The Journal of Supercomputing (2019) 75:6129–6157
https://doi.org/10.1007/s11227-019-02813-w

Indexable sub-trajectory matching using multi-segment approximation: a partition-and-stitch framework

Jae-Jun Yoo[1] · Woong-Kee Loh[2] · Kyu-Young Whang[1]

Published online: 15 March 2019

Abstract

With advances in base technologies for moving objects, many studies have been conducted on the construction of databases of the trajectories of moving objects, including the diverse applications related to the trajectories. Most previous studies deal with *whole trajectory matching*, which finds the trajectories T in the database similar to a given query trajectory Q 'as a whole.' However, we often want to find those T containing the sub-trajectories T_{sub} ($\subseteq T$) that are similar to Q. This problem is known as *sub-trajectory matching* and is more complicated than whole trajectory matching since the query trajectory Q can be of any length and the matching sub-trajectories T_{sub} can be at any position in the data trajectories T. In this paper, we present a novel indexing-based sub-trajectory matching algorithm using multi-segment approximation. Our algorithm partitions a data trajectory into multiple component segments and then stores the individual segments in an index. The query trajectory is also partitioned into its component segments, and the search for similar segments for each query segment is efficiently performed using the index. The sub-trajectories similar to the query trajectory are reconstructed by our 'stitching' algorithm using the individual segments retrieved from the index. Our stitching algorithm is novel and innovative in the sense that it facilitates segment-wise partitioning and indexing of data trajectories. Without stitching, only trajectory-wise operations would be affordable, which causes severe storage space overhead and degradation in search performance. Our study is the first that uses indexing in sub-trajectory matching. We define a (multi-segment) trajectory similarity measure that extends a widely used single-segment similarity measure proposed by Lee et al. (in: Proceedings of ACM SIGMOD international conference on management of data (SIGMOD), 2007; in: Proceedings of IEEE international conference on data engineering (ICDE), 2008; Proc VLDB Endow (PVLDB) 1(1):1081–1094, 2008) by using the Hausdorff distance. We perform extensive experiments to compare our method with EDS (Xie, in: Proceedings of ACM SIGMOD international conference on management of data (SIGMOD), 2014), which has been proved to outperform all representative point-based measures in terms of accuracy and performance.

This work was supported by the National Research Foundation of Korea (NRF) grant funded by Korean Government (MSIT) (No. 2016R1A2B4015929).

Extended author information available on the last page of the article

The accuracy of our similarity measure is better than EDS by up to 52.0%, and our algorithm significantly outperforms that using EDS by up to 22,543 times. The performance of our algorithm is linearly scalable in the size of the database, which is an essential property for handling large-scale databases.

Keywords Sub-trajectory matching · Multi-segment approximation · Partition-and-stitch · Segment-wise indexing

1 Introduction

With advances in base technologies for moving objects such as global positioning system (GPS), mobile computing/communications, and mass storage devices, many studies have been performed on the collection and utilization of a variety of information from moving objects. In particular, studies on tracing, analyzing, and storing the location data of moving objects in a database, including the application of such data to location-based services (LBS), are being actively conducted [16, 26, 32]. A sequence of locations of a moving object obtained for a certain time period constitutes a *trajectory*. There are diverse applications of trajectories in transportation optimization, scientific analysis applications, and sports analyses [2, 32]. In our study, the core operation involves finding *similar* trajectories.

Many previous studies represent a trajectory T as a sequence of points, i.e., $T = (p_1, \dots, p_n)$, where $p_1, \cdots, p_n$ are the d-dimensional points sampled at a sequence of time points. We refer to this representation as *point approximation*. There are various similarity measures for point approximation such as *Dynamic Time Warping* (DTW) [29], *Longest Common Sub-Sequence* (LCSS) [25], *Edit Distance on Real sequence* (EDR) [7], *Edit distance with Real Penalty* (ERP) [6], and *DISSIM* [12]. These measures exhibit a drawback such as the possibility of returning different similarity values even for the trajectories obtained from the same route depending on the moving speed, sampling rate, and sampling phase, which incurs inaccurate search results [7, 16, 23, 27, 32].

Several recent studies represent a trajectory T as a sequence of line segments, i.e., $T = (t_1, \dots, t_k)$, where t_i is the line segment connecting two points p_u and p_v $(1 \leq u < v \leq n)$, and use 'all continuous points' on each segment. We refer to this representation as *segment approximation*. Figure 1 illustrates the difference between two approximations. For an actual trajectory T shown by the dashed line, T is represented as $T = (p_1, \dots, p_6)$ and $T = (t_1, \dots, t_5)$ using point and segment approximations, respectively. As we can observe in the figure, segment approximation better represents the actual trajectory and therefore can contribute to improve search

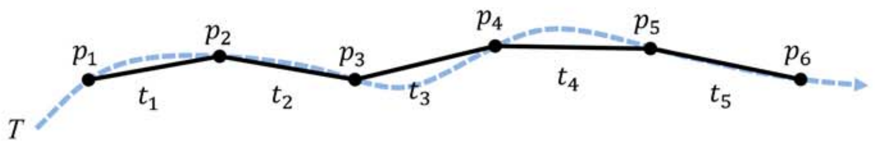


Fig. 1 Representation of a trajectory T using point and segment approximations

accuracy [16, 23, 27]. Lee et al. [16–18] solved the clustering, outlier detection, and classification problems using segment approximation. Ranu et al. [23] proposed a similarity measure called *Edit Distance with Projections* (*EDwP*) using segment approximation and solved the k-nearest neighbor (k-NN) problem based on the measure.

The point-based approach represents a trajectory T as a sequence of geographical points $t_1, \ldots, t_k$ sampled at certain time points from an actual trajectory, i.e., $T = (t_1, \ldots, t_k)$. Any information on the actual trajectory between two adjacent points t_i and t_{i+1} $(1 \le i < k)$ is missing. Segment-based approach forms a line segment L_i between two adjacent points t_i and t_{i+1} to better simulate the actual trajectory. The line segment L_i could be regarded as an interpolation of the actual trajectory between two points t_i and t_{i+1}. For similarity matching, while the point-based approach uses only the sampled points and often incurs inconsistent matching results (see Table 1) [7, 16, 23, 27, 32], the segment-based approach uses not only the sampled points but also all the points on the segments to improve matching accuracy [20, 23]. Even when actual trajectories are sampled at very small intervals, segment-based approach uses more information than point-based approach and therefore demonstrates better accuracy. Such dense sampling also increases storage overhead. Therefore, it is efficient to use interpolation segments between a small number of sampled points.

Most previous studies, including the one by Ranu et al. [23], have dealt with *whole trajectory matching*, which finds the trajectories T in a database similar to a given query trajectory Q 'as a whole.' However, we often prefer finding those T that contain the sub-trajectories T_{sub} $(\subseteq T)$ similar to Q. This problem is called

Table 1 Shortcomings of previous point-based measures

Measure	Speed variation	Sampling rate variation	Phase variation	Additional parameters
DTW		✓	✓	
LCSS		✓	✓	✓
EDR		✓	✓	✓
ERP		✓	✓	✓
DISSIM	✓		✓	

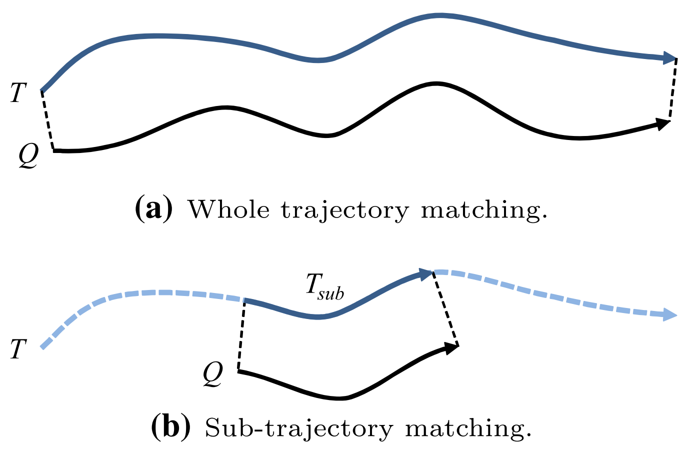
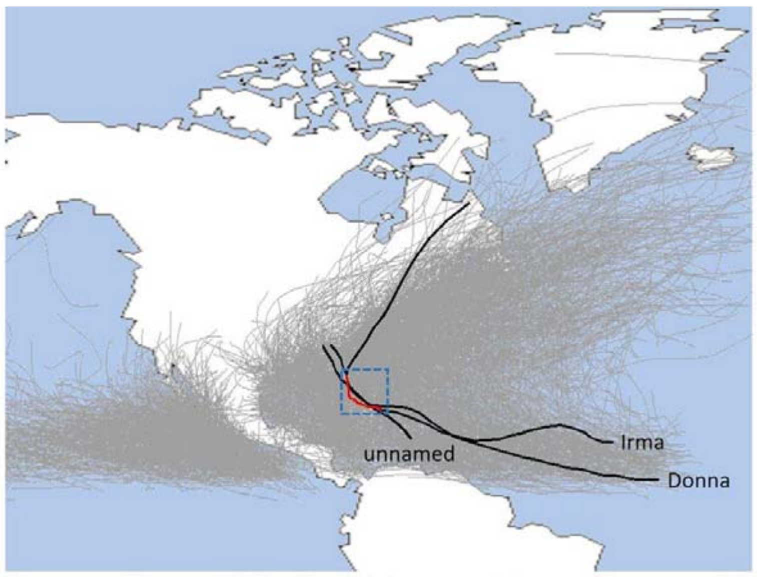
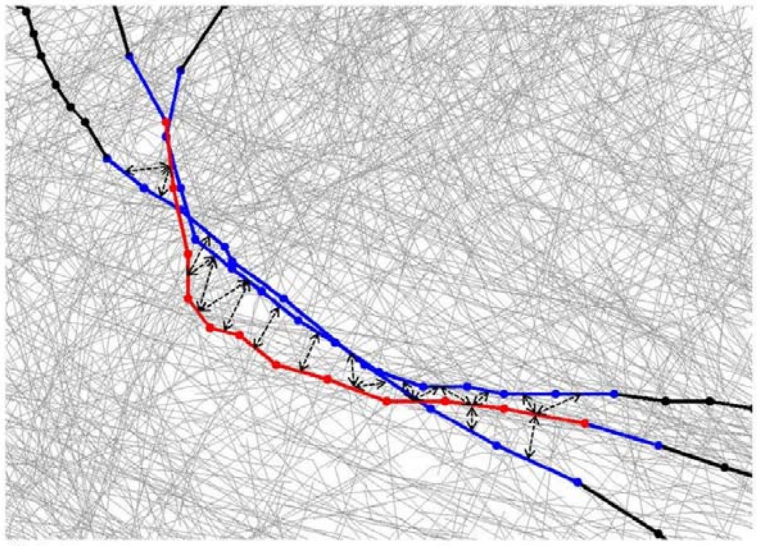


Fig. 2 Comparison of two similar trajectory matching



(a) Query trajectory Q (red) and its matching trajectories (black).



(b) Similar segments in query and matching trajectories.

Fig. 3 Example of sub-trajectory matching (color figure online)

sub-trajectory matching. For instance, consider a hurricane that occurred recently, we may prefer to find past hurricanes with trajectories similar to that of the recent one to predict its future course. In this case, the trajectory of the recent hurricane is 'partially' similar to those of the past and not as a whole. Figure 2 presents the difference between the two matching problems.

