 and 0.0001. Figure 14(a)-(b) show the results.

We could see that Strain-Join has the best performance on both precision and recall. The reasons are as follows. Firstly, our technique allows to align sampling points to any locations (even non-sampling points) to capture the similarity while DTW cannot. Although MA uses gap to capture dissimilar parts, the score for gaps are rather low if the gap points should have been aligned elsewhere. Secondly, for some apparent deviating sample points, DTW stretches the trajectory to match them to another trajectory, and EDwP attempts to use the replace/insert operations which may result in low similarity, while our method can exclude such trajectory pairs early in the filtering step. In addition, a larger τ has better precision but low recall. For example, as τ increases from 0.7 to 0.9, the precision values at $\mathtt{D_{max}} = 20m$ are 0.85, 0.89, 0.91, while the recall values are 0.92, 0.905 and 0.88. This is because a larger τ reduces the denominator of precision and the numerator of recall. In contrast, a larger $\mathtt{D_{max}}$ has lower precision but better recall, because a larger $\mathtt{D_{max}}$ finds more results using a larger region.

Runtime efficiency. We use a dataset of 200,000 vehicular trajectories to test efficiency. For ease of presentation we draw the performance of one threshold for both EDwP and MA. Figure 14(c) indicates that Strain-Join runs much faster than DTW and MA, and is comparable to EDwP due to our high-quality signatures.

Classificaion accuracy. We use the ASL dataset and perform multi-class verification. First we randomly choose c classes(i.e. c signs) and retrieve all the hand movement trajectories under these classes. Then we perform 10-fold cross-validation. We use the nearest neighbors returned by the four metrics to label trajectories. The *accuracy* is the portion of correctly labeled trajectories in all retrieved trajectories. Figure 14(d) presents the variation of accuracy as the number of classes increases from 5 to 25; for Strain-Join, we set $\mathtt{D_{max}} = w = 100m, \tau = 0.8$. We have the following observations. First, our method outperforms others because our method can widely capture the shape information. Meanwhile, the recognition of gap points by MA, as well as the cost of replace/insert operations of EDwP, may impact the similarity results. Second, for all metrics, the accuracy decreases since the searching gets harder as the number of classes increases. Meanwhile, Strain-Join can always achieve the highest accuracy in all cases.

Query performance. For the 200,000-trajectory dataset, the average running time for a single top 1 query of Strain-Join ($\mathtt{D_{max}} = w = 100m, \tau = 0.8$), EDwP (threshold set to 200), MA (threshold set to 0.001), DTW are $7.01, 7, 86, 35.66, 38.26$ milliseconds respectively. Strain-Join is the most efficient method. The precision, recall and accuracy performances have similar trends to the experimental results of the join algorithms.

In all, our method outperforms DTW, EDwP and MA in both quality and efficiency.

6.4 Scalability

In this section, we test the scalability of our method. We vary the number of trajectories on the Taxi dataset. Figure 13 shows the results. We could see that our method scales very well and achieves nearly linear scalability. For $\mathtt{D_{max}} = w = 100m$, τ =0.7, the elapsed time for 100,000 trajectories, 300,000 trajectories, and 500, 000 trajectories are respectively 884 seconds, 3804 seconds, and 9612 seconds.

7 Related Work

7.1 Trajectory Similarity Metrics

A number of trajectory similarity measurement functions have been proposed[1], [2], [5], [6], [7], [10], [11], [15], [23], [20], [24], [25], [28], [30], [32], [34], which can be roughly grouped into two types: (1)The spatial based metrics, such as the Euclidean distance (ED)[10], the Closest-Pair Distance (CPD)[25] and the One Way Distance (OWD)[20]. These metrics directly use the Euclidean distance for corresponding sample point pairs or its variants to define the similarity, and the temporal ordering of sample points are not strictly required when calculating such similarity metrics; and (2)The spatio-temporal metrics, such as the Dynamic Time Warping (DTW)[1], [15], [34], the Longest Common Sub Sequence (LCSS)[32], the Sequence Weighted Alignment model (Swale)[24], the Most Similar Trajectory (DISSIM)[11], the model-driven assignment (MA)[30], and the Edit Distance with Projections (EDwP)[28]. In general, this type of measurements require some form of sample point alignment, i.e., points should be mapped according to the temporal order, to calculate trajectory similarity. Still, time-shifting is allowed, meaning sample points from two trajectories do not need to have the same timestamp.

Spatial-Based Similarity. The Closest-Pair Distance (CPD)[25] is a variation of Euclidean Distance which was introduced to find closest trajectories for given query in spatial networks. The One Way Distance (OWD)[20] focuses on shape similarity for trajectories in grid representations. OWD of two grid trajectories is the sum of distances from the grids where one trajectory's sample points reside in to the grids of the other trajectory. The grid OWD is an estimation of distances between two trajectories which is sensitive to the grid size and was proposed to handle similarity search problem, while our BDS metric measures the exact similarity for two trajectories, the similarity of two trajectories are not affected by the grid size.

