Information Sciences 606 (2022) 549–572

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

X-FIST: Extended flood index for efficient similarity search in massive trajectory dataset

Hani Ramadhan [a], Joonho Kwon [b,*]

[a] Big Data Department, Pusan National University, Busan 46241, South Korea
[b] School of Computer Science and Engineering, Pusan National University, Busan 46241, South Korea

ARTICLE INFO

Article history:
Received 26 October 2021
Received in revised form 30 March 2022
Accepted 16 May 2022
Available online 23 May 2022

Keywords:
Learned index
Similar trajectory search
Indexing
Database management

ABSTRACT

Similarity search tasks in big trajectory datasets often require tree-based indices to shorten the query time through early pruning of dissimilar trajectories early. However, tree-based indices have been outperformed by the learned index in skewed-distribution datasets of multidimensional point experimentally. The learned index performed faster because of its data distribution awareness and machine learning model-based prediction. Directly applying learned index to trajectories can lead to inefficient query performance due to repeating range queries according to the query trajectory length. Thus, we develop X-FIST, an extended Flood index to learn the Minimum Bounding Region of the trajectories and their sub-trajectories. In similarity search, X-FIST prunes dissimilar trajectories effectively independent to the query trajectory length. If the trajectory similarity distance function changes, X-FIST does not need to train new models of its Flood index. The experimental results on three real-world trajectory datasets demonstrate that our approach shortened query time in every distance function and produced better storage size reduction than the tree-based index and direct approach of learned index.

1. Introduction

Improving the speed of trajectory similarity search [1–3] has become important because of the intensive trajectory data collection [4,5] and increasing demand for trajectory analysis tasks in ride-hailing and smart city applications [6–10]. However, computing the trajectory similarity measure between query trajectory and existing trajectories in the dataset requires $O(m^2)$ time, where m is the average length of the trajectories. Improving the speed of the similarity search involves pruning dissimilar trajectories as early as possible. Thus, in a massive dataset, it is possible to compute the similarity measure of the trajectory query with a much smaller pruned candidate set of trajectories instead of the full dataset.

In general, pruning in similarity search employs a tree-based index structure to identify dissimilar trajectories. However, tree-based indices [11,12] demonstrated lesser performance in the case of skewed data distribution of point datasets compared to the recently popularized machine learning (ML)-based indices, called learned index (LI) [13–15]. The skewed data distribution tends to occur in trajectory datasets owing to certain popular routes passed by numerous vehicles, for example, highways and routes to office areas. In tree-based indices, the skewed dataset results in the formation of multiple levels of minimum bounding rectangles (MBRs) over a more popular region that overlaps each other, which results in the creation of

* Corresponding author.
E-mail addresses: hani042@pusan.ac.kr (H. Ramadhan), jhkwon@pusan.ac.kr (J. Kwon).

https://doi.org/10.1016/j.ins.2022.05.062
0020-0255/

an imbalanced and a less storage-efficient tree. Moreover, the imbalanced tree can also cause an ineffective similarity search by traversing multiple levels of MBR.

Learned indices [13] can solve the skewed data distribution problem by modelling the key distributions in the sorted array. Thus, they can efficiently predict the positions provided the keys. In addition, compared to the tree-based indices, LIs demonstrated a better query time in range queries coupled with more efficient storage. Further, the multidimensional LIs [16,17,14,18,19] can be used for trajectory data. A trajectory can be considered as a sequence of multidimensional points in an n-dimensional space. However, a similarity search with multidimensional LI can be challenging owing to their point-based range queries.

Computing trajectory similarity measure requires the retrieval of relevant points to all the points of the query trajectory. Thus, retrieving those points precisely with multidimensional LIs requires multiple range queries. Range queries with LIs involve scanning the predicted results to remove irrelevant points outside the range query boundary. Then, we need to match the retrieved points to the query trajectory and then output the similar trajectories. The cost of "scan" and match operation are expensive as they require accessing the actual points in the dataset.

In contrast, a range query with the boundary of the MBR of the query trajectory can facilitate the reduction number of scans required to one. However, the MBR-based range query returns a larger candidate set than the multiple point-based range queries of LI. Larger candidate set implies more executions of the trajectory similarity computation, which we desire to minimize.

The trade-off between one MBR-based range query and multiple point-based range queries with LI must balance the level of precision and computational cost. Thus, some tree-based similarity trajectory indexes [20,21] reduces the complexity of the trajectory by splitting it to a few sub-trajectories. Consequently, we can perform range queries with the MBR of the split query sub-trajectories for a similarity search pruning. Such pruning require fewer range queries than point-based LI pruning, but with higher precision than MBR-based pruning. However, the number of "scans" still needs to be reduced in such pruning to achieve an efficient similarity search.

Other similar trajectory search techniques, [22,23] which learn the vector representation of the trajectories, demonstrated better performance than tree-based indices for trajectory data [11,24]. The vector representation suggests a linear time $O(m)$ to compute the trajectory similarity measure. However, this approach has two main drawbacks. First, learning the vector representation of trajectories requires a large training data which may not always be available. When the data distribution changes, the training set must be re-examined carefully to produce the appropriate vector representation. The trajectory similarity search, however, should work without the aforementioned training set. Second, the vector representation approach can introduce false negative results, which are undesirable in similarity search.

Thus, we propose an extended Flood index for an efficient similarity search (X-FIST) which learns the distribution of trajectories and sub-trajectories in a massive trajectory dataset. Each trajectory and sub-trajectory is represented as an MBR that consists of minimum (MIN) and maximum (MAX) points. Hence, only two points are indexed instead of all points in the trajectories and sub-trajectories, thereby reducing the build time and the size of the Flood index. The pruning of X-FIST does not use the point-based range queries anymore as we represent the trajectories and sub-trajectories as MBR. Further, two-level pruning steps with MBR representations with X-FIST were devised, which eliminated the need to scan in every range query. The exact points in the dataset are examined at the final step, which is the trajectory similarity computation on the query trajectory and final candidate set. X-FIST also supports various similarity measures of trajectories without the need to retrain models or change the elements of the Flood index. In addition, the time to build X-FIST do not require hours because of the simple MBR representation thus our approach is practical to handle data distribution change.

Our key contributions can be summarized as follows:

- We proposed an extended flood index with efficient trajectory representations which yields a small sized index with shorter build time.
- A novel search strategy was presented with our index to effectively prune dissimilar trajectories in various trajectory distance functions.
- An extensive experiment on three real world trajectory dataset was conducted, wherein the proposed approach demonstrated the best performance in overall query time.

The remainder of this paper is organized as follows. The research related to our approach has been described in Section 2. Essential notions and formulation of our problem are presented in Section 3. Sections 4 and 5 explain the index building and search strategy, respectively. Further, Section 6 shows the experimental results and discussions. Finally, Section 7 presents the conclusion of the study.

2. Related works

Similar trajectory search with tree-based indices. In the similarity search, the index identifies similar trajectory candidates that have sizes significantly smaller than the original dataset. Tree-based indices prune dissimilar trajectories via non-intersection of MBRs of existing trajectories to the MBR of the query trajectory [20,25]. However, tree indices exhibit poor performance in a skewed data distribution wherein the trajectories are spread unevenly and resulting in minimal non-

intersection and ineffective pruning. In addition, the size of in-memory tree-based indices is typically large such that it may result in an to out-of-memory error [20,21,25].

Learned Index. A recent study on indexing [13] proposed a solution for the problem of tree-based indices that are not aware of data distribution. The proposed index learns the distribution of the stored data by modeling a recursive model index (RMI) to output the position of the record for a particular search key in a one-dimensional space. Various enhancements on RMI [26,27] include the dynamic update operations when data insertion or deletion occurs. ALEX [26] and LIPP [27] utilize ML-based prediction to identify the data node to be reorganized. ALEX uses a gapped array node layout to accommodate the index update whereas LIPP uses tightly-bounded tree layout and adjustment strategy to efficiently update the index.

Multidimensional Learned index. Extensions of the LI [16,17,14,18,19,28] aided in performing range search queries on multi-dimensional points with impressive speed and significant reduction of index size compared to tree-based indices. ZM-Index [16] sorts the points based on the Z-order but it is difficult to train owing to the fact that Z-order does not follow the proximity of the points. Another Z-order-based LI [15] introduces a rank-based space transformation to index multidimensional points and supports index update on data insertion and deletion efficiently. Further, similar to the ZM-Index, the sorting of the grid index [17] can place spatially close points distantly in the sorted array owing to the partitioning of different dimensions. Consequently, Flood and Tsunami solved the issue of the grid-based index [17] by partitioning the data based on the density over dimensions [14] and the correlation between dimensions [18]. Flood [14] exploits the advantage of the query workload to optimize the configuration of partitioning and obtain the best dimension to index using ML models. Tsunami [18], which is an improved version of Flood, assumes that certain correlation between dimensions of the dataset and the queries are skewed. However, the application of Tsunami to trajectory data can be disadvantageous because the dimensions of the dataset, such as latitude and longitude, are mostly uncorrelated. In contrast, the ML index [19] assigns sorted array keys based on the distance of the points to certain centroid clusters. In a high-dimensional environment, the clustering-based key assignment of ML-Index exhibits effective performance; however, it causes numerous irrelevant points to appear before the refinement stage. Consequently, considering these factors, despite their advantages over tree-based indices, multidimensional indices have not been clearly applied in trajectory-based cases.

Learned indices on unstructured data. Distance-bounded spatial approximation [29] utilizes a polygon-based learned index for the polygon case. However, the polygon index primarily focuses on 2D space and does not translate well to a similar trajectory search which highlights the ordering of the points. An approach to unstructured dataset, the metric learned index [28] demonstrated the similarity search as a classification problem. However, the metric learned index is unable to directly work on trajectory data as it does not impose sequential relationships. In addition, it requires a prebuild tree index to perform both training and search for similar objects.

Deep learning-based approaches on similarity search: Several deep learning-based approaches explored the similar trajectory search with representation learning technique [23,30–32,22]. RNE [22], T2Vec [30], and NeuTraj [31] learn the vector representation of trajectory dataset with recurrent neural networks to approximate the trajectory distance computation to linear time $O(m)$ rather than the classic quadratic time. Another representation learning model, Traj2SimVec [23],addresses a trajectory proximity approach with tree-based index and trajectory simplification to compute similarity of sub-trajectories. On the other hand, a reinforcement learning technique [33] focuses on similar sub-trajectory search tasks by generating the models of sub-trajectory with the Markov Decision Process. A point-of-interest (POI)-based representation framework, GTS [32] applies Long Short-Term Memory network to generate Graph Neural Network-based embeddings over spatial road networks. GTS improves the accuracy of the retrieved similar candidate set over [30,31] because the embeddings of GTS are independent to the trajectory similarity learning and learns the partial ordering between trajectories. However, [32] only supports trajectory dataset based on spatial networks, whereas [22,30,23,33,31] support various the more general trajectory-to-trajectory distances. Moreover, performing similar search with these techniques can produce incorrect results due to the incomplete representations of the full trajectory dataset. Furthermore, [30,23,33,31] require hours to train the model, which causes retraining the model impractical when the data distribution shifts and has not been applied to space larger than 2D space.

In this study, we focus on extending multidimensional LI to prune dissimilar trajectories in a similarity search. Our proposed approach incorporates Flood [14] to learn the distribution of trajectories and sub-trajectories rather than individual points. Further, representing trajectories and sub-trajectories as MBRs may reduce the training time and introduce a new search strategy with Flood to determine MBRs that have spatial relationships to the other MBRs. Tsunami [18] was not employed as our base LI because the dimensions of the trajectory points have a low correlation. On the other hand, the index update because of data insertion, update, and deletion is beyond the scope of this study.

3. Preliminaries and problem definition

In this section, we briefly present the concepts of trajectories and trajectory similarity measures. Then, we define the problem of trajectory **similarity search pruning**. The essential notations and symbols used in this paper are summarized in Table 1.

Table 1
Essential notations and symbols.