Figure 3a demonstrates an example of sub-trajectory matching that finds the hurricanes that traversed through Florida and Cuba. The query trajectory Q, indicated by a red line, was extracted from hurricane 'Irma,' which struck both locations in September 2017. The data and result trajectories were denoted by light gray and black lines, respectively. The result presents the hurricanes Irma (2017), Donna (1960), and an unnamed one (1899), which have passed through Florida and Cuba.

Figure 3b presents a detailed query and the data trajectories consisting of multiple segments obtained by enlarging the small dashed box in Fig. 3a. The dashed arrows indicate pairs of 'similar' segments[1] whose distances are not larger than a pre-specified threshold ϵ, which is set as the horizontal width of Florida. Data segments matching with query segments are shown in blue. Note that, in the method that we proposed in Sect. 4, a query segment can be similar to one or more data segments, and vice versa.

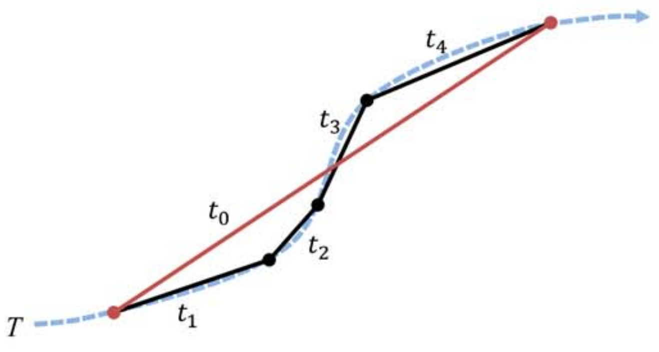
Since query trajectories can assume any length and the matching sub-trajectories can be at any position in data trajectories, sub-trajectory matching is more complicated than whole trajectory matching, and the latter is considered a simple form of the former. In whole trajectory matching, since the query trajectory Q is compared with a whole trajectory T, the distance between Q and T is computed only once for each data trajectory T. On the contrary, in sub-trajectory matching, since Q is compared with all possible sub-trajectories T_{sub} in T, the number of distance computations is $O(n^2)$ for each T, where n is the average length of data trajectories [27]. In addition, there have been very few studies that address the matching problem. Xie [27] proposed a segment-based similarity measure called *Edit Distance on Segment* (*EDS*) for sub-trajectory matching. Although EDS has demonstrated better performance than the previous point-based measures in sub-trajectory matching, the algorithm using EDS is hardly indexable by nature and therefore cannot achieve improvements in performance. We present further discussion on EDS in Sect. 2.

In this paper, we present an indexing-based sub-trajectory matching algorithm using multi-segment approximation. Our algorithm partitions the multi-segment data trajectories into their component segments and then independently stores each individual segment in an index—even those from the same trajectory. The query trajectory is also partitioned into its component segments, and the search for similar segments for each query segment is efficiently performed using the index. The data sub-trajectories similar to the query trajectory are reconstructed by our 'stitching' algorithm using the individual segments retrieved from the index.

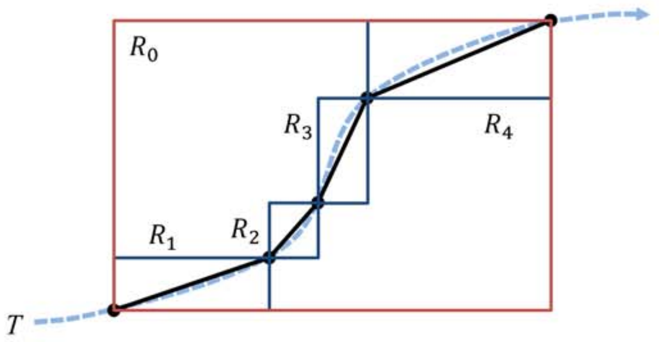
This partition-and-stitch framework for multi-segment sub-trajectory matching has never been explored in the literature. In this study, we define a multi-segment trajectory similarity measure that extends a single-segment similarity measure by using the Hausdorff distance. The Hausdorff distance facilitates segment-wise processing of trajectories; it enables segment-wise partitioning, indexing, and stitching—leading to significant improvements in search accuracy and performance. We adopt the widely used measure by Lee et al. [16–18] as a segment similarity measure. Although Lee et al. represented a trajectory as a sequence of multiple segments, they defined the measure only for individual segments (rather than trajectories), and therefore the target of their algorithms comprises individual segments; e.g., TRACLUS [16] forms clusters of similar segments. We extend this measure to handle multi-segment trajectories.

Our stitching algorithm is novel and innovative in the sense that (1) it facilitates segment-wise partitioning and indexing of data trajectories (see Sect. 4), and

[1] Some arrows are omitted to avoid clutter.



(a) Representation of a trajectory using single-segment and multi-segment approximations.



(b) Representation of a trajectory with MBRs of the whole trajectory and of its component segments.

Fig. 4 Advantages of using multi-segment approximation (color figure online)

(2) it is composed of only a few simple bitmap operations and therefore incurs very marginal overhead in finding all similar sub-trajectories (see Sect. 5). Without our stitching algorithm, since our trajectory similarity measure (as well as all other previous measures) is complicated and does not satisfy triangular inequality, it can be extremely challenging to design trajectory-wise efficient indexing and searching algorithms. Even when such algorithms are feasible, we must build an index that contains all possible sub-trajectories extracted from all trajectories in the database. The complexity of the index is $O(Dn^2)$, where D is the number of data trajectories and n is the average number of segments in a data trajectory. This evidently results in severe storage space overhead and search performance degradation. Our stitching algorithm enables reconstruction of similar sub-trajectories from similar segments and therefore makes simple and efficient segment-wise indexing and searching feasible.

Figure 4 demonstrates the advantages of using multi-segment approximation. Figure 4a compares the representations of a trajectory T by a single segment t_0 (red) and a sequence of multiple segments $(t_1, \ldots, t_4)$. The multi-segment representation is closer to the actual trajectory and therefore improves search accuracy. Figure 4b compares the representations of a trajectory T with minimum bounding rectangles (MBRs), where MBRs R_0 (red) and $R_1, \ldots, R_4$ (blue) are the smallest rectangles containing segments t_0 and $t_1, \ldots, t_4$, respectively. We can observe that R_0 takes up additional unnecessary space than the space covered by $R_1, \ldots, R_4$.

Therefore, R_0 incurs more false positives and degrades search performance. The indexing element, called *vantage descriptor* for k-NN search using EDwP [23], is defined for a whole data trajectory rather than its component segments, resulting in search inefficiency. In our algorithm, we can set a limit λ such that any segment longer than λ must be divided into several segments shorter than λ, reducing the size of MBRs and false positives. This is made possible through our trajectory similarity measure and our stitching algorithm that facilitates many-to-many (i.e., not necessarily one-to-one) segment correspondences between a data sub-trajectory and the query trajectory (see Sect. 3).

The contributions of this paper are summarized as follows:

- We present a partition-and-stitch framework for sub-trajectory matching. In this framework, we partition the trajectory into individual segments and employ indexing to facilitate fast search for matching segments. To the extent of our knowledge, the 'indexing-based' approach to 'sub-trajectory matching' is novel. We reconstruct a matching sub-trajectory from individual matching segments using our innovative 'stitching' algorithm based on simple and fast matrix operations. We formally prove that our stitching algorithm is sound and complete, i.e., it produces neither false positives nor false negatives.
- We define a new similarity measure between two multi-segment trajectories using the Hausdorff distance [21], which is widely used in the field of pattern recognition [5]. The Hausdorff distance effectively handles the multi-segment nature of trajectories by decomposing the distance between trajectories into those between individual segments (not necessarily one-to-one correspondence) in the trajectories. We employ the similarity measure by Lee et al. [16–18] for finding similar (individual) segments in our algorithm.
- We perform extensive experiments using real and synthetic datasets to evaluate our similarity measure and sub-trajectory matching algorithm. We compare our similarity measure with EDS, which has demonstrated to be superior over all representative point-based measures in terms of accuracy and performance [27]. Evidently, the accuracy of our similarity measure is better than that of EDS by up to 52.0%, and our algorithm significantly outperforms that using EDS by up to 22,543 times. The performance of our algorithm is linearly scalable in the size of the database, which is an essential property for handling large-scale databases.

This paper is organized as follows: Sect. 2 discusses existing trajectory similarity measures and matching algorithms. Section 3 formally defines the sub-trajectory matching problem based on multi-segment approximation, and Sect. 4 presents our indexing-based algorithm for solving the matching problem in detail. Section 5 evaluates our algorithm through a series of experiments. Finally, Sect. 6 concludes this paper.

2 Related work

In this section, we discuss previous similar trajectory measures and matching algorithms. Many previous studies presented various similarity measures using point approximation including DTW [29], LCSS [25], EDR [7], ERP [6], and DISSIM [12]. These measures have a drawback that there exists a possibility of returning different similarity values even for trajectories obtained from the same route depending on moving speed, sampling rate, and sampling phase [7, 16, 23, 27, 32]. Some measures require additional threshold parameters [23], which may also cause discrepancies in similarity values. Table 1 summarizes the factors that may cause discrepancies for each similarity measure [23]. For example, DTW may return different similarity values for trajectories with different sampling rates and phases. Pelekis et al. [22] proposed a temporal-constrained sub-trajectory clustering algorithm using an index structure called *Representative Trajectory Tree* (*ReTraTree*), which supports incremental grouping of similar sub-trajectories. The trajectory clustering algorithms that use point approximation are summarized in [30].

Segment approximation can overcome the shortcomings of point approximation; in addition to sampled points, segment approximation uses 'all continuous points' on the line segments to improve accuracy. Lee et al. [16–18] defined a similarity measure dist(t_i, t_j) between two line segments t_i and t_j. Our similarity measure is an extension of dist() combined with the Hausdorff distance and is applicable to multi-segment (sub-)trajectories. Our similarity measure overcomes all the shortcomings in Table 1 since it uses all continuous points as well as end points on the line segments and does not require any additional parameter for similarity computation. Ranu et al. [23] presented a segment-based measure EDwP and demonstrated through experiments that it is robust against all factors in Table 1 and exhibits higher accuracy than the previous point-based measures. Mao et al. [20] proposed a similarity measure called *Segment-based Dynamic Time Warping* (*SDTW*), which adopts point-segment distance to reduce discrepancies caused by the factors in Table 1 for whole trajectory matching, and demonstrated that its accuracy was improved by about 86% over EDR. Xie et al. [28] proposed a distributed framework for whole trajectory matching over a large trajectory database. In the framework, the segments extracted from all data trajectories are partitioned according to their spatial locality, and each partition is stored in a local multidimensional index. The global result of similar whole trajectory search is constructed by merging local search results.

Many similar trajectory matching algorithms have been proposed that use the measures mentioned above. The k-NN algorithms using EDR, DISSIM, EDwP, and LCSS have been proposed in [7, 12, 23, 25]. Since the measures do not satisfy triangular inequality, these algorithms presented new indexing or pruning methods. Chen et al. [7] presented three pruning methods using mean value Q-grams, near triangle inequality, and trajectory histogram to accelerate the computation of EDR whose complexity is $O(mn)$, where m and n are the lengths of the two trajectories. Ranu et al. [23] presented a new index structure called *TrajTree*, which is analogous to the R-tree, and illustrated that their indexing-based algorithm using EDwP is superior to the ones using LCSS, EDR, and DISSIM in terms of accuracy and performance.