They are different from ours: (1) Some of them require uniform sampling rate, which is not practical for real-world datasets; (2) Sample points in a trajectory have to be aligned to sample points (not closest positions) in the other trajectory. However, the vehicles may have different speeds in different times, and thus the sample points may not be well aligned even if two trajectories have the same sampling rate. Our metrics can address these problems by aligning sample points to close locations of other trajectories.

Spatio-Temporal Similarity. The Dynamic Time Wrapping (DTW)[1], [15], [34] distance allows some sample points to repeat in order to achieve the best alignment, i.e., one point in one trajectory can match multiple points in another trajectory. DTW was claimed to be vulnerable to noises since some noise points can introduce large distance between trajectories. However, the authors of [9] and [33] argued and experimentally proved that DTW on average is comparative to other similarity measurements on large data sets. The Longest Common Sub Sequence (LCSS)[32] is used to eliminate the effect of noise points. The LCSS method skips points (taking them as noises) if their distance exceeds a matching threshold. Similar to LCSS, Edit Distance with Real Penalty (ERP)[5] uses a threshold ϵ to quantify a match, and gaps between matched sub-trajectories are assigned penalties to

reveal the dissimilarity. As an improvement, Edit Distance on Real Sequence (EDR)[6] combines the strength of DTW and ERP. It handles time shifting and computes distance using a constant reference point. DISSIM[11] defines dissimilarity of two trajectories as the definite integral of the function of time of the Euclidean distance between two trajectories, which are required to be valid during the same period (i.e., sample points exist in both trajectories for every sampling timestamp). Swale[24] penalizes unmatched points (gaps) and rewards the matching ones. Spatial Assembling Distance (SpADe)[7] is a pattern-based measurement, which finds matching sub-sequences for the whole series (patterns). Threshold Queries (TQuEST)[2] is a threshold-based measurement. The Minkowski sum of two sequences is thus the similarity of two time series. MA[30] is an improvement of DTW, and it is more flexible in aligning points. Similar parts of trajectories contribute a much higher portion to the MA score than the dissimilar parts (gaps). EDwP[28] uses the insert/replace operations to compute similarities between trajectories generated with inconsistent and variable sampling rates.

The difference between MA, EDwP and BDS are: (1) The alignment strategies: MA aims to improve DTW to tolerate the noisy points. To achieve this goal, in MA, sample points of one trajectory can be either aligned to sample points of another trajectory or do not align to any points (which are taken as noisy points). EDwP extends the concept of edit distance and utilizes the insertion operation to equalize the sampling rate difference between trajectories (i.e., given a point in one trajectory, if there is no aligned point in another trajectory, it will insert a new point in the second trajectory). It will give a penalty to an inserted point. Obviously, due to sampling errors, trajectories may miss some points and EDwP will give a high penalty. This will lead to low similarity for two similar trajectories. Meanwhile BDS aligns sample points of one trajectory to their closest locations (not necessarily sample points) on other trajectories to widely capture the shape information. (2)Parameters: MA uses four parameters to compute similarity, which is non-trivial; EDwP requires no parameter and BDS has only one easy-to-set parameter $\mathtt{D}_{\mathtt{max}}$. (3) Handling non-uniform sampling rates: MA employs its asymmetry property, EDwP uses the "insert" operation to equalize the sampling rate, and BDS utilizes closest alignment. In EDwP, two trajectories are aligned by the sampling point order. Given two similar trajectories, if the sample points are out of order due to transmission errors, EDwP will take them as different trajectories. BDS can address this problem by mapping samples to closed points. (4) The modeling of trajectories: for MA, one trajectory is a sequence of discrete points in $\mathbb{R}^d$; EDwP and BDS uses linear interpolation. In fact, BDS can adopt to other feasible interpolations. For example, in an urban road network, we can use the map-matched trajectories to compute similarity. (5) EDwP and MA are asymmetric while BDS is symmetric.

We focus on identifying the trajectories with similar shape. For example, two trajectories with very similar shapes but having asynchronous sample points (due to different traffic conditions) may not be identified as similar by existing metrics; but BDS can address this problem efficiently and effectively. BDS may not work well for the scenarios where the time stamp is important. We leave taking time stamps into consideration as future work.