Symbol	Description
$\mathscr{T}$	A trajectory dataset
n	Number of attributes of $\mathscr{T}$
$\|.\|$	Length of a trajectory or size of a set
P (or Q)	A (query) trajectory, sequence of points $\langle p_1, p_2, .., p_{\|P\|} \rangle$
$P^{(i)}$	Trajectory with id i in $\mathscr{T}$
p_i	i-th point of a trajectory P, consists of n values $p_i = \{p_{i,1}, p_{i,2}, p_{i,3}, .., p_{i,n}\}$
P_{a-b}	A sub-trajectory of P where $P_{a-b} = \langle p_a, .., p_b \rangle, 1 \leqslant a \leqslant b \leqslant \|P\|$
tid, sid	ID of a trajectory and a sub-trajectory in the index
$MBR(P)$	Minimum bounding rectangle of a trajectory P, consists of two extreme points: p_{min} and p_{max}
$\mathscr{D}(P, Q)$	Trajectory distance function $\mathscr{D}$ between P and Q
$\mathscr{R}$	Set of similar trajectories candidate
$d(p, q)$	Euclidean distance between two points p and q
τ	Similarity threshold of $\mathscr{D}(P, Q)$
ϵ	Matching point radius threshold for EDR and LCSS distance
δ	Spacing constraint for LCSS distance
$I_m(P_{a-b}, Q_{a-b})$	Intersection of the MBRs of two sub-trajectories P_{a-b} and Q_{a-b}

3.1. Trajectory, sub-trajectory, and MBR

Consider a moving object with its attribute always recorded as a point within each timestamp. In the spatial context, each point contains the latitude and longitude of the object in 2D outdoor space or the XY-coordinate in 2D indoor space. However, owing to the increasing enrichment of the positioning service, a point in the trajectory may contain additional attributes other than its spatial context. Note that we use the terms attribute and dimension interchangeably. We exclude the timestamp notation in trajectory as we focus on the trajectory similarity measure rather than the time attribute related queries. Thus, each point can contain n values where $n \geqslant 2$. Hence, we define a trajectory as follows:

Definition 1. A trajectory P is a finite length $|P|$ sequence of points in n-dimensional space $P = \langle p_1, p_2, \ldots, p_{|P|} \rangle$where each point consists of n attribute values $p_i = \{p_{i,1}, p_{i,2}, \ldots, p_{i,n}\}$.

A sub-trajectory is a subset of trajectory, and is defined as follows:

Definition 2. For a trajectory P and two integers a and b where $1 \leqslant a \leqslant b \leqslant |P|, P_{a-b}$ is denoted as a sub-trajectory of P, where $P_{a-b} = \langle p_a, p_{a+1}, \ldots, p_b \rangle$.

The shortest sub-trajectory P_{a-a} is a single point whereas the longest sub-trajectory $P_{1-|P|}$ is a full trajectory P. Further, a trajectory $P^{(i)} \in \mathscr{P}$ can be fully decomposed into and reconstructed from a sequence of s sub-trajectories SP such that $SP = \langle P^{(i)}_{a_1-b_1}, .., P^{(i)}_{a_s-b_s} \rangle$ where $a_1 = 1, b_s = |P|$, and $a_{j+1} = b_j + 1$.

An MBR of a trajectory (or a sub-trajectory) is the smallest possible hyperrectangle that includes all the points in the trajectory. The MBR of a trajectory P can be denoted by a set of intervals of the minimum and maximum values of each dimension of $p_{i,j}$ of P. Thus, MBR of a trajectory P (or to P_{a-b}) can be defined as follows:

Definition 3. The MBR of a trajectory P can be defined as $MBR(P) = \{p_{min}, p_{max}\}$ where $p_{min} = \left\{p_{min_j} = min_{1 \leqslant i \leqslant |P|} p_{i,j}\right\}$ and $p_{max} = \left\{p_{max_j} = max_{1 \leqslant i \leqslant |P|} p_{i,j}\right\}$. $MBR(.)$ notation also applies to the sub-trajectory P_{a-b}.

3.2. Trajectory similarity

In this study, we measured the similarity between trajectories considering spatial-only trajectory-to-trajectory distances $\mathscr{D}$ [34,35,24,36] which are extended to n-dimensional space. Two trajectories $P = \langle p_1, p_2, \ldots, p_{|P|} \rangle, Q = \langle q_1, q_2, \ldots, q_{|Q|} \rangle$ are similar when $\mathscr{D}(P, Q)$ is less than or equal to the threshold τ. Computing trajectory-to-trajectory distance requires $O\left(|P|^2\right)$ time as trajectory-to-trajectory distance must match each point in P, Q in the same order. Further, we use four different trajectory-to-trajectory-based distances: DF [34], Dynamic Time Warping (DTW) [35], Edit Distance on Real Sequence (EDR) [24], and Longest Common Subsequence (LCSS) [36]. The four distances introduce local shifts into trajectory [24], such that some trajectories have similar movement patterns but certain sub-trajectories have a slight shift in time, where in spa-

tial context, is the order of point of the trajectories. The respective formulae are described in Eqs. (1)–(4). The $cost(q_1, p_1)$ in Eq. 3 is 1 when $d(q_1, p_1) > \epsilon$ and 0 otherwise.

$$DF(Q,P) = \begin{cases} \max_{i=1}^{|Q|} d(q_i, p_1) & \text{if } |P| = 1 \\ \max_{j=1}^{|P|} d(q_1, p_j) & \text{if } |Q| = 1 \\ \max\left(d(q_1,p_1), \min\left(DF(Q, P_{2-|P|}), DF(Q_{2-|Q|}, P), DF(Q_{2-|Q|}, P_{2-|P|})\right)\right) & \text{otherwise} \end{cases} \tag{1}$$

$$DTW(Q,P) = \begin{cases} \sum_{i=1}^{|Q|} d(q_i, p_1) & \text{if } |P| = 1 \\ \sum_{j=1}^{|P|} d(q_1, p_j) & \text{if } |Q| = 1 \\ d(q_1,p_1) + \min\left(DTW(Q, P_{2-|P|}),\right. & \\ \left. DTW(Q_{2-|Q|}, P), DTW(Q_{2-|Q|}, P_{2-|P|})\right) & \text{otherwise} \end{cases} \tag{2}$$

$$EDR_\epsilon(Q,P) = \begin{cases} |P| & \text{if } |Q| = 0 \\ |Q| & \text{if } |P| = 0 \\ \min\left(EDR_\epsilon(Q_{2-|Q|}, P_{2-|P|}) + cost(q_1,p_1),\right. & \\ \left. 1 + EDR_\epsilon(Q, P_{2-|P|}), 1 + EDR_\epsilon(Q_{2-|Q|}, P)\right) & \text{otherwise} \end{cases} \tag{3}$$

$$LCSS_{\delta,\epsilon}(Q,P) = \begin{cases} |P| & \text{if } |Q| = 0 \\ |Q| & \text{if } |P| = 0 \\ LCSS_{\delta,\epsilon}(Q_{1-|Q-1|}, P_{1-|P-1|}) & \text{if } ||Q| - |P|| \leqslant \delta \,\&\& \\ & d\left(q_{|Q|}, p_{|P|}\right) \leqslant \epsilon \\ 1 + \min\left(LCSS_{\delta,\epsilon}(Q, P_{1-|P-1|}),\right. & \text{otherwise} \\ \left. LCSS_{\delta,\epsilon}(Q_{1-|Q-1|}, P)\right) & \end{cases} \tag{4}$$

Important dissimilarity characteristics are described as follows. Given two trajectories P, Q, thresholds τ, and matching radius ϵ. For DF and DTW, P and Q are dissimilar when at least one point $q_j \in Q$ exists such that it cannot reach any point in P within a radius of τ. In addition, P and Q are dissimilar in DF when the Euclidean distance of their first or last points, $d(p_1, q_1)$ and $d\left(p_{|P|}, q_{|Q|}\right)$, respectively, are larger than τ [20]. However, DTW follows a stricter rule for its similarity, wherein P and Q are dissimilar when $d(p_1, q_1) + d\left(t_{|P|}, q_{|Q|}\right) > \tau$. In contrast, in the case of EDR and LCSS, P and Q are dissimilar when every point in Q cannot reach any point in P within a radius of ϵ.

The four distances cover interesting conditions of trajectory similarity such as metric space and sensitivity to sampling error. DF is a metric-based distance whereas the other three are not. In EDR and LCSS, the sampling error is considered by the introduction of the matching radius parameter ϵ, whereas DF and DTW allow errors to contribute to their computation. However, certain pointwise similarity measures exist as well, such as Hausdorff and Edit Distance with Real Penalty (ERP). Hausdorff distance [37] disregards the order of points such as the four distances; thus, similar trajectories according to the Hausdorff distance may have opposing directions. Moreover, in terms of behavior, the Hausdorff distance is a less strict version of the DF distance. Hence, in subSection 5.3.2, we discuss certain DF characteristics that may be applicable to Hausdorff distance. Further, the ERP distance [38], which uses SUM similar to DTW, requires a reference point to measure trajectory similarity. However, the change of reference point significantly changes the similarity of two trajectories. Thus, applying similarity search with ERP distance, which behaves in different manner with the four distances, is an interesting research proposition.

3.3. Problem definition

Consider that a user performs a similarity search query in a big trajectory dataset $\mathscr{T}$ with distance function $\mathscr{D}$ and threshold τ for a particular query trajectory Q. However, as performing similarity search with $O\left(|\mathscr{T}| * |P|^2\right), P \in \mathscr{T}$ time in every trajectory is undesirable. Applying index to effectively prune the dataset is necessary. Consequently, we can formally define the pruning in similarity search problem as follows.

Definition 4. Given a query trajectory Q with distance function $\mathscr{D}$, threshold τ, and trajectory dataset $\mathscr{T}$ of n attributes, **similarity search pruning** determines the smallest candidate set of trajectories $\mathscr{R}_f = \mathscr{T} \setminus \mathscr{R}\prime$ where $\mathscr{R}\prime$ is the pruning set $\mathscr{R}\prime = \{P\prime | \mathscr{D}(P\prime, Q) > \tau \wedge P\prime \in \mathscr{T}\}$. Subsequently, the **similarity search problem** determines the trajectory set $\mathscr{R}_{similar}$ on the candidate set $\mathscr{R}_f$, instead of the full trajectory dataset $\mathscr{T}$, where $\mathscr{R}_{similar} = \{P | \mathscr{D}(P, Q) \leqslant \tau \wedge P \in \mathscr{R}_f\}$.

4. Indexing

In this section, we introduce X-FIST for the trajectory similarity search as depicted in Fig. 1. X-FIST comprises of Flood MBR, sub-trajectory table, Flood MBR-Sub, and trajectory length map. The Flood MBR and Flood MBR-Sub index the MBR of every trajectory and sub-trajectory in $\mathscr{T}$, respectively, whereas the sub-trajectory table stores the information of the trajectory ID (*tid*) and order of indexed sub-trajectory, which we use to identify the sub-trajectories *sid*. Storing *tid* and *sid* is necessary as the rule to match the points in every sub-trajectory depends on *tid* and *sid* when pruning is performed later. The trajectory length map specifically supports the EDR and LCSS-based pruning. Before describing the construction of each component, we explain a slight modification made to the Flood index [14].

4.1. Flood sorted array modification

Flood [14] sorts multidimensional points into an array according to a certain attribute and partitioning. Typically, Flood also places the pointer of each point onto the actual dataset on each entry of the sorted array. However, each entry in the sorted array is associated with additional information in X-FIST. Further, in Flood MBR, *tid* is added to each entry whereas in Flood MBR-Sub, *sid* is added, which later is discussed in the sub-trajectory table subsection. This addition does not change the behavior of the initial notion that the learned indices learn the mapping of the points to their positions in the sorted array. In Section 5, we discuss the manner in which X-FIST can use *tid* and *sid* to prune dissimilar trajectories.

4.2. Sub-trajectory table

The sub-trajectory table stores the information regarding all sub-trajectories that can fully reconstruct every trajectory in the dataset $P^{(i)} \in \mathscr{T}$. Each entry of the sub-trajectory table consists of the last index b_j of each sub-trajectory $P^{(i)}_{a_j - b_j} \in SP = \left\langle P^{(i)}_{a_1 - b_1}, .., P^{(i)}_{a_{|SP|} - b_{|SP|}} \right\rangle$ and *tid i*. In addition, the position of the entry of the sub-trajectory table is considered as the ID of a sub-trajectory *sid*. Now, we describe the construction of the sub-trajectory table that records every entry that is ordered by *tid* and the last index in Algorithm 1.