All these algorithms deal with whole trajectory matching. A straightforward method of solving the sub-trajectory matching problem using these algorithms involves the extraction and comparison of all possible sub-trajectories from all data trajectories, and the complexity is $O(Dn^2)$, where D is the number of data trajectories and n is their average length (the number of comprising points/segments). Even though we adopt an indexing method such as the TrajTree [23], we must build an index for every possible length of the query trajectory, which causes severe storage and management overhead. In this paper, we present an algorithm that can handle query trajectories of any length with only one multidimensional index.

Lee et al. [16–18] proposed three algorithms using the dist() measure: a clustering algorithm *TRACLUS* [16], which is a variation of a density-based clustering algorithm DBSCAN [11], an outlier detection algorithm *TRAOD* [17], and a classification algorithm *TraClass* using region-based and trajectory-based clustering [18]. While these algorithms handle only segments (e.g., TRACLUS forms clusters of similar segments), our algorithm handles multi-segment (sub-)trajectories.

Alamri et al. [1] proposed a taxonomy of moving object queries in five perspectives, namely *location*, *motion*, *object*, *temporal*, and *patterns* perspectives, to address many features of moving objects. Trajectory queries are included in the temporal perspective, which concerns the temporal aspects and characteristics of moving objects. Alamri et al. also introduced a few data structures for trajectory queries such as TB-tree (Trajectory Bundle tree) and STR-tree (Spatio-Temporal R-tree).

Buchin et al. [4] proposed an algorithm that finds a pair of similar sub-trajectories from two given trajectories. Given a large-scale trajectory database $\mathcal{D}$ and a query trajectory $\mathcal{Q}$, the algorithm must sequentially compare $\mathcal{Q}$ with all data trajectories in $\mathcal{D}$, which results in severe performance overhead. Hung et al. [13] proposed a trajectory pattern mining framework that fills in ‘silent durations,’ i.e., time durations with no data points, by clustering and merging the trajectories consisting of different data points sampled from similar routes. Xie [27] presented a segment-based similarity measure EDS for sub-trajectory matching. EDS between two (sub-)trajectories is defined as the minimum cost of a series of segment-wise transformation. The transformation between two segments is defined as changing (i.e., displacing, stretching, and rotating) from one segment to another. Since the transformation is different for every pair of trajectories, the matching algorithm using EDS is hardly indexable by nature and cannot achieve high-performance improvement. We demonstrate that our matching algorithm dramatically outperforms that using EDS (see Sect. 5).

Ding et al. [8] proposed an Apache Spark-based distributed platform named *UlTraMan* for managing and analyzing a large dataset of trajectories in a unified manner. While conventional trajectory collection, storage, transformation, and querying were carried out in a series of respective non-compatible systems, overall trajectory processing and analysis in UlTraMan are pipelined without unnecessary data copying and serialization. Shang et al. [24] proposed a distributed in-memory trajectory analysis system named *DITA* by extending Apache Spark SQL. Although they claimed that DITA supports most of trajectory similarity measures, most of algorithm descriptions and experimental evaluations in [24] were made using DTW, which may return inconsistent similarity results as shown in Table 1. Both UlTraMan and DITA dealt with only whole trajectory matching without the notion of segments.

Table 2 Summary of notation

Notation	Description
T, Q	Data and query trajectories
T_{sub}	A sub-trajectory in T
t_i, q_i	Segments in T and Q
$\|T\|$, $\|Q\|$	The number of segments in T and Q

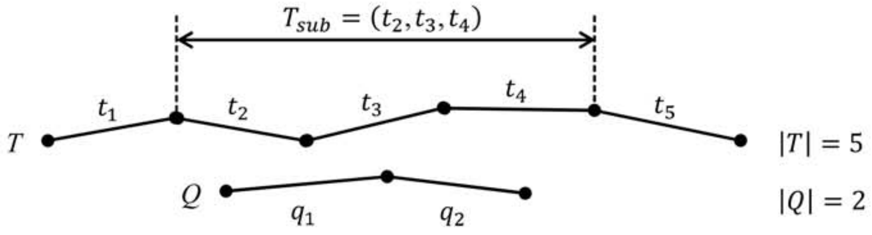


Fig. 5 Example of using notations

There have been a few approaches for privacy preservation in trajectory databases. Dong and Pi [9] proposed an approach named *TOPF* promoting not only privacy but also usability of published trajectory databases; TOPF removes infrequent paths and forms trajectory groups satisfying *k*-anonymity based on frequent paths. Kaplan et al. [15] showed that 'secure' trajectory query services are not robust against known-sample attacks; given a set *K* of known trajectories *T*, by querying the distance of target trajectory *X* or *k*-nearest trajectories from *T*, one can infer privacy information such as specific locations and timestamps about *X*. Mao et al. [19] proposed a scalable de-centralized outlier detection approach over distributed trajectory streams. Their definition on outlier is based on 'feature-grouping' which considers various features such as relative speed rather than spatial proximity.

3 Problem definition

In this section, we formally define sub-trajectory matching problem. Table 2 summarizes the notation used in this paper, and Fig. 5 presents an example of the notations in Table 2 used for two trajectories T and Q. In the figure, the data trajectory T and the query trajectory Q consist of $|T| = 5$ and $|Q| = 2$ segments and are represented as $T = (t_1, \dots, t_5)$ and $Q = (q_1, q_2)$, respectively. The sub-trajectory T_{sub} composed of three segments in T is represented as $T_{\text{sub}} = (t_2, t_3, t_4)$.

Note that we do not impose any restriction on segment partitioning. In other words, regardless of whether a segment is made only between two adjacent points in the trajectory (as in this paper) or it is made to represent two or more adjacent points as in [16–18], our similarity measure and sub-trajectory matching algorithm are still valid. Our similarity measure between two (sub-)trajectories is defined using the Hausdorff distance [5] based on the distance $d_{\text{seg}}()$ between segments as follows:

Definition 1 For any two trajectories $T = (t_1, t_2, \ldots, t_n)$ and $Q = (q_1, q_2, \ldots, q_m)$, their Hausdorff distance is defined as:

$$D_H(T, Q) = \max_j \left\{ \min_i \left\{ d_{\text{seg}}(t_i, q_j) | t_i \in T \right\} | q_j \in Q \right\}, \tag{1}$$

where $d_{\text{seg}}(t_i, q_j)$ is the distance between two segments t_i $(i = 1..n)$ and q_j $(j = 1..m)$ (explained later in this section). □

Note that two trajectories T and Q may have different lengths in Definition 1 (i.e., $n \neq m$). In general, it does not always hold that $D_H(T, Q) = D_H(Q, T)$ for arbitrary T and Q [14]. Therefore, we define the distance $D()$ between T and Q as follows:

Definition 2 The distance between two trajectories T and Q is defined as:

$$D(T, Q) = \max \left\{ D_H(T, Q), D_H(Q, T) \right\}, \tag{2}$$

where $D_H()$ is defined in Eq. (1). □

Before defining the distance $d_{\text{seg}}()$ between two segments, we define *base distance* $d_{\text{seg},0}()$ as follows:

Definition 3 For any two segments s_1 and s_2, the *base distance* $d_{\text{seg},0}(s_1, s_2)$ is defined as the shortest distance between any two points contained in each segment as follows:

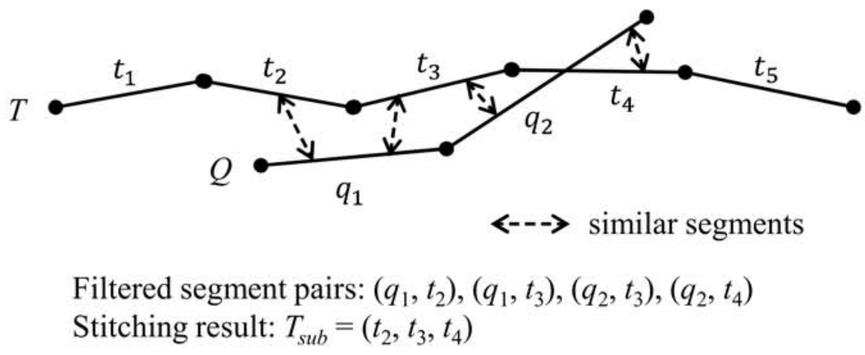
$$d_{\text{seg},0}(s_1, s_2) = \min \left\{ d(p_1, p_2) | p_1 \in s_1, p_2 \in s_2 \right\}, \tag{3}$$

where p_1 and p_2 are the points contained in s_1 and s_2, respectively, and $d()$ is the Euclidean distance between the points. □

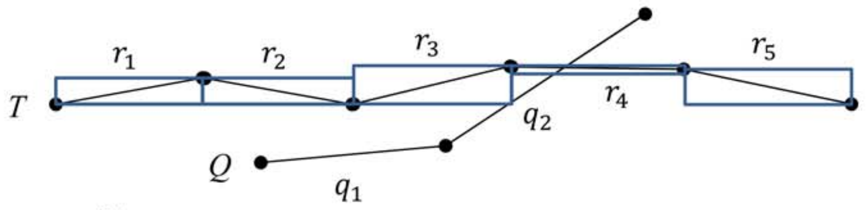
We demonstrate in the next section that our sub-trajectory matching algorithm that uses the base distance $d_{\text{seg},0}()$ is sound and complete. We also illustrate that for any distance $d_{\text{seg}}()$ that satisfies $d_{\text{seg}}(s_i, s_j) \geq d_{\text{seg},0}(s_i, s_j)$ for any two segments s_i and s_j, our algorithm is sound and complete (Lemma 1 and Corollary 1). In this paper, we use $d_{\text{seg}}(s_i, s_j) = \text{dist}(s_i, s_j)$ as the distance between two segments s_i and s_j, where dist() is defined by Lee et al. [16–18] and is a widely used measure for the distance between two segments. We can easily illustrate that it always holds that $d_{\text{seg}}(s_i, s_j) \geq d_{\text{seg},0}(s_i, s_j)$ (see "Appendix").

In this paper, two segments s_1 and s_2 whose distance $d_{\text{seg}}(s_1, s_2)$ is less than or equal to the search range ϵ are called *similar segments*. The sub-trajectory matching problem using the similarity measure mentioned above is formally defined as follows:

Definition 4 (*Sub-trajectory matching*) Given a query trajectory Q and a search range ϵ, all sub-trajectories $T_{i,\text{sub}}$ that satisfy the following equation in the


t_1
t_2
t_3
t_4
t_5
T
Q
q_1
q_2
similar segments
Filtered segment pairs: (q_1, t_2), (q_1, t_3), (q_2, t_3), (q_2, t_4)
Stitching result: $T_{sub} = (t_2, t_3, t_4)$

(a) Matching between data and query segments.



(b) Indexing of data segments in form of MBRs.

Fig. 6 Example of sub-trajectory matching

trajectories T_i $(1 \leq i \leq |\mathcal{D}|)$ in the database $\mathcal{D}$ are returned as the result of sub-trajectory matching:

$$\left\{ T_{i,\text{sub}} \middle| D\left(Q, T_{i,\text{sub}}\right) \leq \epsilon, T_{i,\text{sub}} \subseteq T_i \right\}, \tag{4}$$

where $D()$ is given in Definition 2. □

$T_{i,\text{sub}}$ is called the *similar sub-trajectory* of Q. Note that two trajectories Q and $T_{i,\text{sub}}$ can have different lengths (i.e., numbers of segments) according to Definition 1.