BDS can handle the case that the trajectories have rather different sampling rates as it aligns sample points to the closest locations on other trajectories. Given varying sampling rates, as long as all the sample points of one trajectory are close enough to another trajectory(and vice versa), the two trajectories are guaranteed to be similar. For low sampling rate trajectories, it is important to match them to road networks. For example, we could use the shortest paths or the most frequent paths to recover the missing parts between sampling points, trying to reveal the real physical shape of each trajectory before we compute their similarities. For vastly different sampling rates, it is rather hard to align them, because a trajectory with low sampling rate can correspond to many trajectories with high sampling rate and it is hard to tell which is better. To address this problem, a common technique is to adopt some assumptions, e.g., taking the shortest path or the most frequent path as the actual path between two sampling points. This is an interesting problem and we leave it as a future work.

7.2 Similarity-Based Trajectory Processing

Similarity join and search is widely studied [17], [14], [35], [36], [18] and recently a number of works are extended to support similarity-based trajectory processing [12], [21]. Lee et. al.[16] propose a two-phase trajectory clustering framework which first partitions trajectories into line segments and then groups similar line segments to find common sub-trajectories, which is helpful in applications such as analysis of regions of interest, etc. The three-step approach of [26] aims to deal with uncertainties in trajectories (such as those introduced by GPS errors) when finding clusters of trajectories. In [13] the convoy discovery in trajectory databases is studied, where a convoy is a group of objects traveling together for a while. Convoys are formalized using density-based notations. Simplified trajectories first form candidate sets and are then finalized if they are actually convoys. The authors of [19] introduce the concept of *swarm*, group of objects that move together but maybe non-consecutively, to find moving-together objects with relaxed conditions. Pruning strategies are proposed to reduce the search space, and a closure checking rule helps to report swarms on demand. Algorithms in [4] try to find similar sub-trajectories with trajectory similarity measure being the average distance at corresponding timestamps. The shortage is if two objects travelled the exact route are at different speeds at corresponding segments, these two trajectories cannot be found as similar ones, while intuitively they are similar trajectories. In contract, our BDS measurement can overcome such issue.

8 Conclusion

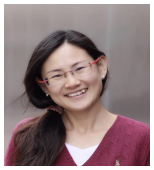
In this paper, we have studied the trajectory similarity join problem. We proposed a new trajectory similarity metric that did not rely on aligning sample points of trajectories. We presented a signature-based method to address the similarity join problem. We first generated high-quality signatures for trajectories and pruned the dissimilar pairs that did not share common signatures. We then devised effective verification algorithms to verify the candidates that

are not pruned. Experimental study on real datasets verified the effectiveness and efficiency of our algorithms.

Acknowledgement. This paper was supported by 973 Program of China (2015CB358700), NSF of China (61373024, 61661166012, 61422205, 61632016, 61472198), Shenzhou, Tencent, FDCT/116/2013/A3, MYRG105 (Y1-L3)-FST13-GZ.