Algorithm 1: Building sub-trajectory table

Input : Trajectory dataset $\mathcal{P}$, split parameter η
Output: Sub-trajectory table $STable$
Func *BuildSTable* $(\mathcal{P}, \eta)$
  Initialize $STable$ as empty table of $< int, int >$;
  for $i \leftarrow 1$ **to** $|\mathcal{P}|$ **do**
    Set $SP = \langle P^{(i)}_{a_1\text{-}b_1}, ..., P^{(i)}_{a_{|SP|}\text{-}b_{|SP|}} \rangle$ as a list of sub-trajectories with $Split(P^{(i)}, \eta)$;
    foreach $P^{(i)}_{a_j\text{-}b_J} \in SP$ **do**
      $STable.addEntry(i, b)$; `// add an entry consisting of` i `as` ***tid*** `and` b `as last index`
  return $STable$;

Algorithm 2: Split trajectory P into sub-trajectories with η

Input : A trajectory P, split parameter η
Output: A list SP that contains the sub-trajectories of P

```
1  Func Split (P, η)
2      Set ST as empty list;
3      i ← 2;
4      for j ← i to |P| − 1 do
5          Get {p_min, p_max} = MBR(P_{i−j});
6          if max_{1≤k≤n} |p_{max,k} − p_{min,k}| > η then
7              SP.add(P_{i-(j−1)});
8              i ← j;
9      SP.add(P_{i-(|P|−1)});
10     Add P_{1-1} and P_{|P|-|P|} as start and end of SP, respectively;
11     return SP;
```

In Algorithm 1, we split each trajectory in the dataset $P^{(i)} \in \mathscr{T}$ into a sequence of sub-trajectories with the parameter η and placed them in the order of the sequence and *tid* i with Algorithm 2. Consequently, the *tid* and the last index b of each entry in the sub-trajectory table are ordered in ascending. The ordered *tid* and last index in the sub-traj table provide important details of the sub-trajectory, such as the first point, the last point, the length, and the order of the sub-trajectories in the original trajectory, as described as follows. Consider an *tid* k obtained as the result of the retrieved key from a query. A sub-trajectory with *sid* k is the first sub-trajectory of a trajectory $P^{(i)}$ when k-th entry of the sub-trajectory table is the first entry of the sub-trajectory table ($k = 1$) or its *tid* is different from the previous entry ($STable[k].tid = i \neq STable[k-1].tid$). In a similar manner, a sub-trajectory with *sid* k is the last sub-trajectory of a trajectory $P^{(i)}$ when k-th entry of the sub-trajectory table is the last entry of the sub-trajectory table ($k = STable.length$) or the *tid* of its subsequent entry is different ($STable[k].tid = i \neq STable[k+1].tid$). The length of a sub-trajectory $|P_{a-b}|$ with *sid* k can be acquired by subtracting the last index of k-th entry in the sub-trajectory table by the last index of its previous entry $STable[k].lastIndex - STable[k-1].lastIndex$ or simply $STable[k].lastIndex$ if $|P_{a-b}|$ with *sid* k is the first sub-trajectory.

Algorithm 2 provides a lightweight trajectory split with parameter η. The parameter η provides a boundary for a type of range query with the minimum and maximum point of an MBR which we will discuss later in subsection 5.1.2. As a rule of thumb, smaller η gives better range query results but more sub-trajectories whereas larger ηgives worse range query results but fewer sub-trajectories. Thus, we guarantee that the maximum range of minimum and maximum point of every MBR of the split sub-trajectories is always less or equal than η (line 6). The algorithm always splits first $p_1^{(i)}$ and last points $p_{|P|}^{(i)}$ of a trajectory $P^{(i)}$ separately and put them as the one-element first and last sub-trajectories, respectively as depicted in line 9, to conform with the dissimilarity rule of DF and DTW distances. We discuss the optimal value of split parameter η in subsection 4.4.

4.3. Flood MBR and Flood MBR-Sub

The Flood index is extended to learn the distribution of the MBR of trajectories and sub-trajectories in dataset $\mathscr{T}$. The procedure of building Flood MBR can be applied to Flood MBR-Sub by substituting the terms of the trajectories to sub-trajectories, for example MBR to MBR-sub and *tid* to *sid*. The Flood MBR consists of two different Flood indices (Flood MIN and Flood MAX) that index the minimum points and maximum points of each MBR $MBR\left(P^{(i)}\right), P^{(i)} \in \mathscr{T}$, respectively. Note that sub-trajectories are indexed the same way by interchanging $P^{(i)}$ with $P_{a-b}^{(i)}$. The construction of Flood indices with only two points (MIN and MAX points) for each trajectory significantly reduces the build time, regardless of the length $|P|$.

As described before in Section 1, we do not perform 'scan' process to minimize the computation towards the actual point values in pruning. Thus, the precision of the range query of Flood in X-FIST is only limited to the sorted dimension of Flood. Then, the Flood MIN and Flood MAX should have different sorted dimensions to improve the precision of the range query result without scan. The improved result also contributes to prune irrelevant candidates for DF and DTW distance as discussed in subsection 5.3.1.

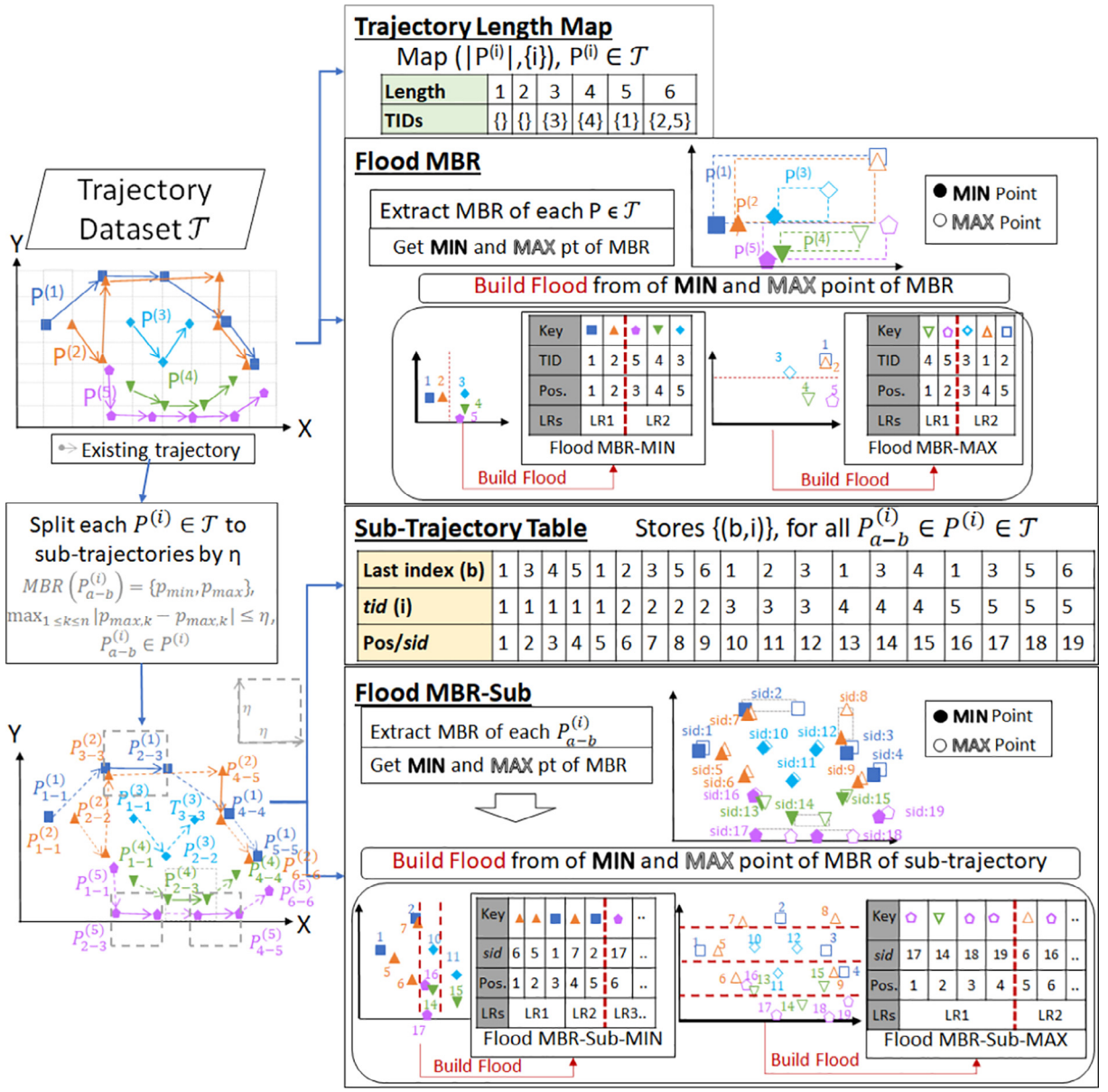


Fig. 1. The overview architecture of X-FIST. X-FIST consists of several Flood indices where each Flood index applies to each MBR of trajectories and sub-trajectories. The components of X-FIST consists of Flood MBR (Flood-MBR MIN & Flood-MBR MAX), Flood-MBR-Sub (Flood MBR-Sub-MIN & Flood-MBR-Sub MAX), a sub-trajectory table, and trajectory length map. For a particular parameter η, X-FIST splits the trajectories into sub-trajectories and constructs a sub-trajectory table containing the details of the sub-trajectories in the dataset.

We discuss how to find the optimal layout pair of Flood MIN and MAX in Algorithm 3. The optimal layout pair requires the sample of trajectory dataset $\mathcal{T}$ and the sample of query workload $\mathcal{Q}$. The query workload contains both the query trajectory set and the MBR extension parameter ϵ, wherein we generalize τ for DF and DTW as ϵ. The cost model $\mathcal{C} : (\mathcal{D}, q, L)$, where $\mathcal{D}$ is point dataset, q is range query, and L is a layout, slightly differs from [14] in that we do not require 'scan' to process our pruning. We extract the points of MBRs of the sample (sub-) trajectories and put them in separate lists (Line 5–6). Then, we extract range query boundaries according to the type of the index points (trajectory MBR or sub-trajectory MBR) which shown in Table 2 and later we discuss the cause in subsection 5.1.2. Provided the range queries as the query workload samples, we build the choice of optimal layouts with *FindOptimalLayout* with its respective points ($\mathcal{R}\mathcal{Q}_{min}$ for *minPts* and $\mathcal{R}\mathcal{Q}_{max}$ for *maxPts*) (line 10–11).

Algorithm 3: Find optimal layout pair of Flood MIN and MAX

Input : Sample trajectory dataset $\hat{\mathcal{T}}$, sample query workload $\hat{\mathcal{Q}}$, cost model $\mathcal{C} : (\mathcal{D}, q, \mathcal{L})$, split parameter η
Output: Best layout pair for Flood MBR L_{min}, L_{max}
1 **Func** ***FindOptimalLayoutPair*** $(\hat{\mathcal{T}}, \hat{\mathcal{Q}}, \mathcal{C}, \eta)$
2 Set $\mathcal{RQ}_{min}, \mathcal{RQ}_{max}$ as empty lists of range query boundaries;
3 Set $minPts$ & $maxPts$ as empty $\langle double[], int\rangle$ lists of (point,id);
4 **forall** $P^{(i)} \in \hat{\mathcal{T}}$ **do**
5 $\{p_{min}, p_{max}\} \leftarrow MBR(P^{(i)})$;
6 $minPts$.add($(p_{min}$,$i)$), $maxPts$.add($(p_{max}$,$i)$);
7 **forall** $(Q, \epsilon) \in \hat{\mathcal{Q}}$ **do**
8 rQ_{min}, $rQ_{max} \leftarrow GenerateRangeQueryPair(Q, \eta, \epsilon)$;
9 $\mathcal{RQ}_{min}$.add(rQ_{min}),$\mathcal{RQ}_{max}$.add(rQ_{max});
10 $LC_{min} \leftarrow FindOptimalLayout(MinPts, \mathcal{RQ}_{min}, \mathcal{C})$;
11 $LC_{max} \leftarrow FindOptimalLayout(MaxPts, \mathcal{RQ}_{max}, \mathcal{C})$;
`// Find best Flood MIN-MAX layout pair`
12 Set L_{min}, L_{max} as the best layout combination in $LC_{min} \times LC_{max}$ that minimizes the sum of their costs;
13 return L_{min}, L_{max};

The original *FindOptimalLayout* in [14] require the full dataset and query workload as it includes sampling and only returns the best layout with a sorted dimension. Thus, the algorithm is slightly modified to remove the sampling and to return two best layouts with different sorted dimension paired with cost. If *FindOptimalLayout* only returns one best layout of each Flood, the combination of Flood MIN and Flood MAX can have the same sorted dimensions thus leading to low precision result. Then, with two layout-cost pairs LC_{min} and LC_{max}, we find the best layout pair combination that minimizes their sum of cost efficiently. Consequently, having the pair of the layouts, we can build Flood MIN and MAX separately as described in [14].