Figure 6a presents an example of sub-trajectory matching; the sub-trajectory similar to the given query trajectory Q is $T_{i,\text{sub}} = (t_2, t_3, t_4)$. In the figure, the bidirectional dashed arrows indicate the matching segments similar to each other. We can observe that $D_H(Q, T_{i,\text{sub}}) \leq \epsilon$ and $D_H(T_{i,\text{sub}}, Q) \leq \epsilon$ according to Eq. (1) and thus $D(Q, T_{i,\text{sub}}) \leq \epsilon$ according to Eq. (2). Note that a segment in one trajectory can match with two or more segments in the other trajectory. Figure 6b presents MBRs for data segments in Fig. 6a; MBRs $r_1, \ldots, r_5$ contains $t_1, \ldots, t_5$, respectively. The data segments are handled in form of MBRs; MBRs are stored in the index and returned as a search result. We discuss the details in the next section.

Our sub-trajectory matching algorithm returns only maximal sub-trajectories as the search result. In Fig. 6, for example, $T'_{i,\text{sub}} = (t_2, t_3)$ is also a similar sub-trajectory. However, $T_{i,\text{sub}}$ contains $T'_{i,\text{sub}}$ (i.e., $T'_{i,\text{sub}} \subset T_{i,\text{sub}}$), and therefore, only $T_{i,\text{sub}}$ is returned as a result of sub-trajectory matching against Q. The maximal sub-trajectory is defined as follows:

Definition 5 (*Maximal sub-trajectory*) For a query trajectory Q, if there exists no similar sub-trajectory T^*_{sub} that properly contains a similar trajectory T_{sub} (i.e., $T_{\text{sub}} \subset T^*_{\text{sub}}$), T_{sub} is defined as a *maximal sub-trajectory*. □

Although Hausdorff distance is incorporated in our algorithm, any similarity measure $\hat{D}(S, T)$ between two (sub-)trajectories S and T can be applicable in our partition-and-stitch framework if there are functions F and $\hat{d}$ such that $\hat{D}(S, T) = F_{i,j}\{\hat{d}(s_i, t_j)\}$ and $\hat{d}(s_i, t_j) \geq d_{\text{seg},0}(s_i, t_j)$, where s_i and t_j are the segments comprising S and T, respectively, and $\hat{d}(s_i, t_j)$ is the distance between s_i and t_j.

Most of moving object applications require obtaining the same similarity value of the trajectories with different speed, sampling rates, and sampling phases. For example, the same sign languages should be accurately recognized as having the same meaning in spite of different speed, sampling rates, and sampling phases. We show in Sect. 5 that our similarity measure based on Hausdorff distance represents the similarity between two (sub-)trajectories better than the previous measures including DTW, LCSS, EDR, ERP, and EDS by comparing their accuracies using various real and synthetic datasets.

4 MaTIS: an indexing-based multi-segment sub-trajectory matching algorithm

In this section, we propose an efficient algorithm for solving the sub-trajectory matching problem defined in the previous section. Our algorithm is an indexing-based one that improves performance by dramatically reducing the number of candidate data trajectories to be compared with the query trajectory using a multidimensional index. We call our algorithm as *MaTIS* (***M** ulti-segment sub-**T**rajectory matching with **I**ndexing and **S**titching*). MaTIS is composed of the indexing and searching phases. The latter, in turn, consists of three sub-phases: index searching, filtering, and stitching phases.

- *Indexing phase* All segments t_i in all data trajectories T in the database $\mathcal{D}$ are stored in a multidimensional index such as the R*-tree (see Sect. 4.1).
- *Index searching phase* For each segment q_j in the query trajectory Q, all candidate segments t_i that are possibly within the ϵ distance from q_j are retrieved using the index. We prove that there is no false drop (false negative) in the set of candidate segments (see Sect. 4.2).
- *Filtering phase* For each candidate segment t_i obtained in the index searching phase, the actual distance $d_{\text{seg}}(t_i, q_j)$ to the corresponding query segment q_j is computed, and those within ϵ distance are returned (see Sect. 4.3).
- *Stitching phase* The sub-trajectories similar to Q are obtained by stitching contiguous segments in the same data trajectory returned during the filtering phase. We formally prove that our stitching algorithm is sound and complete (see Sect. 4.4).

We use simple and conventional index construction and range search algorithms for the indexing and index searching phases. As explained in Sect. 1, the use of

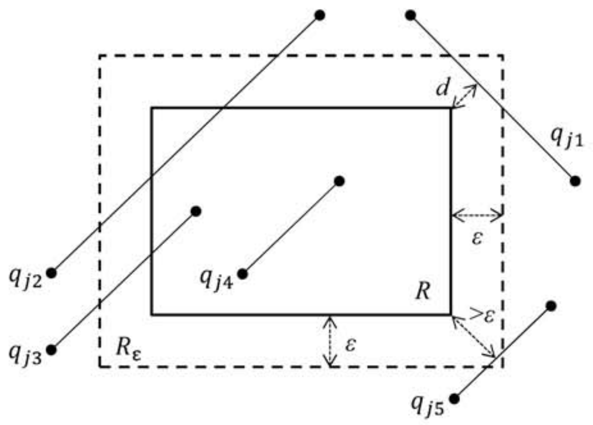


Fig. 7 Spatial relationships between a query segment and an MBR

segment-wise algorithms, which are much more efficient than (sub-)trajectory-wise algorithms, is made possible through our stitching algorithm. The stitching algorithm itself also runs fast since it is composed of a few simple bitmap operations, as seen in Sect. 5. Figure 6 demonstrates an example of filtering and stitching phases. As a result of filtering, data segments t_2 and t_3 are found to be similar to query segment q_1, and data segments t_3 and t_4 are found to be similar to query segment q_2 (many-to-many correspondence among segments). By stitching these data segments, we obtain $T_{\text{sub}} = (t_2, t_3, t_4)$ as a maximal sub-trajectory.

4.1 Indexing phase

In the indexing phase, the data segments t_i extracted from all data trajectories T are stored in a multidimensional index. Each data segment is stored in the R*-tree [3] in the form of a minimum bounding rectangle (MBR) that contains the data segment. In addition to the segment MBR, the ID of data trajectory T and the position of t_i in T are stored together in the index.

4.2 Index searching phase

In the index searching phase, all data segment MBRs within the given ϵ distance from each segment q_j are retrieved using the index built in the previous phase. The distance between a segment and an MBR is formally defined in Definition 6. In Fig. 7, for example, if the distance d from the query segment q_{j1} to the MBR R is less than or equal to ϵ, R is returned. Since the internal and terminal nodes represent the data region in the form of an MBR in the R*-tree, the distance computation between the query segment and an MBR is recursively performed along the search path.

Definition 6 The distance between a segment s and an MBR R is defined as the shortest distance between any two points contained in s and R as follows:

$$d(s,R) = \min\left\{d(p_1,p_2) | p_1 \in s, p_2 \in R\right\}, \tag{5}$$

where p_1 and p_2 are arbitrary points contained in s and R, respectively, and $d()$ is the Euclidean distance. □

There are four spatial relationships between a query segment q_j and an MBR R as shown in Fig. 7: (1) q_j and R never overlap (q_{j1}), (2) q_j and R overlap, but both end points of q_j reside outside of R (q_{j2}), (3) q_j and R overlap, and one end point of q_j resides inside of R (q_{j3}), and finally (4) q_j is contained in R (q_{j4}). In the case of q_{j3} and q_{j4}, where one or more end points of q_j reside inside of R, the distance is $0\,(\leq \epsilon)$. In the case of q_{j1} and q_{j2}, however, the distances from both end points to R might be greater than ϵ. Therefore, for q_{j1}, the shortest distance d should be computed from all vertices of R, and for q_{j2}, we must check whether it intersects with any side of R. For a query segment q_j whose spatial relationship with R is unknown, we must perform both operations, i.e., computing the shortest distance from all vertices and checking the intersection with any side.

To check whether the distance between a query segment q_j and an MBR R is less than ϵ, we use a fast method that allows a small number of false alarms as follows. First, a new MBR R_ϵ (designated by the dashed rectangle in Fig. 7) is computed by extending all sides in R by ϵ along all axes. We then check whether any end point of the query segment is contained in R_ϵ as q_{j3} and q_{j4}. Since the sides of R_ϵ are parallel to the coordinate axes, it can be checked very quickly with a few inequality comparisons. If any end point is contained in R_ϵ, R is returned as a similar MBR to q_j. Next, we check whether q_j intersects with any side of R as q_{j1} and q_{j2}. Since the sides of R_ϵ are parallel to the coordinate axes, this can also be done very quickly with a few equation computations [10]. If q_j intersects with any side of R_ϵ, R is regarded to be closer than ϵ from q_j and returned as an MBR similar to q_j. However, there are segments such as q_{j5} in Fig. 7 that intersect with R_ϵ, but their distance d from R is larger than ϵ. These false alarms are removed in the filtering phase.

Algorithm 1 summarizes this procedure, and Lemma 1 illustrates that there are no falsely dropped MBRs (i.e., data segments) by Algorithm 1 for the base distance $d_{\text{seg},0}()$ defined in Eq. (3) since the set of data segments returned by Algorithm 1 is a superset of those that could be obtained using Eq. (3). Corollary 1 demonstrates that we can use Algorithm 1 for any distance $d_{\text{seg}}()$ between segments that satisfy $d_{\text{seg}}() \geq d_{\text{seg},0}()$. In the previous section, we mentioned that MaTIS uses $d_{\text{seg}}() = \text{dist}()$ by Lee et al. [16–18] and proved that $d_{\text{seg}}() \geq d_{\text{seg},0}()$ for any two segments (see "Appendix").

Algorithm 1 Determination of similarity between a query segment and an MBR.

```
Input: Query segment $q_j$, data MBR $R$, and search range $\epsilon$
Output: true if $d(q_j, R) \leq \epsilon$; false otherwise
1: Construct $R_\epsilon$ by extending each side of $R$ by $\epsilon$ along every axis;
2: for each end point $p$ of $q_j$ do
3:     if $p$ resides within $R_\epsilon$ then return true;
4: end for
5: for each side $S$ of $R_\epsilon$ do
6:     if $q_j$ intersects $S$ then return true;
7: end for
8: Return false;
```

Lemma 1 *Algorithm 1 produces no false drop for the base distance $d_{\mathrm{seg},0}()$ between segments in Eq.* (3).

Proof For a segment q and an MBR R containing a segment $t\,(\neq q)$, it always holds that $d_{\mathrm{seg},0}(q,t) \geq d(q,R)$ and therefore $d_{\mathrm{seg},0}(q,t) \leq \epsilon \implies d(q,R) \leq \epsilon$ for any search range ϵ. Thus, it also holds that:

$$\left\{t | d_{\mathrm{seg},0}(q,t) \leq \epsilon\right\} \subseteq \{t | t \in R, d(q,R) \leq \epsilon\}\,. \tag{6}$$

Therefore, there is no false drop by Algorithm 1 for the base distance. □

Corollary 1 *Algorithm 1 produces no false drop for any distance $d_{\mathrm{seg}}()$ between segments that satisfy $d_{\mathrm{seg}}() \geq d_{\mathrm{seg},0}()$.*

Proof For any two segments q and t, since it holds that $d_{\mathrm{seg}}(q,t) \geq d_{\mathrm{seg},0}(q,t)$, it also holds that $d_{\mathrm{seg}}(q,t) \geq d(q,R)$ for an MBR R containing t. Therefore, it holds that:

$$\left\{t | d_{\mathrm{seg}}(q,t) \leq \epsilon\right\} \subseteq \{t | t \in R, d(q,R) \leq \epsilon\}\,. \tag{7}$$

Therefore, there is no false drop by Algorithm 1 for $d_{\mathrm{seg}}()$. □

4.3 Filtering phase

For each MBR R obtained in the index searching phase, we compute the distance $d_{\mathrm{seg}}()$ between the actual data segment in R and the corresponding query segment. For each query segment q_j and the data segments t_i within actual distance ϵ from q_j, all (t_i, q_j) pairs are passed to the next phase.