REFERENCES

[1] I. Assent, M. Wichterich, R. Krieger, H. Kremer, and T. Seidl. Anticipatory dtw for efficient similarity search in time series databases. *VLDB*, 2(1):826–837, 2009.
[2] J. Aßfalg, H.-P. Kriegel, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Similarity search on time series based on threshold queries. In *EDBT*, pages 276–294. 2006.
[3] S. Brakatsoulas, D. Pfoser, and N. Tryfona. Practical data management techniques for vehicle tracking data. In *ICDE*, pages 324–325, 2005.
[4] K. Buchin, M. Buchin, M. Van Kreveld, and J. Luo. Finding long and similar parts of trajectories. *Computational Geometry*, 44(9):465–476, 2011.
[5] L. Chen and R. Ng. On the marriage of lp-norms and edit distance. In *VLDB*, pages 792–803, 2004.
[6] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, pages 491–502, 2005.
[7] Y. Chen, M. Nascimento, B. C. Ooi, A. K. Tung, et al. Spade: On shape-based pattern detection in streaming time series. In *ICDE*, pages 786–795, 2007.
[8] P. Cudre-Mauroux, E. Wu, and S. Madden. Trajstore: An adaptive storage system for very large trajectory data sets. In *ICDE*, pages 109–120, 2010.
[9] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *VLDB*, 1(2):1542–1552, 2008.
[10] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, volume 23, 1994.
[11] E. Frentzos, K. Gratsias, and Y. Theodoridis. Index-based most similar trajectory search. In *ICDE*, pages 816–825, 2007.
[12] H. Hu, G. Li, Z. Bao, J. Feng, Y. Wu, Z. Gong, and Y. Xu. Top-k spatio-textual similarity join. *IEEE Trans. Knowl. Data Eng.*, 28(2):551–565, 2016.
[13] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen. Discovery of convoys in trajectory databases. *VLDB*, 1(1):1068–1080, 2008.
[14] Y. Jiang, G. Li, J. Feng, and W. Li. String similarity joins: An experimental evaluation. *PVLDB*, 7(8):625–636, 2014.
[15] E. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and information systems*, 7(3):358–386, 2005.
[16] J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: a partition-and-group framework. In *SIGMOD*, pages 593–604, 2007.
[17] G. Li, D. Deng, J. Wang, and J. Feng. PASS-JOIN: A partition-based method for similarity joins. *PVLDB*, 5(3):253–264, 2011.
[18] G. Li, Y. Wang, T. Wang, and J. Feng. Location-aware publish/subscribe. In *KDD*, pages 802–810, 2013.
[19] Z. Li, B. Ding, J. Han, and R. Kays. Swarm: Mining relaxed temporal moving object clusters. *Proceedings of the VLDB Endowment*, 3(1-2):723–734, 2010.
[20] B. Lin and J. Su. Shapes based trajectory queries for moving objects. In *GIS*, pages 21–30, 2005.
[21] S. Liu, G. Li, and J. Feng. A prefix-filter based method for spatio-textual similarity join. *IEEE Trans. Knowl. Data Eng.*, 26(10):2354–2367, 2014.
[22] W. Luo, H. Tan, L. Chen, and L. M. Ni. Finding time period-based most frequent path in big trajectory data. In *SIGMOD*, pages 713–724, 2013.
[23] R. Mao, P. Zhang, X. Li, X. Liu, and M. Lu. Pivot selection for metric-space indexing. *Int. J. Machine Learning & Cybernetics*, 7(2):311–323, 2016.
[24] M. D. Morse and J. M. Patel. An efficient and accurate method for evaluating time series similarity. In *SIGMOD*, pages 569–580, 2007.
[25] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *VLDB*, pages 802–813, 2003.
[26] N. Pelekis, I. Kopanakis, E. Kotsifakos, E. Frentzos, and Y. Theodoridis. Clustering trajectories of moving objects in an uncertain world. In *ICDM*, pages 417–427, 2009.
[27] M. Qu, H. Zhu, J. Liu, G. Liu, and H. Xiong. A cost-effective recommender system for taxi drivers. In *SIGKDD*, pages 45–54, 2014.
[28] S. Ranu, P. Deepak, A. D. Telang, P. Deshpande, and S. Raghavan. Indexing and matching trajectories under inconsistent sampling rates. In *ICDE*, pages 999–1010, 2015.
[29] M. Rohani, D. Gingras, and D. Gruyer. A novel approach for improved vehicular positioning using cooperative map matching and dynamic base station dgps concept. *IEEE Transactions on Intelligent Transportation Systems*, 17(1):230–239, 2016.
[30] S. Sankararaman, P. K. Agarwal, T. Mølhave, J. Pan, and A. P. Boedihardjo. Model-driven matching and segmentation of trajectories. In *SIGSPATIAL*, pages 234–243, 2013.
[31] H. Su, K. Zheng, K. Zeng, J. Huang, and X. Zhou. Stmaker–a system to make sense of trajectory data. *VLDB*, 7(13):1701–1704, 2014.
[32] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *ICDE*, pages 673–684, 2002.
[33] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh. Experimental comparison of representation methods and distance measures for time series data. *DMKD*, 26(2):275–309, 2013.
[34] B.-K. Yi, H. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *ICDE*, pages 201–208, 1998.
[35] M. Yu, G. Li, D. Deng, and J. Feng. String similarity search and join: a survey. *Frontiers of Computer Science*, 10(3):399–417, 2016.
[36] M. Yu, G. Li, T. Wang, J. Feng, and Z. Gong. Efficient filtering algorithms for location-aware publish/subscribe. *IEEE Trans. Knowl. Data Eng.*, 27(4):950–963, 2015.



Na Ta received her M.S. degree in Computer Science from Tsinghua University in 2007. She is currently a PhD student at the Department of Computer Science in Tsinghua University. Her main research interests include urban computing and spatial databases.

Guoliang Li is currently working as an associate professor in the Department of Computer Science, Tsinghua University, Beijing, China. He received his PhD degree in Computer Science from Tsinghua University, Beijing, China in 2009. His research interests mainly include data cleaning and integration and crowdsourcing.

Yongqing Xie received his B.E. degree in Computer Science from Beihang University in 2014. He is currently a post graduate student at the Department of Computer Science in Tsinghua University. His research interests mainly include urban computing and spatial databases.

Changqi Li is a high school student in senior year, class 11, high school affiliated to Remin university. His main research interests include large-scale spatial data analysis.

Shuang Hao received the bachelor's degree from the Institute of Software Engineering, Shandong University, China. She is currently working toward the PhD degree in the Department of Computer Science, Tsinghua University, Beijing, China. Her research interests include data cleaning and data integration.

Jianhua Feng received his B.S., M.S. and PhD degrees in Computer Science from Tsinghua University. He is currently working as a professor of Department Computer Science in Tsinghua University. His main research interests include large-scale data management and analysis and spatial database.