4.4. Finding optimal trajectory split parameter

The performance of the index building and the range query on the sub-trajectories highly depend on the value of η. Thus, we estimate the best η by minimizing the cost in Algorithm 3 (line 8) simply with grid or random search. The search space of η starts from the first quartile of the maximum range between two consecutive points in a trajectory $(\max_{1 \leqslant k \leqslant n} |p_{j,k} - p_{j+1,k}|)$, $2 \leqslant j \leqslant |P| - 1$ excluding first and last point. We use the first quartile of the maximum range as the lower bound to avoid outliers (maximum range close to 0). The upper bound of η is the maximum range between MBR points $\max_{1 \leqslant k \leqslant n} |p_{max,k} - p_{min,k}|$, $MBR(P_{2-|P|-1})$ of every trajectory P in the sample dataset $\widehat{\mathcal{T}}$.

Table 2
The boundaries of range queries based on the MBRs $\{q_{min}, q_{max}\}$ of query trajectory Q and sub-trajectories SQ of Q.

Index type	Range query boundary
MIN	$[q_{min-\tau}$,$q_{min+\tau}]$
MAX	$[q_{max-\tau}$,$q_{max+\tau}]$
SUB-MIN	$[q_{min-\epsilon-\eta}$,$q_{max+\epsilon}]$
SUB-MAX	$[q_{min-\epsilon}$,$q_{max+\epsilon+\eta}]$

4.5. Trajectory length map

The trajectory length map *lengthMap* identifies the set of relevant trajectories based on the trajectory length. The EDR and LCSS distances highly depend on the length of the two trajectories when attempting to determine their similarity. Thus, the length of each trajectory is stored in a key-value map wherein the key is an integer signifying the trajectory length, and the value is a set of *tid* of all trajectories $P \in \mathscr{T}^{(i)}$. The *lengthMap* can be simply built by iterating all trajectories $P^{(i)} \in \mathscr{T}$ and updating the value *tid* in $< |P^{(i)}|, tids >$ by adding i to *tids*.

5. Similarity search

Fig. 2 presents an overview of the search using X-FIST. The similarity search accepts a query trajectory Q, threshold τ, and distance function $\mathscr{D}$ as inputs and returns a set of trajectories $\mathscr{R}_{similar}$ that are determined to be similar to Q, by following prune-and-refine phases, which is slightly similar to the branch-and-bound technique. The pruning phase consists of trajectory and sub-trajectory-level pruning. X-FIST selects candidate trajectories from $\mathscr{T}$ with Flood MBR or the length map in trajectory level pruning which then reduces the similarity search space with Flood MBR-Sub and sub-trajectory table in sub-trajectory level pruning into candidate set $\mathscr{R}_f$. The local space shifting restriction of the four distances is imposed by MatchTable Dynamic Programming (DP) during the sub-trajectory-level pruning process. Subsequently, X-FIST refines the set of candidate trajectories as the final answer via computation of the distance between the trajectory query and the refined candidate set $\mathscr{R}_f$.

Before explaining the steps in detail, we introduce pruning techniques that depend on several spatial relationships between MBRs in X-FIST.

5.1. MBR-based pruning with MIN and MAX points

X-FIST prunes the dissimilar trajectories based on the indexed MBR trajectories in dataset $\mathscr{T}$. First, we define an extended MBR in Definition 5 and certain spatial relationships of MBRs in Definitions 6–8. Consequently, we propose a dissimilarity relationship based on a specific distance function among trajectories in Lemma 1. Subsequently, the details of the pruning technique using X-FIST are described. In addition, we also describe the retrieval of similar trajectories candidate without performing "scan" step in the original Flood range query using X-FIST.

Definition 5. τ-extended-MBR $MBR_\tau(P)$ is the MBR of trajectory P where the MBR is extended using τ in every dimension such that $MBR_\tau(P) = \{p_{min-\tau}, p_{max+\tau}\}$ where $p_{min-\tau} = \left\{p_{min_j} - \tau, 1 \leqslant j \leqslant n\right\}$ and $p_{max-\alpha} = \left\{p_{max_j} + \tau, 1 \leqslant j \leqslant n\right\}$. τ is interchangeable with ϵ and η according to the trajectory distance.

Definition 6. An MBR of Q $MBR(Q) = \{q_{min}, q_{max}\}$ covers another MBR of P $MBR(P) = \{p_{min}, p_{max}\}$ such that in every dimension $q_{min_i} \leqslant p_{min_i}$ and $q_{max_i} \geqslant p_{max_i}, 1 \leqslant i \leqslant n$.

Definition 7. An MBR $MBR(Q)$ strictly covers another MBR $MBR(P)$ within τ if and only if $MBR_\tau(Q)$ covers $MBR(P)$ and $MBR_\tau(P)$ covers $MBR(Q)$. Then, $\max_{1 \leqslant i \leqslant n} |q_{min_i} - p_{min_i}|$ and $1 \leqslant i \leqslant n |q_{max_i} - p_{max_i}|$ are always less than τ.

Definition 8. An MBR $MBR(Q)$ intersects another MBR $MBR(P)$ within ϵ if $MBR_\epsilon(Q) \cap MBR(P) \neq \varnothing$.

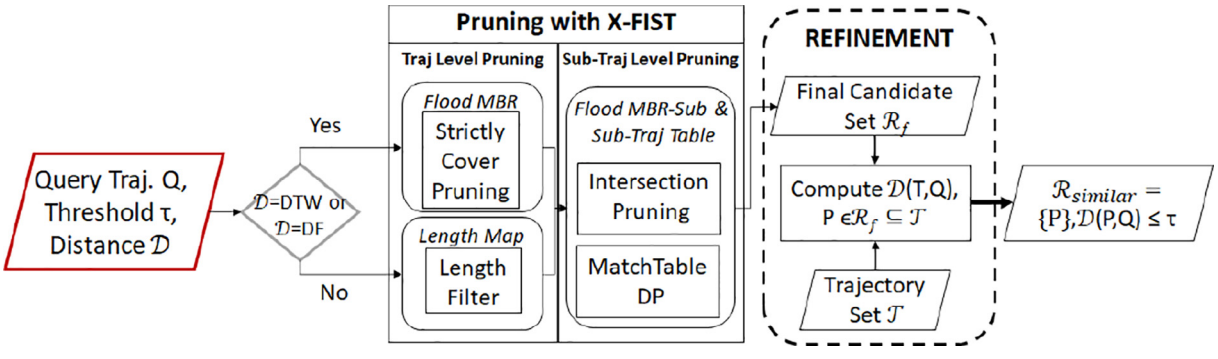


Fig. 2. The overview of similarity search with X-FIST.

Consider two trajectories P and Q and a threshold τ for DF and DTW and ϵ for EDR and LCSS, respectively. We provide the dissimilarity characteristics of each distance function $\mathscr{D}(P,Q)$ described in Section 3.2 in Lemma 1 and 2. Then, we demonstrate the correctness of each lemma using proof by contradiction.

Lemma 1. *In DF and DTW, P and Q cannot be similar if MBR(Q) does not strictly cover MBR(P) within* τ.

Proof by contradiction. Suppose that MBR of P and Q do not strictly cover each other by τ. Then, there exists at least a point $p_i \in P$ or $q_j \in Q$ such that $|q_{min_k} - p_{min_k}| > \tau$ or $|q_{max_k} - p_{max_k}| > \tau, 1 \leqslant k \leqslant n$. Hence, at least, in dimension k, the difference of a point towards every point in opposing trajectory in that attribute $|p_{i_k} - q_{a_k}|, q_a \in Q$ or $|q_{j_k} - p_{b_k}|, p_b \in P$ are always $> \tau$. Such difference contributes to any Euclidean distance to that point which becomes larger than τ. Thus, $DF(P,Q)$ and $DTW(P,Q)$ cannot be similar by τ.

Lemma 2. *Whereas, when MBR(Q) does not intersect MBR(P) within* ϵ*, in EDR T and Q cannot be similar by* $\tau, \tau \leqslant max(|P|,|Q|)$ *whereas in LCSS P and Q cannot be similar by* $\tau, \tau \leqslant |P| + |Q|$.

Proof by contradiction. Suppose that MBR of P does not intersect MBR of Q within ϵ. Hence, every point $p_i \in P$ cannot reach any point $q_j \in Q$ by ϵ in every attribute within ϵ thus $d(p_i, q_j) > \epsilon$. In EDR, to fulfill the condition $d(p_i, q_j) \leqslant \epsilon$, all points in P should be changed to match every point in Q. If $|P| > |Q|, |P| - |Q|$ points needs to be deleted whereas if $|P| < |Q|$ $|P| - |Q|$ points needs to be inserted. Hence, $max(|P|,|Q|)$ points need to be edited. In LCSS, no common subsequence occurs to match the condition of ϵ and δ. Hence, the dissimilarity becomes $|P| + |Q|$ according to Eq. 4.

The dissimilarity by $max(|P|,|Q|)$ in EDR and $|P| + |Q|$ in LCSS require a whole trajectory transformation (full edit or no common point). Thus, if the purpose the similarity metric is to find a similar pattern, such dissimilarity is undesired.

Example 1. In Fig. 3(a), consider using distance DF or DTW and extending $MBR(Q)$ with τ. Here, $MBR(Q)$ does not strictly cover the MBR of $P^{(3)}, P^{(4)}$, and $P^{(5)}$ within τ. In $P^{(3)}, P^{(4)}$, and $P^{(5)}$, there always exists at least one point that cannot reach any point in Q within τ. Thus, considering the proof by contradiction of Lemma 1, $\left\{P^{(3)}, P^{(4)}, P^{(5)}\right\}$ cannot be similar to Q in DF and DTW. However, consider using distance EDR or LCSS and extending $MBR(Q)$ with ϵ. Then, $MBR(Q)$ does not intersect the MBR of $P^{(4)}$ and $P^{(5)}$ within ϵ. Herein, no point in $P^{(4)}$, and $P^{(5)}$ can reach any point of Q within ϵ. Thus, considering the proof by contradiction of Lemma 2, $\left\{P^{(4)}, P^{(5)}\right\}$ cannot be similar to Q in the EDR and LCSS by τ equal to 5 and 9, respectively.

Consequently, X-FIST can prune dissimilar trajectories employing strictly cover pruning and intersection pruning. The pruning techniques provide different boundaries of the employed Flood-based range queries without scan according to the point type as summarized in Table 2. We discuss the provided boundaries as follows.

5.1.1. Strictly cover pruning with Trajectory MBR

The "strictly covers" relationship (Definition 7) cannot occur if, at any dimension $j, 1 \leqslant j \leqslant n, |q_{min_j} - p_{min_j}| > \tau$ or $|q_{max_j} - p_{max_j}| > \tau, MBR(P) = \{p_{min}, p_{max}\}, MBR(Q) = \{q_{min}, q_{max}\}$. Thus, the use q_{min} and q_{max} with an extension τ can be considered as the range query to their respective indexed MIN and MAX points, respectively. Then, a set of *tid* or *sid* associated with each key that exists in two range query results is retrieved against Flood MBR-MIN and Flood MBR-MAX with the bound $[q_{min-\tau}, q_{min+\tau}]$ and $[q_{max-\tau}, q_{max+\tau}]$, respectively. For example, we return $\left\{P^{(1)}\right\}$ as the intersection of the result sets of $\mathscr{R}_{qmin}$ and $\mathscr{R}_{qmax}$ as the strictly cover result with Flood in Fig. 3(b).