4.4 Stitching phase

In the stitching phase, we find the maximal sub-trajectories T_{sub} that are similar to the query trajectory Q using the (t_i, q_j) pairs obtained from the filtering phase. Since the data segments that form a final similar sub-trajectory cannot be derived from different data trajectories, the first task in this phase is to group (t_i, q_j) pairs from the same data trajectory. This task can be easily performed by hashing or sorting the ID of trajectories T. The (t_i, q_j) pairs from the same data trajectory are collected in the same hash bucket using the hash function on the trajectory ID.

To find the maximal sub-trajectories, we need the following definitions:

Definition 7 (*Adjacency segment matrix*) For a data trajectory $T = (t_1, \ldots, t_n)$ and a query trajectory $Q = (q_1, \ldots, q_m)$, the adjacency segment matrix M is an adjacency matrix of size $m \times n$, whose cell values are defined as follows:

Fig. 8 Adjacency segment matrix for the matching segment pairs in Fig. 6

Adjacency segment matrix M

M	t_1	t_2	t_3	t_4	t_5
q_1	0	1	1	0	0
q_2	0	0	1	1	0

Vertical projection of M

$P_v(M)$	0	1	1	1	0

$$M(j,i) = \begin{cases} 1 & \text{if } d_{\text{seg}}(q_j, t_i) \le \epsilon \\ 0 & \text{otherwise} \end{cases}. \tag{8}$$

□

Figure 8 presents the adjacency segment matrix M for T and Q shown in Fig. 6, where the matching segment pairs are $(q_1, t_2), (q_1, t_3), (q_2, t_3)$, and (q_2, t_4). The cell values corresponding to the pairs are all 1; 0 for the remaining cells.

Definition 8 (*Vertical projection*) For an adjacency segment matrix M of size $m \times n$, the vertical projection $P_{\text{v}}(M)$ is a vector of size n consisting of only 0's and 1's, whose element values are defined as follows:

$$P_{\text{v}}(M)[i] = M(1,i)\,\text{OR}\ \ldots\ \text{OR}\,M(m,i)\,. \tag{9}$$

□

Definition 9 (*Horizontal projection*) For an adjacency segment matrix M of size $m \times n$, the horizontal projection $P_{\text{h}}(M)$ is a vector of size m consisting of only 0's and 1's, whose element values are defined as follows:

$$P_{\text{h}}(M)[j] = M(j,1)\,\text{OR}\ \ldots\ \text{OR}\,M(j,n)\,. \tag{10}$$

□

Algorithm 2, referred to as *ProjStitch* (*Stitching by VH-projection*), summarizes the procedure of finding similar sub-trajectories using these definitions. ProjStitch is performed for each data trajectory T. First, we generate the adjacency segment matrix M for (t_i, q_j) pairs collected for a data trajectory T (as defined in Definition 7) and then the vertical projection $P_{\text{v}}(M)$ for the matrix M (as defined in Definition 8). We then extract an adjacency segment sub-matrix N for each sequence of contiguous 1's divided by one or more 0's in $P_{\text{v}}(M)$. Let s and f be the start and final positions of the sequence of 1's in $P_{\text{v}}(M)$. Then, N is an $m \times k$ $(k = f - s + 1)$ matrix composed of the sth through fth columns of M. For example, the start and end positions of the sequence of contiguous 1's obtained from $P_{\text{v}}(M)$ in Fig. 8 are s = 2 and f = 4, and the corresponding adjacency segment sub-matrix N consists of the 2nd, 3rd, and

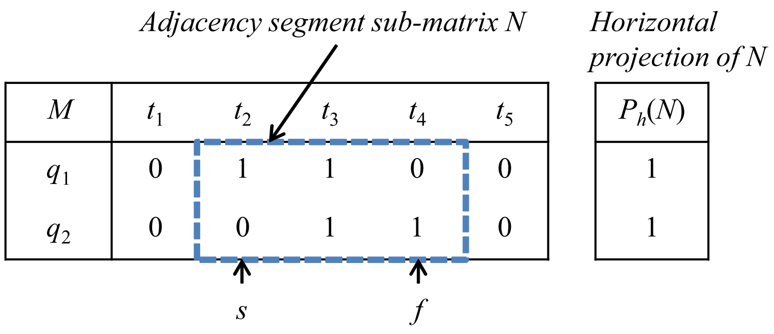


Fig. 9 Adjacency segment sub-matrix N obtained from the adjacency segment matrix M in Fig. 8

4th columns of M as shown in Fig. 9 (designated by a dashed box). For each sub-matrix N, we compute the horizontal projection $P_{\mathrm{h}}(N)$. If all elements in the vector $P_{\mathrm{h}}(N)$ are 1, ProjStitch returns the corresponding sub-trajectory $T_{\mathrm{sub}} = (t_s, \dots, t_f)$ ($T_{\mathrm{sub}} = (t_2, t_3, t_4)$ in Fig. 9) as a similar sub-trajectory.

Lemmas 2–4 demonstrate that ProjStitch is sound and complete. Lemma 2 demonstrates that ProjStitch returns only sub-trajectories similar to the query trajectory without false positives (soundness). Lemmas 3 and 4 illustrate that all similar sub-trajectories obtained by ProjStitch are maximal sub-trajectories and that there is no falsely dropped maximal sub-trajectory by ProjStitch, respectively (completeness).

Algorithm 2 ProjStitch.

```
Input: Set of (data segment $t_i$, query segment $q_j$) pairs for a data trajectory $T$
Output: Set $\mathcal{R}$ of sub-trajectories $T_{sub}$ similar to $Q$
 1: $\mathcal{R} \leftarrow \varnothing$;
 2: Construct an adjacency segment matrix $M$ as defined in Definition 7;
 3: Compute $P_v(M)$;
 4: for each sequence of consecutive 1's (separated by 0's) in $P_v(M)$ do
 5:     Let $s$ and $f$ be segment positions corresponding to the starting and final 1's;
 6:     Construct an adjacency segment sub-matrix $N$ that consists of the $s$-th, ..., $f$-th
        columns of $M$;
 7:     Compute $P_h(N)$;
 8:     if all elements in $P_h(N)$ are 1 then
 9:         Add $T_{sub} = (t_s, \dots, t_f)$ in $\mathcal{R}$;
10:     end if
11: end for
12: Return $\mathcal{R}$;
```

Lemma 2 *The sub-trajectory T_{sub} obtained by ProjStitch satisfies $D(Q, T_{\mathrm{sub}}) \leq \epsilon$.*

Proof The adjacency segment sub-matrix N constructed in line (6) corresponds to a sub-trajectory $T_{\mathrm{sub}} = (t_s, \dots, t_f)$ composed of contiguous data segments $t_s, \dots, t_f$. Since all element values in the horizontal projection $P_{\mathrm{h}}(N)$ are 1 in line (8), there exists $i\,(s \leq i \leq f)$ such that $d_{\mathrm{seg}}(t_i, q_j) \leq \epsilon$ for all j, and therefore, it holds that $D_H(T_{\mathrm{sub}}, Q) \leq \epsilon$ according to Definition 3. Since all element values in the vertical projection $P_{\mathrm{v}}(N)$ are also 1 in line (4), there exists j such that $d_{\mathrm{seg}}(q_j, t_i) \leq \epsilon$ for all

Table 3 Complexities of each phase and the entire algorithm

Phase	Complexity
Index searching	$O(\|Q\| \log N)$
Filtering	$O(\|Q\|N)$
Stitching	$O(\|Q\|N)$
Entire algorithm	$O(\|Q\|N)$

i, and therefore, it also holds that $D_H(Q, T_{\text{sub}}) \leq \epsilon$. By combining these, we obtain $D(T_{\text{sub}}, Q) \leq \epsilon$ according to Definition 2, and therefore, the sub-trajectory T_{sub} is a similar sub-trajectory of Q, i.e., ProjStitch is sound. □

Lemma 3 *The similar sub-trajectory T_{sub} obtained by ProjStitch is a maximal sub-trajectory.*

Proof The adjacency segment sub-matrix N corresponding to T_{sub} is constructed for a maximal sequence S of contiguous 1's divided by one or more 0's in $P_{\text{v}}(M)$. Any super-sequence S' of S must contain at least one 0, which indicates that there exists no matching data segment for every query segment, i.e., for the data segment t'_i corresponding to the 0, it holds that $d_{\text{seg}}(t'_i, q_j) > \epsilon$ for all j. For the sub-trajectory T'_{sub} corresponding to S', we obtain $D_H(T'_{\text{sub}}, Q) > \epsilon$ or $D_H(Q, T'_{\text{sub}}) > \epsilon$ according to Definition 3 and therefore $D(T'_{\text{sub}}, Q) > \epsilon$. Since any sub-trajectory T'_{sub} properly containing T_{sub} is not similar to Q, T_{sub} is a maximal sub-trajectory. □

Lemma 4 *There is no falsely dropped maximal sub-trajectory by ProjStitch.*

Proof We prove by contradiction. Assume that there exists a maximal sub-trajectory M_f that ProjStitch has missed. For any similar sub-trajectory T_f of M_f (including M_f itself), there exists an adjacency segment sub-matrix N_f in which the cells corresponding to similar segment pairs $(q_j, t_i)\,(q_j \in Q, t_i \in T_f)$ have a value 1. Since, as shown in line (8) of Algorithm 2, ProjStitch handles all adjacency segment sub-matrices N whose vertical projection vectors $P_{\text{h}}(N)$ consist of all 1 values, the vertical projection vector $P_{\text{h}}(N_f)$ must have at least one 0 value. That indicates that there exists at least one q_j that has no similar segments t_i, i.e., $d_{\text{seg}}(q_j, t_i) > \epsilon$ for all t_i. Therefore, by Definitions 1 and 2, it holds that $D_H(T_f, Q) > \epsilon$ and thus $D(T_f, Q) > \epsilon$, i.e., T_f is *not* a similar sub-trajectory—resulting in contradiction. Therefore, there is no falsely dropped maximal sub-trajectory by ProjStitch, i.e., ProjStitch is complete. □

4.5 Complexity analysis

Table 3 summarizes the complexities of each phase and the entire algorithm. In the index searching phase, since we perform index searching for every segment $q_j\,(1 \leq j \leq |Q|)$ in the query trajectory Q and the complexity of range search on a multidimensional index is $O(\log N)$ [11], the complexity of this phase is

Table 4 Additional information on datasets

Dataset	GeoLife	Hurricane	Athens trucks	Random
Number of data trajectories	45,782	1,566	1,081	20,000
Number of data segments	15,189,025	55,569	106,986	4,216,640
Average segment length	0.000654	0.013185	0.003822	0.000113
Dataset storage size	806 MB	2.9 MB	5.7 MB	249 MB
Index storage size	10.95 MB	52 KB	88 KB	3.48 MB

$O(|Q| \log N)$, where N is the number of all segments in the database $\mathcal{D}$. Any multi-dimensional index can be used in this phase. In the filtering phase, we compute the distance $d_{\text{seg}}()$ to all candidate segments t_i for each query segment q_j. The candidate segments are the data segments residing within ϵ distance from q_j, and the number of candidate segments is proportional to $\epsilon^d N$, where d is the dimension of trajectory data. By considering ϵ^d a constant, the complexity of the filtering phase is $O(|Q|N)$. In the stitching phase, we compute the vertical and the horizontal projections (i.e., OR operations) for each data trajectory obtained in the filtering phase. In the worst case, the projection should be performed for all data trajectories in $\mathcal{D}$, and therefore the complexity is $O(|Q|N)$. By combining the complexities of all phases, we obtain the complexity $O(|Q|N)$ of the entire algorithm. As indicated by the complexity, the execution time of MaTIS is linearly proportional to the dataset size N. This scalability is highly desirable especially when dealing with a large-scale trajectory database. We confirm the scalability of MaTIS through experiments in the next section.