5.1.2. Intersection pruning with Sub-Trajectory MBR

The "intersection" relationship (Definition 8) cannot occur if, at any dimension $j, 1 \leqslant j \leqslant n, q_{min_j} - p_{max_j} > \epsilon$ and $p_{min_j} - q_{max_j} > \epsilon, MBR(P) = \{p_{min}, p_{max}\}, MBR(Q) = \{q_{min}, q_{max}\}$. In contrast to "strictly cover" pruning, the range query bound is defined towards Flood MBR-MIN and Flood MBR-MAX to $[-\infty, q_{max+\epsilon}]$ and $[q_{min-\epsilon}, \infty]$, respectively.

For the case of sub-trajectory MBR, we can bound the range query toward Flood MBR-Sub-MIN and Flood MBR-Sub-MAX to $\left[sq_{min-\epsilon-\eta}, sq_{max+\epsilon}\right]$ and $\left[sq_{min-\epsilon}, sq_{max+\epsilon+\eta}\right]$, respectively, to reduce the search space, given the extended MBR $MBR_\epsilon(Q_{a-b}) = \{sq_{min-\epsilon}, sq_{max+\epsilon}\}, Q_{a-b} \in SQ = Split(Q)$. The restrictive bound by η can happen due to splitting parameter η as described in the indexing in Section 4 such that maximum range of the MBR $\max_{1 \leqslant k \leqslant n} |p_{min,k} - p_{max,k}|, MBR(P_{a-b}) = \{p_{min}, p_{max}\}$ of every sub-trajectories $P_{a-b} \subseteq P \in \mathscr{T}$ cannot exceed η. Thus, extending the bound by η from ϵ can always contain every MBR of the sub-trajectory that intersects an MBR of $Q_{a-b} \in SQ = Split(Q)$.

For example, we index the MIN and MAX points of sub-trajectory MBRs and perform intersection with Q_{3-4} as depicted in Fig. 3(c). Only sub-trajectory MBR with id 8 of $P^{(2)}$ is considered as the intersecting MBR to Q_{3-4} of Q. Although MBRs with id $3, 9, 10, 11, 12$ are inside the boundary of range query towards indexed MAX points, they do not exist in the result of range query towards indexed MIN points. Then, in the match table DP step, we only consider a result $P^{(2)}$ in pruning with Q_{3-4}.

However, in both strictly cover and intersection pruning, "scan" process is not necessary. The pruning without the "scan" step can be considered as the lower bound of the candidate trajectory set. Owing to the set intersection of the results of two

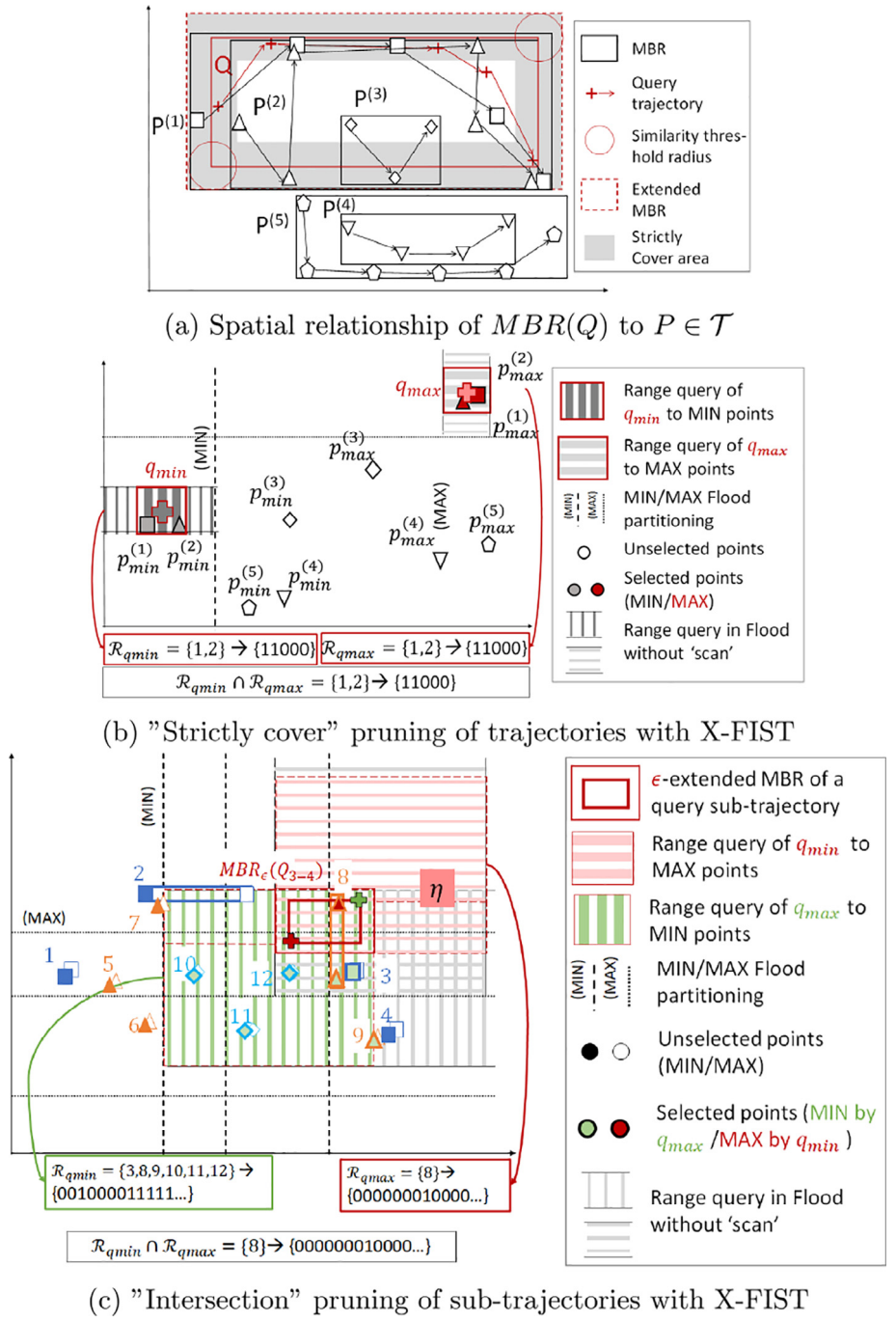


Fig. 3. Performing pruning techniques with X-FIST.

range queries, X-FIST can minimize irrelevant points without examining the actual points in the dataset. In addition, X-FIST employs bitset operations for quicker set intersections as illustrated in Fig. 3(b) and (c). The pruning of irrelevant trajectories can be maximized by repeating the set intersection with sub-trajectories later in sub-trajectory-level pruning.

5.2. Trajectory-level pruning

As depicted before in Fig. 2, we perform strict cover pruning with Flood MBR if the distance is DF or DTW and length filtering with length map if the distance is EDR or LCSS, which described in Algorithm 4. We put the *tid* of similar trajectory

candidates inside the set $\mathscr{R}$. First, we consider the case if the distance is DF or DTW. Flood MBR of X-FIST acquires the *tid* of similar trajectory candidates that strictly cover $MBR(Q)$ within offset τ (line 4), as described in Section 5.1.1. In contrast, for EDR and LCSS, the length map of X-FIST selects *tid* of trajectories whose length is $[|Q| - \tau, |Q| + \tau]$ (line 6–7). Then, $\mathscr{R}$ serves as the input in sub-trajectory-level pruning.

Algorithm 4: Trajectory Level Pruning with X-FIST

Input : A query trajectory Q, X-FIST $xfist$, distance function $\mathcal{D}$, threshold τ
Output: A similar trajectory candidate set $\mathcal{R}$
1 **Func** ***TrajLvlPrune*** $(Q, xfist, \mathcal{D}, \tau)$
2 Init $\mathcal{R}$ as empty set $\emptyset$;
3 **if** $\mathcal{D} \in \{DF, DTW\}$ **then**
4 $\mathcal{R} \leftarrow strictlyCover(MBR(Q), xfist.getFloodMbr(), \tau)$;
5 **else**
6 Set $l^- \leftarrow max(1, |Q| - \tau)$, $l^+ \leftarrow |Q| + \tau$;
7 **for** $i \leftarrow l^-$ **to** l^+ **do** $\mathcal{R}$.addAll($xfist.lengthMap$.get(i)) ;
8 return $\mathcal{R}$;

5.3. Sub-trajectory-level pruning

Sub-trajectory level pruning increases the precision of the pruning of X-FIST from the trajectory level pruning. We describe the idea of the dissimilarity of the sub-trajectory in Lemma 3, followed by intersection pruning and dynamic programming, which returns $\mathscr{R}_f$ as the final candidate set of trajectories.

Lemma 3. *For the sub-trajectories of P and Q, $SP = \{P_{a_i-b_i}, 1 \leqslant i \leqslant |SP|\}$ and $SQ = \left\{Q_{a_j-b_j}, 1 \leqslant j \leqslant |SQ|\right\}$, where SP and SQ are sequences of sub-trajectories of P and Q split by η, respectively, and thresholds τ and parameter ϵ. P cannot be similar to Q for DF and DTW when an MBR $MBR\left(Q_{a_j-b_j}\right)$ does not intersect any MBR of $P_{a_i-b_i}$ in SP within τ. Whereas, P cannot be similar to Q for EDR and LCSS when every $MBR\left(Q_{a_j-b_j}\right)$ in SQ does not intersect with any MBR of $P_{a_i-b_i}$ in ST within ϵ when $\tau \leqslant max(|P|, |Q|), \mathscr{D} = EDR$ or $\tau \leqslant |P| + |Q|, \mathscr{D} = EDR$.*

The correctness of Lemma 3 is demonstrated in Example 2.

Example 2. In Fig. 4, the query trajectory Q is split into $\{Q_{1-1}, Q_{2-2}, Q_{3-4}, Q_{5-5}\}$. $P^{(2)}$ and Q are dissimilar in DF and DTW as q_1 cannot reach any point in $P^{(2)}$ within τ. Assuming that we extend the MBR of Q with τ, $MBR(Q_{1-1})$, which contains q_1, does not intersect any $MBR\left(P^{(2)}_{a-b}\right)$ of $P^{(2)}$ within τ, which has *sid* of 5, 6, 7, 8, and 9. Thus, X-FIST should output $\mathscr{R}_f$ without $P^{(2)}$.

In contrast, $P^{(3)}$ and Q are dissimilar by $max(|P|, |Q|)$ for EDR and by $|P| + |Q|$ for LCSS because no point in $P^{(3)}$ can reach any point in Q within ϵ. Assuming that we extend the MBR of Q with ϵ, every $MBR(Q_{a-b})$ of Q does not intersect any MBR of the sub-trajectory of $P^{(3)}$ within ϵ, which has *sid* 10,11, and 12. Thus, X-FIST should output $\mathscr{R}_f$ without $P^{(3)}$ when $\tau \leqslant 5$ for EDR and $\tau \leqslant 8$ for LCSS.

5.3.1. Sub-trajectory MBR intersection pruning

We describe the sub-trajectory MBR pruning in Algorithm 5 as follows. First, Q is split into a set of $|SQ|$ sub-trajectories $SQ = \left\{Q_{a_j-b_j}, 1 \leqslant j \leqslant |SQ|\right\}$ with parameter η (line 1).

Next, we initialize a match table of whose value is only 0 and 1 *MTable* (line 3) with $|SQ|$ rows and the length of sub-trajectory table columns $|STable|$, with cell content equal 0. Note that we employ a special intersection when $Q_{a_j-b_j}$ is the first or last sub-trajectory of Q (line 6) if the distance is DF and DTW as described in Section 3.2 and [20]. The j-th row of match table represents the intersection pruning result with Flood MBR-Sub *FloodMS* and each sub-trajectory $Q_{a_j-b_j}$ in SQ (line 7) and we assume the $\epsilon = \tau$ when the distance is DF or DTW. For LCSS distance, we define additional pruning rule with the spacing constraint δ.

Algorithm 5: Sub-Trajectory Level Prune with X-FIST

Input : A query trajectory Q, distance function $\mathcal{D}$,threshold τ, ϵ, δ, split parameter η, Flood MBR-Sub $FloodMS$, Sub-trajectory table $STable$, candidate set of tids $\mathcal{R}$

Output: Final similar trajectory candidate set $\mathcal{R}_f$

Func ***SubTrajLvlPrune*** $(Q, \mathcal{D}, \tau, \epsilon, \delta, \eta, FloodMS, table, \mathcal{R})$

$SQ = Split(Q, \eta)$;
Init match table $MTable$ with $|SQ|$ rows $\times$ $|STable|$ columns;
foreach $Q_{a_j\text{-}b_j} \in SQ$ **do**
  if $\mathcal{D} \in \{DF, DTW\}$ *and* $Q_{a_j\text{-}b_j}$ *is first or last sub-traj.* **then** $\mathcal{R}_i = DfDtwIntersect(Q_{a_j\text{-}b_j}, \tau, FloodMS, STable, \eta, \mathcal{D})$;
  else $\mathcal{R}_i = intersect(MBR(Q_{a_j\text{-}b_j}), FloodMS, \epsilon, \eta)$;
  if $\mathcal{D}$ *is* $LCSS$ **then** LcssPrune$(\mathcal{R}_i, \{a_j, b_j\}, STable, \delta)$;
  foreach $k \in \mathcal{R}_i$ **do** $MTable[j, k] = 1$;
$\mathcal{R}f = MatchDP(SQ, MTable, \mathcal{D}, \tau, \mathcal{R}, STable)$;
return $\mathcal{R}f$;

Algorithm 6: Special intersection pruning for DF and DTW

Input : A query sub-trajectory $Q_{a\text{-}b}$, threshold τ, Flood MBR-Sub $FloodMS$, Sub-trajectory table $STable$, η, distance function $\mathcal{D}$

Output: An improved intersection set $\mathcal{R}_i$

Func ***DfDtwIntersect*** $(Q_{a\text{-}b}, \tau, FloodMS, R_i, STable, \eta, \mathcal{D})$

If global var. $DMap$ is NULL, initialize $DMap$ as empty map;
$\{\mathcal{R}_{min}, \mathcal{R}_{max}\} \leftarrow$ intersection of $MBR_\tau(Q_{a\text{-}b})$ towards $FloodMS$;
Set $\{d_{min}, d_{max}\}$ as the sorted dim. of $\{FloodMS.Min, FloodMS.Max\}$;
Put common sids of $\mathcal{R}_{min}$ and $\mathcal{R}_{max}$ into $\mathcal{R}_i$;
foreach $sid \in \mathcal{R}_i, r_1 \in \mathcal{R}_{min}, r_2 \in \mathcal{R}_{max}, sid = r_1.id = r_2.id$ **do**
  Get info of $P^{(j)}_{a\text{-}b}$ from sid-th entry of $STable$;
  if $P^{(j)}_{a_j\text{-}b_j}$ *not first or last sub-traj.* **then continue**;
  $dist = \sqrt{(q_{min,d_{min}} - r_1.key)^2 + (q_{max,d_{max}} - r_2.key)^2}$;
  if $dist > \tau$ **then** remove sid from $\mathcal{R}_i$, **continue**;
  if $P^{(j)}_{a\text{-}b}$ *and* $Q_{a\text{-}b}$ *are first sub-traj.* **then** Put $\langle j, dist\rangle$ into $DMap$;
  else if $P^{(j)}_{a\text{-}b}$ *and* $Q_{a\text{-}b}$ *are last sub-traj.* **then** remove sid from $\mathcal{R}_i$ if $DMap.get(j)$ empty or $dist + DMap.get(j) > \tau$;
return $\mathcal{R}_i$;

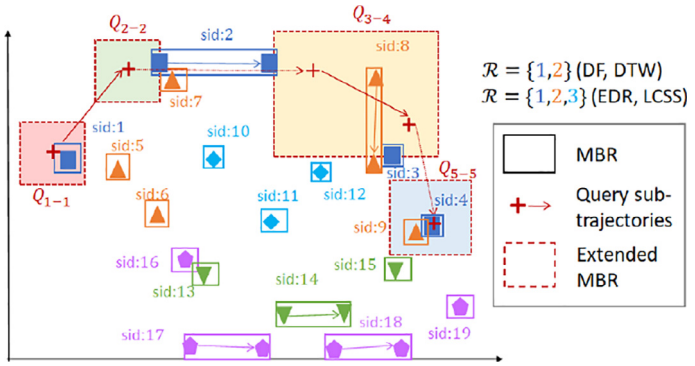


Fig. 4. Intersection of the sub-trajectories of Q with the MBRs of the sub-trajectories of $P \in \mathscr{T}$.

DTW always prunes a trajectory $P^{(i)}$ when $d\left(p_1^{(i)}, q_1\right) + d\left(p_{|P|}^{(i)}, q_{|Q|}\right) > \tau$ (first and last point dissimilarity) whereas DF uses $d\left(p_1^{(i)}, q_1\right) > \tau$or $d\left(p_{|P|}^{(i)}, q_{|Q|}\right) > \tau$, $P^{(i)} = \left\langle p_1^{(i)}, .., p_{|P|}^{(i)} \right\rangle$. However, computing the exact Euclidean distance requires to "scan" relevant points in the dataset, whose process is eliminated in X-FIST. We can approximate the Euclidean distance with the key difference across different sorted dimensions of the Flood Sub-MBR-MIN and Flood Sub-MBR-MAX. We estimate such distance and use it for pruning in Algorithm 6.

The first step of the special intersection in Algorithm 6 is initializing an empty map *DMap* that we will use for the first and last points sum dissimilarity rule of DTW distance (line 2). *DMap* has the ⟨ *key, value* ⟩ format of $\langle tid, dist \rangle$ where *tid* identifies a candidate trajectory and *dist* is its associated estimated distance to Q. If we encounter Q_{a-b} as first sub-trajectory the map will be initialized and used later when Q_{a-b} is the last sub-trajectory. Then, we execute an intersection with the MBR $MBR(Q_{a-b})$ towards Flood MBR-Sub (line 3), but we keep the result of each range query of Flood MBR-Sub-MIN and Flood MBR-Sub-MAX separately in $\mathscr{R}_{\min}$ and $\mathscr{R}_{\max}$, respectively, and use (id, key) as the result entry format (line 3). Then, by iterating the common result of acquired *sid* (line 5), we perform the distance estimation (line 8). However, we do not remove any *sid* if the acquired sub-trajectory is neither first nor last sub-trajectory (line 7). If the estimated distance is larger than τ, we omit the *sid*. Then, in line 11, we apply the dissimilarity rule of DTW by storing the estimated distance between first points in *DMap*. The pruning of *sid* due to dissimilarity rule violation is denoted by line 12 if (1) the first sub-trajectory is larger than τ (*DMap* of corresponding trajectory is empty) or (2) the sum of estimated distances is larger than τ.

Algorithm 7: Sub-Trajectory Pruning in LCSS with δ

Input : Sub-trajectory intersection result $\mathcal{R}_i$, First and last index of sub-trajectory $Q_{a_j\text{-}b_j}\{a_j, b_j\}$, sub-traj table $STable$, δ
Output: A refined intersection set $\mathcal{R}_i$

```
1 Func LcssSubTrajPrune (R_i, {a_j, b_j}, STable, δ)
2     foreach sid ∈ R_i do
3         b_i ← STable[sid].getLastIndex();
4         a_i = b_i − STable[sid].getLen();
5         if a_i − b_j > δ ∨ a_j − b_i > δ then R_i.remove(sid);
6     return R_i;
```

As described in Eq. 4, LCSS restricts the matching between points p_i and $q_j, p_i \in P, q_j \in Q$ such that $|i-j| \leqslant \delta$[36]. Thus, this restriction is extended to the sub-trajectory level. Given two sub-trajectories $P_{a_i-b_i}$ and $Q_{a_j-b_j}$. a_i, a_j are the start indices and b_i, b_j are the end indices of the two sub-trajectories $P_{a_i-b_i}, Q_{a_j-b_j}$ respectively. Any point in $P_{a_i-b_i}$ cannot match every point in $Q_{a_j-b_j}$ when $a_i - \delta > b_j$ or $a_j - \delta > b_i$. However, the resulting set of intersection pruning $\mathscr{R}_i$ (Algorithm 5 line 7) only contains the retrieved *sid* but the explicit start and end index of the retrieved sub-trajectories are not included. Thus, X-FIST removes the non-matching sub-trajectories in $\mathscr{R}_i$ using Algorithm 7 where sub-trajectory table *STable* is included as input. The sub-trajectory table easily retrieves the last index and the length of a sub-trajectory as the *sid*-th entry as previously described in subSection 4.2. Subsequently, X-FIST removes the *sid* from $\mathscr{R}_i$ in line 5 and returns $\mathscr{R}_i$ as result.

After performing the intersection between the MBRs, the final candidate set of TIDs $\mathscr{R}_f$ is returned using MatchTable DP, an iterative DP pruning technique with a match table.

5.3.2. Match table dynamic programming

The Match Table DP technique imposes the local shifting of the trajectories according to the sub-trajectory table ordering of the sub-trajectories. Although the sub-trajectories of Q and $P \in \mathscr{T}$ intersect completely, the four distances still require similar trajectories to allow movement in the same direction. However, if a trajectory similarity measure disregards the ordering of the points (such as Hausdorff), X-FIST can output the candidate set directly skipping this step. We illustrate the steps of the match table DP with Algorithm 8 and Fig. 5.

Algorithm 8: Pruning with an iterative DP

Input : Set of query subtrajectories SQ, match table $MTable$, $\mathcal{D}$, τ, candidate set of tids $\mathcal{R}_f$, sub-trajectory table $STable$
Output: A refined candidate set $\mathcal{R}_f$
1 **Func** ***MatchDP*** $(SQ, MTable, \mathcal{D}, \tau, \mathcal{R}_f, STable)$
2 **if** $\mathcal{D} \in \{DF, DTW\}$ **then** return traverse$(SQ, MTable, \mathcal{R}_f, STable)$;
3 Init distance table $dTable$ with the shape of $MTable$;
4 Update $dTable$ cells $[i, c]$ of corresponding sub-trajectories in $\mathcal{R}_f$ by FillDistanceCell$(Q_{a_i - b_i}, \mathcal{D}, dTable, i, c)$);
5 **if** *Estimated distance of* $T^{(j)} > \tau$ **then** remove j from $\mathcal{R}_f$;
6 return $\mathcal{R}_f$;

Suppose that we have intersection result as depicted in Fig. 5 (a). For DF and DTW, we traverse the values in the match table and return the similar candidate set $\mathscr{R}_f$ in three steps as depicted in Fig. 5. For each candidate trajectories, if (1) the cell content of intersection of first sub-trajectories or (2) the last trajectories of Q and $P^{(i)}$ in the match table *MTable* are 0, we immediately prune the trajectory. Then, (3) we check the down, right, and down-right neighboring cells from cell (1). If the cell content is 1, we can move to that cell and repeat the neighboring cell checking until we reach cell (2). The traversal can be easily achieved by DP, depth-first search, or breadth-first search.

Example 3. Fig. 5 (b) shows $\mathscr{R} = \left\{P^{(1)}, P^{(2)}\right\}$ as the similar candidate trajectories to a query trajectory Q. First, we check the contents of cells that represent the intersections of the first and last sub-trajectories of Q to the indexed trajectory in match table cell [1,1,1,5,4,4,4,9]. Because Q_{1-1} and $P^{(2)}_{1-1}$ does not intersect (cell [1,5] = 0), $P^{(2)}$ is removed from the candidate set. Moreover, we can traverse from cell [1,1] to [4,4] by moving in all down-right directions. Thus, we return $\left\{P^{(1)}\right\}$ as final candidate set $\mathscr{R}_f$.