5 Experimental evaluation

In this section, we perform a series of experiments using real and synthetic datasets to evaluate our trajectory similarity measure and sub-trajectory matching algorithm MaTIS. We compare both the accuracy of our similarity measure (see Sect. 5.1) and the performance of MaTIS (see Sect. 5.2) with those of previous state-of-the-art methods including DTW, LCSS, EDR, ERP, and EDS. We then confirm the complexity of MaTIS as analyzed theoretically in the previous section (see Sect. 5.3). All experiments are performed on Ubuntu 14.10 Linux installed on a workstation equipped with Intel i7-4790 3.6GHz CPU, 32GB RAM, and 256GB SSD. For implementation of MaTIS, we modified the open source R*-tree [3] to suit our purpose. The real and synthetic datasets used in the experiments are briefly described below with additional information summarized in Table 4. The regions of all datasets are mapped to a square region of 1.0×1.0 size, and the coordinates of all trajectory points are also transformed accordingly.

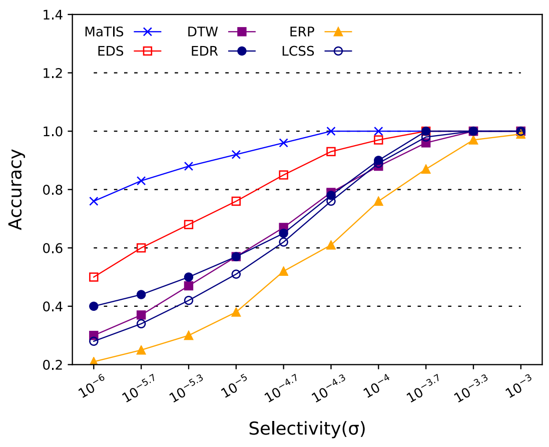
GeoLife [31] GPS trajectories of automobiles, bicycles, airplanes, and pedestrians collected from more than 30 cities in China, mostly from Beijing, between April 2007 and October 2011.[2] In this paper, we use only the Beijing data.
Hurricane [16] Hurricane trajectories that occurred in the North Atlantic Basin region between the years 1851 and 2010[3].
Athens trucks [27] 1100 trajectories composed of 112,300 GPS positions of 50 trucks transporting concrete in Athens.[4]
Random Synthetic trajectories generated by choosing a random starting point in the entire data region and then appending a series of random points within the specified distance from the previous one.

5.1 Similarity measure accuracy

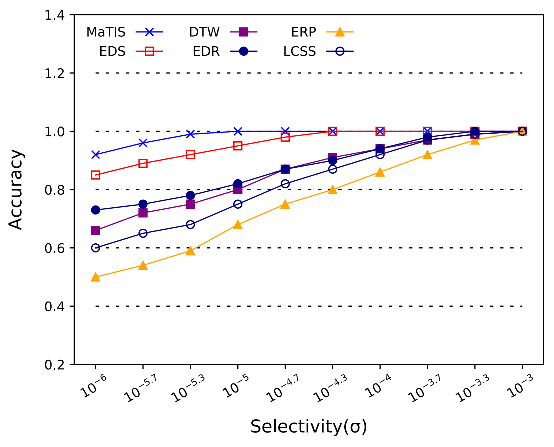
In this section, we compare the accuracy of our similarity measure $D()$ in Definition 2 with that of DTW, LCSS, EDR, ERP, and EDS using all the datasets mentioned above. A query trajectory Q is generated by choosing an arbitrary data trajectory T from a dataset and then extracting an arbitrary sub-trajectory of the specified length from T. Each point in Q is then slightly moved to a random direction within 5% of the average segment length. The query length $|Q|$ is specified as 20, 40, and 60 as in [27]. The accuracy is computed in the same way as in [27], i.e., if the original sub-trajectory is contained in the search result set of a query trajectory, we consider it as a correct answer.

Figure 10 presents the result. Every measurement is computed with 100 different query trajectories. A search range ϵ is computed for each combination of a query trajectory Q and a selectivity $\sigma\,(0 \leq \sigma \leq 1)$; we compute the distances $D()$ to all possible data sub-trajectories from Q, and then, ϵ is set to be the distance of the σSth nearest data sub-trajectory, where S is the number of all possible data sub-trajectories. An accuracy 0.8 indicates that 80 of 100 queries returned correct answers. As explained in Sect. 3, our trajectory similarity measure is based on the widely used segment distance measure $d_{\text{seg}}() = \text{dist}()$ defined by Lee et al. [16–18]. While the accuracy has an expected tendency to increase with the increase in selectivity, the accuracy of our similarity measure is always higher than that of EDS, which demonstrates highest accuracy among all previous measures including DTW, LCSS, EDR, and ERP. Our measure is better than EDS by 16.4%, 3.1%, 15.6%, and 7.7%, respectively, for each dataset on average. The improvement of MaTIS over previous measures becomes more significant as the selectivity decreases. This is a highly desirable property in large-scale databases where sub-trajectory queries with smaller selectivities are much frequent than those with larger selectivities. The improvement ratio reaches up to 52.0% than EDS with the selectivity $\sigma = 10^{-6}$ using the GeoLife dataset.

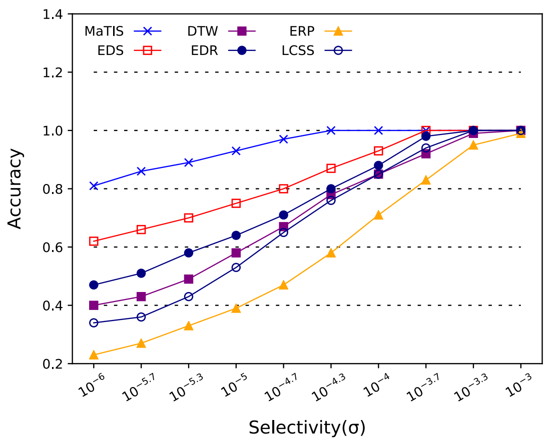
[2] http://research.microsoft.com/en-us/projects/geolife/.
[3] http://www.nhc.noaa.gov/data/.
[4] http://www.chorochronos.org/?q=node/5.



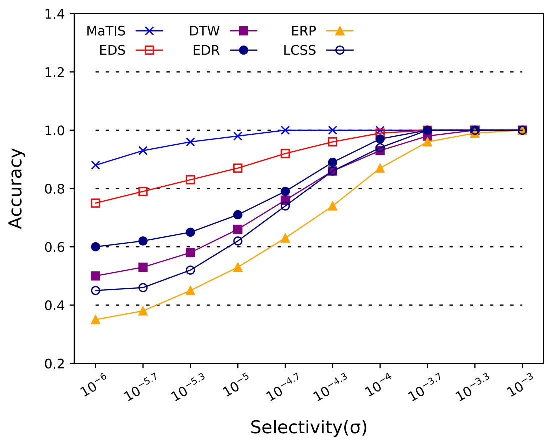
(a) GeoLife dataset.



(b) Hurricane dataset.



(c) Athens trucks dataset.



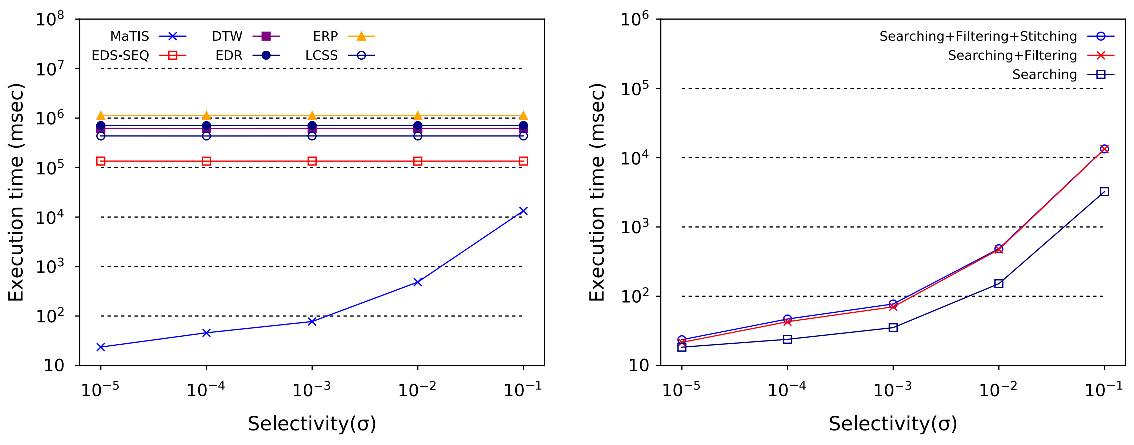
(d) Random dataset.

Fig. 10 Comparison of accuracy of our trajectory similarity measure

5.2 Search performance

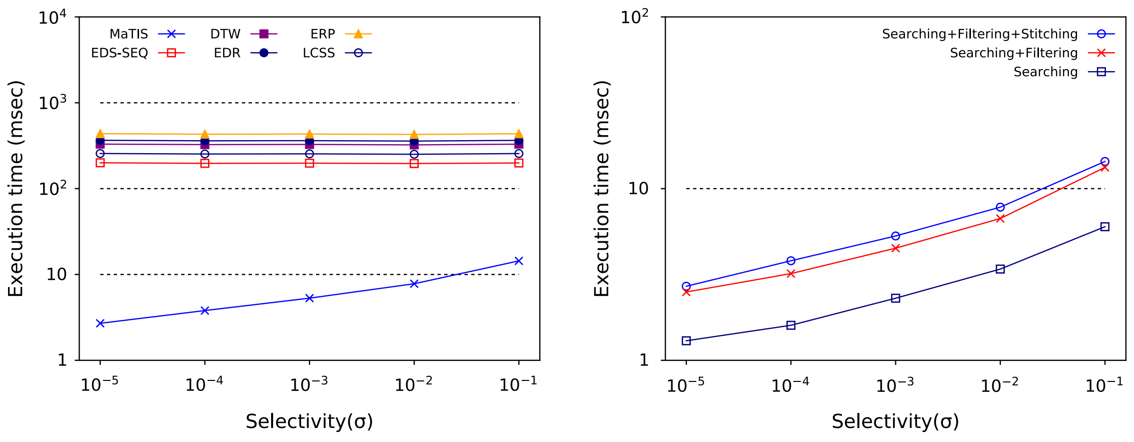
In this section, we evaluate the search performance of our algorithm MaTIS. We compare sub-trajectory matching performance with the algorithms using previous state-of-the-art measures including DTW, LCSS, EDR, ERP, and EDS. We call the algorithm using EDS as *EDS-SEQ*. We perform experiments using the GeoLife, Hurricane, Athens trucks, and Random datasets. Every experiment is performed for each of the 20 different query trajectories, and we take the average as the result. As mentioned earlier in this paper, since the algorithms using DTW, LCSS, EDR, and ERP are whole trajectory matching algorithms, when applying them in sub-trajectory matching, all possible sub-trajectories T_{sub} in each trajectory T in the dataset $\mathcal{D}$ must be compared with the query trajectory Q.