Then, for EDR and LCSS, we need to fill out the estimate distance table *dTable* (Algorithm 8, line 3). Each cell [i,c] of the estimate distance table corresponds to the lower bound of EDR or LCSS distance of Q_{1-b_i} to a P_{1-b_j} where c is the *sid* of an indexed sub-trajectory $P_{a_j - b_j}$. Hence, the cell [i',c'] where $i\prime = |SQ|$ and $c\prime$ is *sid* of last sub-trajectory of a trajectory $P^{(k)}, kin\mathscr{R}$ holds the estimate distance of Q and $P^{(k)}$. We can prune $P^{(k)}$ from the candidate set when the last estimated distance is larger than τ.

Algorithm 9: Filling a cell of estimate distance table

Input : Query sub-trajectory $Q_{a_i\text{-}b_i}$, distance function $\mathcal{D}$, distance table $dTable$, indices i, c

1 **Procedure** ***FillDistanceCell*** $(Q_{a_i\text{-}b_i}, \mathcal{D}, dTable, i, c)$

2 Get $P^{(k)}_{a_j\text{-}b_j}$ info from c-th entry of $STable$;

// $I_m(.)$ returns 1 if MBRs intersect and 0 otherwise

3 **if** $i = 1 \wedge j = 1$ **then** $dTable[i, c] = 1 - I_m(Q_{a_i\text{-}b_i}, P^{(k)}_{a_j\text{-}b_j})$, **return**;

// Adding previous distances and conditional costs

4 $dU = dU + dTable[i - 1, c] + cost_{\mathcal{D}}(I_m(Q_{a_i\text{-}b_i}, P^{(k)}_{a_j\text{-}b_j}), |P^{(k)}_{a_j\text{-}b_j}|)$;

5 $dL = dL + dTable[i, c - 1] + cost_{\mathcal{D}}(I_m(Q_{a_i\text{-}b_i}, P^{(k)}_{a_j\text{-}b_j}), |Q_{a_i\text{-}b_i}|)$;

6 $dUL = dUL + dTable[i - 1, c - 1] + 1 - I_m(Q_{a_i\text{-}b_i}, P^{(k)}_{a_j\text{-}b_j})$;

7 **if** $I'_m(Q_{a_i\text{-}b_i}, P^{(k)}_{a_j\text{-}b_j}) \wedge \mathcal{D} = LCSS$ **then** $dUL = \infty$;

// Add previous sub-trajectories length if they don't match

8 $dU = (1 - I_m(Q_{a_{i-1}\text{-}b_{i-1}}, P^{(k)}_{a_j\text{-}b_j}))*(|Q_{a_{i-1}\text{-}b_{i-1}}| - 1)$;

9 $dL = (1 - I_m(Q_{a_i\text{-}b_i}, P^{(k)}_{a_{j-1}\text{-}b_{j-1}}))*(|P^{(k)}_{a_{j-1}\text{-}b_{j-1}}| - 1)$;

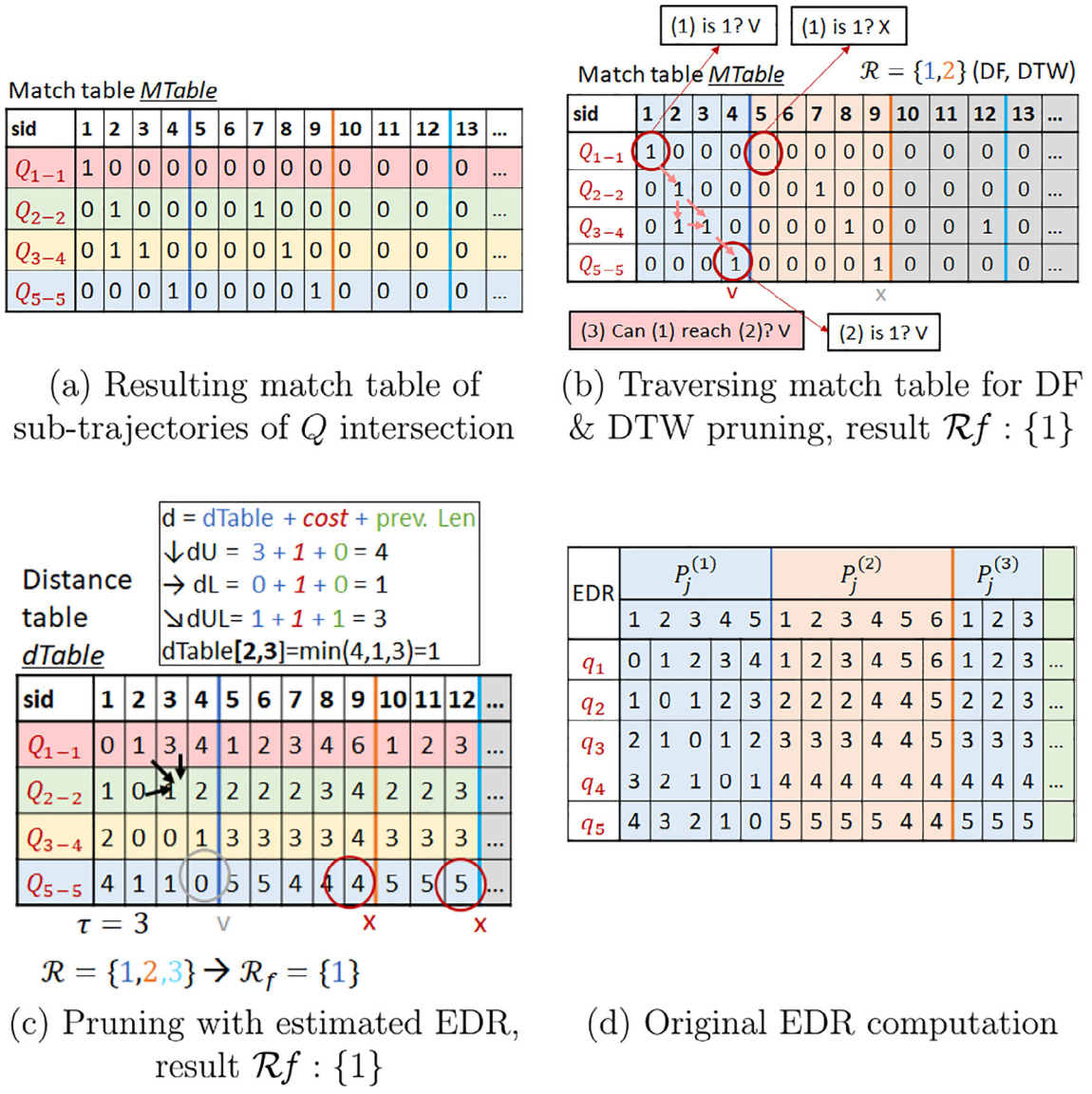
10 $dUL = (1 - I_m(Q_{a_{i-1}\text{-}b_{i-1}}, P^{(k)}_{a_{j-1}\text{-}b_{j-1}}))*(max(|Q_{a_{i-1}\text{-}b_{i-1}}|, |P^{(k)}_{a_{j-1}\text{-}b_{j-1}}|) - 1)$;

11 $dTable[i, c] = min(\{dUp, dLeft, dUL\})$;

Algorithm 9 fills a cell [i,c] in *dTable* that closely follows the steps from the original formulae of EDR and LCSS in eqs. 3 and 4. Notation $I_m\left(Q_{a_i-b_i}, P_{a_j-b_j}\right)$ denotes whether the MBR of $Q_{a_i-b_i}$ and $P_{a_j-b_j}$ intersect each other and corresponds to value at cell $[i, c]$ of match table *MTable*, or in a sense to the original formulae, $Q_{a_i-b_i}$ matches to $P_{a_j-b_j}$. Computing value of distance table *dTable* at cell $[i, c]$ follows the dynamic programming principle. Computing the distance at cell $[i, c]$ requires the values of previous cells at $[i-1, c]$ (upper), $[i, c-1]$ (left), $[i-1, c-1]$ (upper-left) with some modifications, which are denoted by dU, dL, and dUL, respectively (line 4–6). We ignore the value of dU and dUL (set to ∞) if the current examined row is $i = 1$ which means no upper cell whereas we ignore the value of dL and dUL if the previous column ($sid - 1$) belongs to another trajectory or inexists. Note that we do not require such step for the first sub-trajectories (line 3). Eq. 5 denotes the cost based on the distance function and match of $Q_{a_i-b_i}$ and $P_{a_j-b_j}$. In LCSS, we ignore the upper left distance dUL (line 7) if current sub-trajectories do not match, similar to the original formula. Then, we consider the case of the previous sub-trajectories $Q_{a_{i-1}-b_{i-1}}$ and $P_{a_{j-1}-b_{i-1}}$ that do not match $Q_{a_i-b_i}$ or $P_{a_j-b_i}$ (line 8–10). Such non-matches add the length of previous trajectories to corresponding estimated distance, which can improve the pruning.

$$cost_{\mathscr{D}}(match, len) = \begin{cases} 1, & match = 0 \\ 0, & len > 1 \\ \infty, & \mathscr{D} = LCSS \\ 1 & otherwise \end{cases} \tag{5}$$

Example 4. Fig. 5(c) describes the filling of the estimated distance table for EDR between $\{Q_{1-1}, Q_{2-2}, Q_{3-4}, Q_{5-5}\}$ with the indexed sub-trajectories in Flood MBR-Sub. The estimation of LCSS distance is omitted because LCSS behave in a similar manner to EDR but with different cost estimation in Eq. 5. Thereafter, following the steps described in Algorithm 9, the final



sid	1	2	3	4	5	6	7	8	9	10	11	12	13	...
Q_{1-1}	1	0	0	0	0	0	0	0	0	0	0	0	0	...
Q_{2-2}	0	1	0	0	0	0	1	0	0	0	0	0	0	...
Q_{3-4}	0	1	1	0	0	0	0	1	0	0	0	0	0	...
Q_{5-5}	0	0	0	1	0	0	0	0	1	0	0	0	0	...

sid	1	2	3	4	5	6	7	8	9	10	11	12	13	...
Q_{1-1}	1	0	0	0	0	0	0	0	0	0	0	0	0	...
Q_{2-2}	0	1	0	0	0	0	1	0	0	0	0	0	0	...
Q_{3-4}	0	1	1	0	0	0	0	1	0	0	0	1	0	...
Q_{5-5}	0	0	0	1	0	0	0	0	1	0	0	0	0	...

sid	1	2	3	4	5	6	7	8	9	10	11	12	...
Q_{1-1}	0	1	3	4	1	2	3	4	6	1	2	3	...
Q_{2-2}	1	0	1	2	2	2	2	3	4	2	2	3	...
Q_{3-4}	2	0	0	1	3	3	3	3	4	3	3	3	...
Q_{5-5}	4	1	1	0	5	5	4	4	4	5	5	5	...

EDR	$P_j^{(1)}$					$P_j^{(2)}$						$P_j^{(3)}$			
	1	2	3	4	5	1	2	3	4	5	6	1	2	3	
q_1	0	1	2	3	4	1	2	3	4	5	6	1	2	3	...
q_2	1	0	1	2	3	2	2	2	4	4	5	2	2	3	...
q_3	2	1	0	1	2	3	3	3	4	4	5	3	3	3	...
q_4	3	2	1	0	1	4	4	4	4	4	4	4	4	4	...
q_5	4	3	2	1	0	5	5	5	5	4	4	5	5	5	

Fig. 5. Performing matching and distance estimation of sub-trajectories of Q.

cell of trajectories with id $\{1,2,3\}$ have value of $\{0,4,5\}$. Thus, we can reject $T^{(2)}$ and $T^{(3)}$ to be in $\mathscr{R}f$, as shown in Fig. 5(c). If we compare the estimate distance to the original EDR in Fig. 5 (d), the estimation is always equal or slightly smaller than original distance of the correspoding points. Thus, we have a good lower bound to pruning irrelevant trajectories.

Finally, the significantly reduced set of similar trajectory candidates $\mathscr{R}_f$ containing the trajectory id i is obtained. Thus, the actual trajectory distance is computed and trajectories with distance $> \tau$ with $\mathscr{D}\left(P^{(i)}, Q\right), P^{(i)} \in \mathscr{T} \wedge i \in \mathscr{R}$ were removed.