Figure 11 presents the result as the selectivity is varied from 10^{-5} to 10^{-1} using the GeoLife dataset. Note that the vertical axes are presented in the log scale. The search range ϵ is computed in the same manner as in the previous subsection. Figure 11a compares the execution time of MaTIS with the algorithms using DTW, LCSS, EDR, ERP, and EDS. Since all the algorithms except MaTIS inspect all data trajectories one by one regardless of selectivity, their execution time remains almost



(a) Comparison of execution time. **(b)** Execution time for each phase.

Fig. 11 Comparison of execution time as the selectivity is varied (GeoLife dataset)

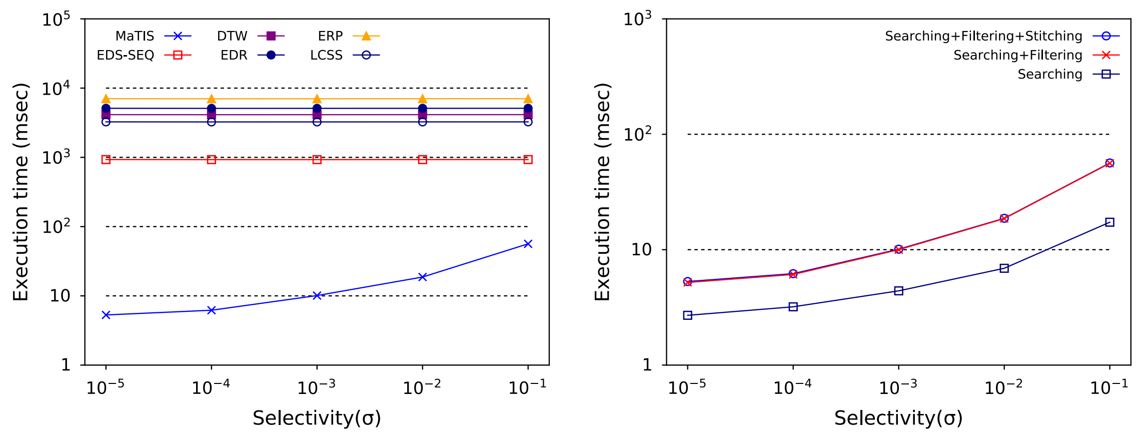


(a) Comparison of execution time. **(b)** Execution time for each phase.

Fig. 12 Comparison of execution time as the selectivity is varied (Hurricane dataset)

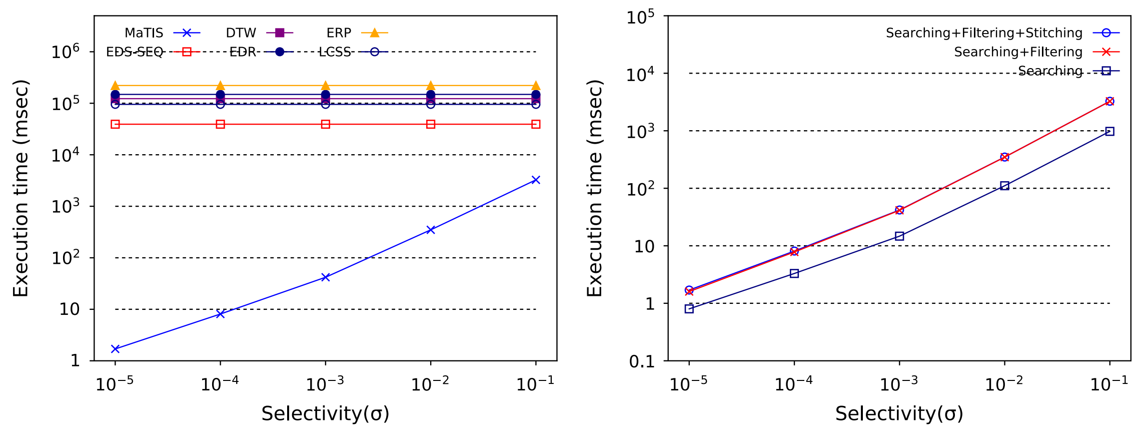
constant. On the contrary, the execution time of MaTIS increases as the selectivity grows since the number of candidate data segments within the search region increases. Figure 11b presents the execution time required for index searching, filtering, and stitching phases of MaTIS. Very little time is needed for the stitching phase. In fact, the bit operations for vertical and horizontal projections on adjacency segment matrices in the stitching phase are much less costly compared with the operations of determining the similarity between a query segment and an MBR in the index searching phase and computing the segment distances in the filtering phase. MaTIS dramatically outperformed EDS-SEQ, which demonstrated the highest performance among the algorithms using previous measures, by up to 5739 times for $\sigma = 10^{-5}$.

Figures 12, 13 and 14 present the results using the Hurricane, Athens trucks, and Random datasets; we obtain trends similar to those shown in Fig. 11 for the GeoLife dataset. As a result using the Random dataset, MaTIS outperformed EDS-SEQ by up to 22,543 times for $\sigma = 10^{-5}$. EDS-SEQ finds all sub-trajectories similar to a



(a) Comparison of execution time. **(b)** Execution time for each phase.

Fig. 13 Comparison of execution time as the selectivity is varied (Athens trucks dataset)



(a) Comparison of execution time. **(b)** Execution time for each phase.

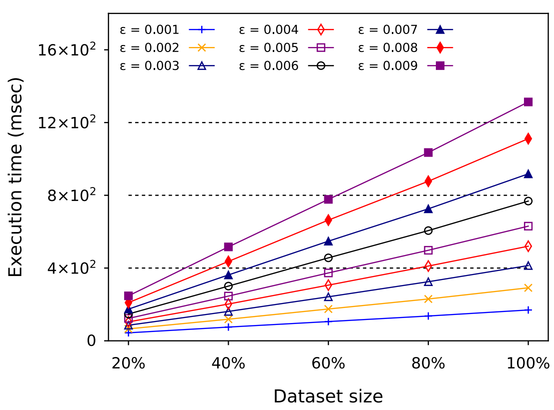
Fig. 14 Comparison of execution time as the selectivity is varied (Random dataset)

query trajectory Q in a data trajectory T ($\in \mathcal{D}$) by computing EDS with all possible *suffices* (of complexity $O(n)$) rather than with all possible sub-trajectories (of complexity $O(n^2)$), where n is the number of segments in T. Therefore, for a database $\mathcal{D}$ with D trajectories, EDS-SEQ performs as many as $O(Dn)$ EDS computations, which is equal to the number of all segments in $\mathcal{D}$. On the contrary, the number of segments returned in the index search phase in MaTIS (of complexity $O(\log Dn)$) is much smaller. Accordingly, MaTIS achieves a much better search performance than EDS-SEQ.

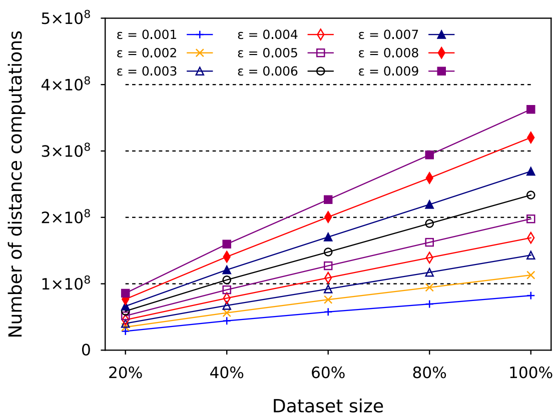
5.3 Complexity

We experimentally demonstrate the scalability of MaTIS, which is analyzed theoretically in Sect. 4.5. We compare the execution time for various dataset sizes using the GeoLife and the Random datasets.

Figure 15 presents the result using the GeoLife dataset. We used five sub-datasets containing 20%, 40%, 60%, 80%, and 100% of the entire dataset, respectively. Each

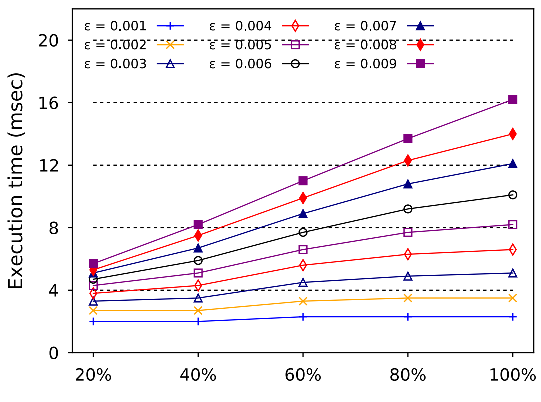


(a) Execution time.

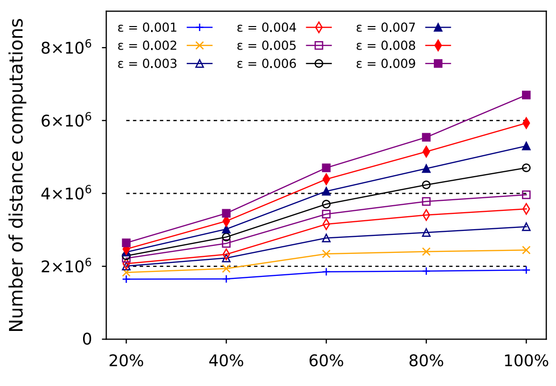


(b) Number of segment distance computations.

Fig. 15 Performance of MaTIS for various dataset sizes (GeoLife dataset)



(a) Execution time.



(b) Number of segment distance computations.

Fig. 16 Performance of MaTIS for various dataset sizes (Random dataset)

sub-dataset is generated to have a trajectory distribution that is as close as possible to the entire dataset. We set the search range as $\epsilon = 0.001, \ldots, 0.009$. Figure 15a shows the execution time of MaTIS for each sub-dataset; we can easily observe that the execution time increases linearly to the sub-dataset size. Figure 15b shows the number of segment distance computations in the filtering phase, which has a trend similar to that of the execution time in Fig. 15a. We had similar performance trends for different ranges of ϵ.

Figure 16 presents the result of the same experiment using the Random dataset. The result using the Random dataset demonstrates a performance trend similar to that using the GeoLife dataset; i.e., the execution time of MaTIS increases linearly as the sub-dataset size increases. The scalability of MaTIS confirmed by these experiments is a highly desirable property especially when dealing with large-scale trajectory databases.

6 Conclusions

In this paper, we proposed an algorithm called *MaTIS* for indexing-based multi-segment sub-trajectory matching. MaTIS partitions each data trajectory into component segments, which are individually stored in a multidimensional index. MaTIS individually searches for segments similar to each component segments of Q by using the index and then reconstructs data sub-trajectories similar to Q by 'stitching' those segments by our *ProjStitch* algorithm. ProjStitch is novel and innovative in the sense that it facilitates segment-wise partitioning and indexing. Without ProjStitch, trajectory-wise operations would lead to severe storage space overhead and search performance degradation. The partition-and-stitch framework of MaTIS is an entirely new approach. We adopted the widely used segment similarity measure by Lee et al. [16–18] and extended it to handle multi-segment trajectories using the Hausdorff distance. Our approach is the first that uses indexing for sub-trajectory matching. The indexing is made possible by our partition-and-stitch framework. We compared our method with EDS [27], a state-of-the-art sub-trajectory matching algorithm, which cannot employ indexing. The results illustrate that the accuracy of our similarity measure is better than that of EDS by up to 52.0%, and our algorithm significantly outperforms EDS-SEQ by up to 22,543 times. MaTIS is linearly scalable in the size of the database and therefore efficiently handles large-scale databases.

Appendix

Lemma 5 *For any two segments L_i and L_j, the following always holds:*

$$\mathrm{dist}(L_i, L_j) \geq d_{\mathrm{seg},0}(L_i, L_j), \tag{11}$$

where $\mathrm{dist}(L_i, L_j) = w_\perp \cdot d_\perp(L_i, L_j) + w_\| \cdot d_\|(L_i, L_j) + w_\theta \cdot d_\theta(L_i, L_j)$ *and* $w_\perp = w_\| = w_\theta = 1$ [16].

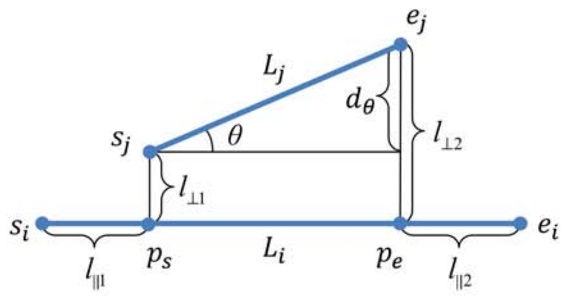
Proof Let $\mathcal{L}_i$ and $\mathcal{L}_j$ be two lines containing two segments L_i and L_j, respectively. Let p_s and p_e be the projection points of two end points s_j and e_j of L_j onto $\mathcal{L}_i$, respectively. Without loss of generality, we assume $d(s_j, p_s) \leq d(e_j, p_e)$. We also assume that L_i is longer than L_j as in [16]. We prove for the following three cases:

Case 1: p_s is located on L_i.

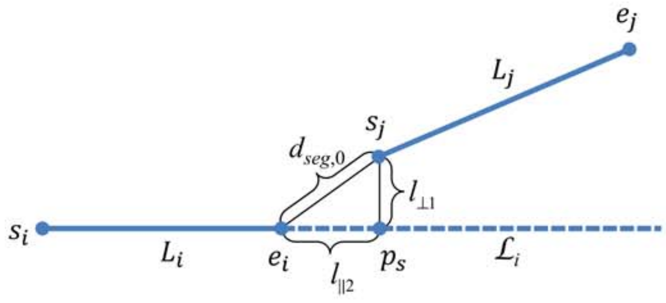
As shown in Fig. 17a, $d_{\mathrm{seg},0}(L_i, L_j) = l_{\perp 1}$. Thus,

$$\begin{aligned}\mathrm{dist}(L_i, L_j) \geq d_\perp(L_i, L_j) &= \frac{l_{\perp 1}^2 + l_{\perp 2}^2}{l_{\perp 1} + l_{\perp 2}} \\ &\geq \frac{l_{\perp 1}^2 + l_{\perp 1} l_{\perp 2}}{l_{\perp 1} + l_{\perp 2}} = l_{\perp 1} \\ &= d_{\mathrm{seg},0}(L_i, L_j).\end{aligned}$$

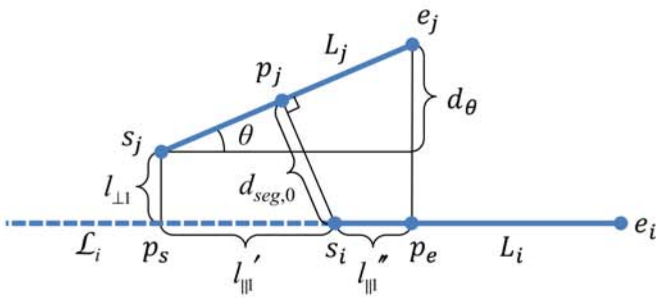
Fig. 17 Similarity measure between two segments L_i and L_j



(a) Case 1.



(b) Case 2.



(c) Case 3.

Case 2: p_s is located behind e_i.

As shown in Fig. 17b, $d_{\text{seg},0}(L_i, L_j) = d(e_i, s_j)$. Thus,

$$\begin{aligned}\text{dist}(L_i, L_j) &\geq d_\perp(L_i, L_j) + d_\|(L_i, L_j)\\ &\geq l_{\perp 1} + l_{\|2} \geq d(e_i, s_j)\\ &= d_{\text{seg},0}(L_i, L_j).\end{aligned}$$

Case 3: p_s is located in front of s_i.

Let p_j be the projection point of s_i in L_i onto $\mathcal{L}_j$, then $d_{\text{seg},0}(L_i, L_j) \leq d(s_i, p_j)$. If it holds that $l'_{\|1} \leq l''_{\|1}$, then $d_\|(L_i, L_j) = l'_{\|1}$. Thus,

$$\begin{aligned}\text{dist}(L_i, L_j) &\geq d_\perp(L_i, L_j) + d_\|(L_i, L_j) \geq l_{\perp 1} + l'_{\|1}\\ &\geq d(s_i, s_j) \geq d(s_i, p_j)\\ &\geq d_{\text{seg},0}(L_i, L_j).\end{aligned}$$

If it holds that $l'_{\|1} \geq l''_{\|1}$, then $d_{\|}(L_i, L_j) = l''_{\|1}$. Thus,

$$\begin{aligned}\mathrm{dist}(L_i, L_j) &= d_{\perp}(L_i, L_j) + d_{\|}(L_i, L_j) + d_{\theta}(L_i, L_j) \\ &\geq l_{\perp 1} + l''_{\|1} + d_{\theta} \geq d(s_i, e_j) \geq d(s_i, p_j) \\ &\geq d_{\mathrm{seg},0}(L_i, L_j).\end{aligned}$$

Therefore, combining these three cases, it always holds that $\mathrm{dist}(L_i, L_j) \geq d_{\mathrm{seg},0}(L_i, L_j)$. □

References

1. Alamri S, Taniar D, Safar M (2014) A taxonomy for moving object queries in spatial databases. Future Gener Comput Syst 37:232–242
2. Atev S, Miller G, Papanikolopoulos NP (2010) Clustering of vehicle trajectories. IEEE Trans Intell Transp Syst 11(3):647–657
3. Beckmann N, Seeger B (2009) A revised R*-tree in comparison with related index structures. In: Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD), pp 799–812
4. Buchin K, Buchin M, Kreveld MV, Luo J (2009) Finding long and similar parts of trajectories. In: Proceedings of ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS), pp 296–305
5. Chen J, Leung MKH, Gao Y (2003) Noisy logo recognition using line segment Hausdorff distance. Pattern Recognit 36(4):943–955
6. Chen L, Ng R (2004) On the marriage of Lp-norms and edit distance. In: Proceedings of International Conference on Very Large Data Bases (VLDB), pp 792–803
7. Chen L, Ozsu MT, Oria V (June 2005) Robust and fast similarity search for moving object trajectories. In: Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD), pp 491–502
8. Ding X, Chen L, Gao Y, Jensen CS, Bao H (2018) UlTraMan: a unified platform for big trajectory data management and analytics. Proc VLDB Endow 11(7):787–799
9. Dong Y, Pi D (2018) Novel privacy-preserving algorithm based on frequent path for trajectory data publishing. Knowl Based Syst 148:55–65
10. Eberly DH (2006) 3D game engine design: a practical approach to real-time computer graphics, 2nd edn. Morgan Kaufmann, Burlington
11. Ester M, Kriegel H-P, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp 226–231
12. Frentzos E, Gratsias K, Theodoridis Y (2007) Index-based most similar trajectory search. In: Proceedings of IEEE International Conference on Data Engineering (ICDE), pp 816–825
13. Hung C-C, Peng W-C, Lee W-C (2015) Clustering and aggregating clues of trajectories for mining trajectory patterns and routes. VLDB J 24(2):169–192
14. Huttenlocher DP, Kedem K (1990) Computing the minimum Hausdorff distance for point sets under translation. In: Proceedings of ACM annual symposium on computational geometry (SCG), pp 340–349
15. Kaplan E, Gürsoy ME, Nergiz ME, Saygin Y (2018) Location disclosure risks of releasing trajectory distances. Data Knowl Eng 113:43–63
16. Lee J-G, Han J, Whang K-Y (2007) Trajectory clustering: a partition-and-group framework. In: Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD), pp 593–604
17. Lee J-G, Han J, Li X (2008) Trajectory outlier detection: a partition-and-detect framework. In: Proceedings of IEEE International Conference on Data Engineering (ICDE), pp 140–149

18. Lee J-G, Han J, Li X, Gonzalez H (2008) TraClass: trajectory classification using hierarchical region-based and trajectory-based clustering. Proc VLDB Endow (PVLDB) 1(1):1081–1094
19. Mao J, Sun P, Jin C, Zhou A (2018) Outlier detection over distributed trajectory streams. In: Proceedings of SIAM International Conference on Data Mining (SDM), San Diego, pp 64–72
20. Mao Y, Zhong H, Xiao X, Li X (2017) A segment-based trajectory similarity measure in the urban transportation systems. Sensors 17(3):524
21. Nutanong S, Jacox EH, Samet H (2011) An incremental Hausdorff distance calculation algorithm. Proc VLDB Endow (PVLDB) 4(8):506–517
22. Pelekis N, Tampakis P, Vodas M, Doulkeridis C, Theodoridis Y (2017) On temporal-constrained sub-trajectory cluster analysis. Data Min Knowl Discov (DMKD) 31(5):1294–1330
23. Ranu S, Deepak P, Telang AD, Deshpande P, Raghavan S (2015) Indexing and matching trajectories under inconsistent sampling rates. In: Proceedings of IEEE International Conference on Data Engineering (ICDE), pp 999–1010
24. Shang Z, Li G, Bao Z (2018) DITA: distributed in-memory trajectory analytics. In: Proceedings of International Conference on Management of Data (SIGMOD), Houston, pp 725–740
25. Vlachos M, Kollios G, Gunopulos D (2002) Discovering similar multidimensional trajectories. In: Proceedings of IEEE International Conference on Data Engineering (ICDE), pp 673–684
26. Wolfson O, Xu B, Chamberlain S, Jiang L (1998) Moving objects databases: issues and solutions. In: Proceedings of IEEE International Conference on Scientific and Statistical Database Management, pp 111–122
27. Xie M (2014) EDS: a segment-based distance measure for sub-trajectory similarity search. In: Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD), pp 1609–1610
28. Xie D, Li F, Phillips JM (2017) Distributed trajectory similarity search. Proc VLDB Endow (PVLDB) 10(11):1478–1489
29. Yi B-K, Jagadish HV, Faloutsos C (1998) Efficient retrieval of similar time sequences under time warping. In: Proceedings of IEEE International Conference on Data Engineering (ICDE), pp 201–208
30. Yuan G, Sun P, Zhao J, Li D, Wang C (2017) A review of moving object trajectory clustering algorithms. Artif Intell Rev 47(1):123–144
31. Zheng Y, Zhang L, Xie X, Ma W-Y (2009) Mining interesting locations and travel sequences from GPS trajectories. In: Proceedings of International Conference on World Wide Web (WWW), pp 791–800
32. Zheng Y, Zhou X (eds) (2011) Computing with spatial trajectories. Springer, Berlin

Affiliations

Jae-Jun Yoo[1] · Woong-Kee Loh[2] · Kyu-Young Whang[1]

✉ Woong-Kee Loh
wkloh2@gachon.ac.kr

✉ Kyu-Young Whang
kywhang@mozart.kaist.ac.kr

Jae-Jun Yoo
phyntier@kaist.ac.kr

[1] School of Computing, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Republic of Korea

[2] Department of Software, Gachon University, Seongnam, Republic of Korea