5.4. Similarity search cost

We describe the execution cost of X-FIST as follows: The trajectory-level pruning includes pruning with Flood MBR or length filtering. The Flood pruning of X-FIST, which follows strictly cover, consists of two range queries. Consider one MBR range query costing c_{RQ}, the length map filtering costing c_f, and set intersection bitset operation costing c_b. Hence, trajectory-level pruning costs $2c_{RQ} + c_b$ (DF and DTW) or c_f (EDR and LCSS). Further, the sub-trajectory-level pruning includes the intersection sub-trajectory MBRs of Q that also requires two range queries. Suppose that the average number of split sub-trajectories is s and computation of a single DP table costs c_{DP} of intermediate candidate set $\mathscr{R}$. Then, the sub-trajectory-level pruning would cost $s(2c_{RQ} + c_b) + s^2(|\mathscr{R}|)(c_{DP})$. The final step requires trajectory distance computation over $\mathscr{R}_f$. If the trajectory distance computation costs $c_{\mathscr{D}}$, then the refinement phase would cost $|\mathscr{R}_f| * c_{\mathscr{D}}$.

In conclusion, the overall execution time of X-FIST has $O(s^2)$ pruning time and $O\left(|T|^2\right)$ refinement time; however, it holds for highly reduced $\mathscr{R}_f$ rather than the whole set $\mathscr{T}$.

6. Experimental evaluation

6.1. Setup

We conducted the experiments on a Ubuntu 18.04 64-bit machine with 12-core Intel (R) i7-8700 CPU @3.20 Hz CPU, 32 GB RAM, and 1 TB hard disk. All the learned indices based on CPU using Java were implemented from scratch.

6.1.1. Dataset

Table 3 describes the three real-world datasets employed for our experiments: Chengdu KDD CUP 2020,[1] Porto [5], and ATC [7]. Chengdu and Porto contain the vehicle 2D trajectories from the Chengdu and Porto cities, respectively, whereas the ATC dataset contains the indoor trajectory of visitors inside a shopping mall. Owing to the limited machine capability, we subset the Chengdu dataset to include only the trajectories of day 1–10. Whereas, the Porto dataset contains certain outliers thus the minimum and maximum values of the points in the dataset were limited. In contrast, we downsampled the ATC dataset by averaging the records within interval of 1 s. In addition, we normalized the values of the six attributes of the ATC dataset (X, Y, Z, speed, movement orientation, and facing angle) to 0–1. The datasets are highly skewed as the skewness [39]is more than 1. Furthermore, we generate new datasets with various skewness to explore the effectiveness of our approach in subsection 6.2.3.

6.1.2. Baseline methods

The performance of X-FIST was compared with that of the four different methods. The first method is the raw all-point Flood [14] (Flood). We indexed all the points $t_j \in T \in \mathscr{T}$ with its index j and TID i as the key. The second method is the an extension of ML-Index [19] with an architecture similar to X-FIST, denoted as XMLI, which can perform strictly cover and intersection pruning but with radial approximation with iDistance rather than rectangular range query. The third method is an extension of R-tree [11] that indexes the MBR points of the trajectories and sub-trajectories (implemented using Conversant[2] with necessary modification). Finally, the fourth method is Kd-tree [12] with similar extension to the X-FIST (implemented using JavaML[3] with necessary modification). Moreover, Flood and XMLI were implemented in JAVA with the error threshold parameter equal to X-FIST. The source code of XFIST is available in.[4]

Moreover, another R-tree-based index for multi-attribute trajectory data [21] may work as a baseline. However, applying [21] to our datasets and the similarity search require a data transformation to create the nominal multi-attributes, which eliminates the precision of the real values. The imprecise attributes can lead to inefficient range query. Omitting the nominal multi-attribute structure of [21]leaves only the R-tree [11].

Furthermore, the usage in the memory (index size), index building time, and query time (pruning and refinement) were utilized as the basis for the evaluations. All metrics are averaged over five experiment runs. The similarity search uses 1000 random query trajectories in each run.

6.1.3. Parameters

Table 4 shows parameters for the experiment with default values indicated in bold. The similarity threshold of each distance function was varied to determine the performance of our index over different thresholds. The threshold τ and matching parameter ϵ of ATC is highly different from Chengdu and Porto because ATC has more attributes and larger range. In addition, in Flood, X-FIST, and XMLI, the number of sorted array prediction error thresholds *err* was fixed. The parameters of the R-tree and Kd-tree were set with its default setting from the source code, except we set the maximum nodes contained in a block to *err* to match the search boundary of predicted location of the LI methods.

6.2. Result

6.2.1. Index building

Figs. 6 and 7 show the growth of the index size and building time of every index within various dataset sample size. Overall, X-FIST showed better performance compared to other approaches based on index building performance. Flood can only build the index with the sample size 0.2 in the Chengdu dataset due to out of memory error. Regarding the building time, X-FIST performed best across all datasets for each sample size, except in Porto, where Kd-Tree performed better. Thus, based on the index building time and memory usage, X-FIST demonstrated its scalability to handle big trajectory dataset.

The short index building time of X-FIST suggests small cost of rebuilding in a batch when data insertion or deletion occurs. However, [15,26,27] can update the index efficiently with some update policy using gapped arrays or tree-based data

[1] https://gaia.didichuxing.com.
[2] https://github.com/conversant/rtree.
[3] http://java-ml.sourceforge.net/api/0.1.7/net/sf/javaml/core/kdtree/KDTree.html.
[4] https://gitlab.com/dslab/XFIST.

Table 3
Details of evaluation dataset.

Dataset	**Chengdu**	**Porto**	**ATC**
Disk size (GB)	6.24	1.08	2.93
Attributes (n)	2	2	6
$\|\mathscr{T}\|$	2,163,572	1,704,769	1,666,753
Average $\|T\|$	163	50	33
Min value	$\{104.04, 30.65\}$	$\{-9.37, 39\}$	$\{0,0,0,0,0,0\}$
Max value	$\{104.13, 30.72\}$	$\{-5.67, 42.54\}$	$\{1,1,1,1,1,1\}$
Skewness	1.38	4.21	1.14

Table 4
Evaluation parameters of our experiment.

Parameter	Values
Sample size	$0.2, 0.4, 0.6, 0.8, \mathbf{1.0}$
Error threshold *err* (Flood)	50
τ (DF, DTW) [Chengdu,Porto]	$\{1, 2, \mathbf{3}, 4, 5\} \times 10^{-3}$
τ (DF, DTW) [ATC]	$\{1, 2, \mathbf{3}, 4, 5\} \times 10^{-2}$
τ (EDR, LCSS) [All]	$\{0, 1, 2, \mathbf{3}, 4, 5\}$
ϵ [Chengdu,Porto]	0.003
ϵ [ATC]	0.03
δ (LCSS)[All]	3

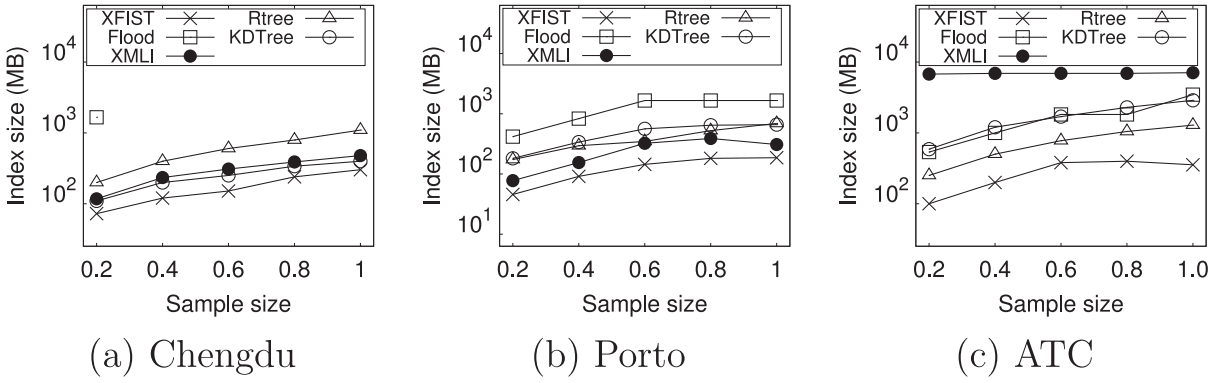


Fig. 6. Comparison of X-FIST index size by varying the size of dataset.

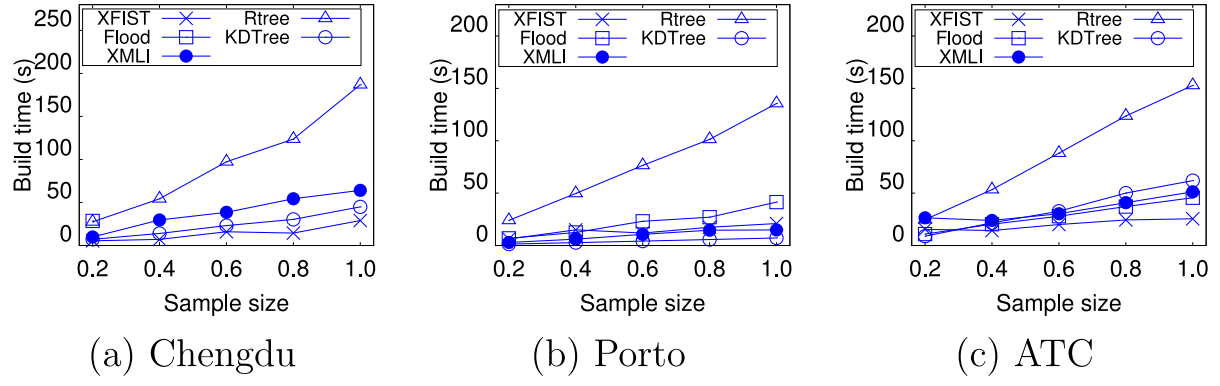


Fig. 7. Comparison of X-FIST building time by varying the size of dataset.

nodes instead of completely rebuilding the index. Currently, X-FIST uses a no-gap sorted array to organize the MBR points. In the future, extending X-FIST with the efficient updatable structures is feasible, because X-FIST uses piecewise linear model (PLM) as base model [14] where PLM has similar base model behavior with those in [15,26,27].

6.2.2. Search time

We discuss the performance of the similarity search in terms of the execution time for every dataset. The performance of original Flood in Chengdu dataset is not shown as it cannot scale to the full dataset size. Figs. 8–10 show the similarity search time for each dataset with various distances and thresholds. First, we compared the search time with different methods and thereafter compared the performance of each distance individually in each dataset.

Overall, in every dataset, the search time of X-FIST surpassed that of all the other search techniques. In Chengdu dataset, X-FIST clearly outperformed XMLI, R-tree, and Kd-tree across all distances. XMLI retrieves more candidates owing to imprecise iDistance indexing which approximates the distance between points to a centroid as reference point. Whereas, X-FIST prunes more irrelevant trajectories because of the more precise partitioning with Flood. R-tree and Kd-tree, however, performs better than XMLI due to different characteristic of range query. X-FIST, R-tree, and Kd-tree use the η boundary extension of MBR which is more suited to range queries, thus XMLI underperformed in most distance functions, except in LCSS, where early pruning with δ helped to outperform R-tree.

In Fig. 9, it is evident that X-FIST performed efficiently for all distances in Porto dataset. The performance of X-FIST is followed by Kd-tree, R-tree, XMLI, and Flood, respectively. In Porto dataset, XFIST, Kd-tree, and R-tree performed similarly to Chengdu dataset whereas XMLI performed better. The better XMLI performance is most likely due to the smaller size of Porto dataset. In every distance function, Flood is the most underperformed method because it indexes all the points thus computing the distance becomes inefficient.

In case of the ATC dataset, as depicted in Fig. 10, X-FIST, Kd-tree, and R-tree behave similarly to the Porto and Chengdu datasets. Flood, however, outperformed XMLI in ATC dataset due to Flood being more precise in pruning irrelevant trajectories due to the iDistance approximation by XMLI, except in EDR. EDR is the less strict version of LCSS that where XMLI may perform better due ATC having more attributes.

Moreover, with different distances, X-FIST demonstrated similar search time with a slight difference. The search time of X-FIST, as well as other methods, for DF and DTW were shorter than EDR and LCSS in Chengdu dataset. However, in Porto and ATC, the search time of EDR and LCSS with $\tau = 0$ is similar to DF with τ equal to 0.003 and 0.03, respectively. Performing the

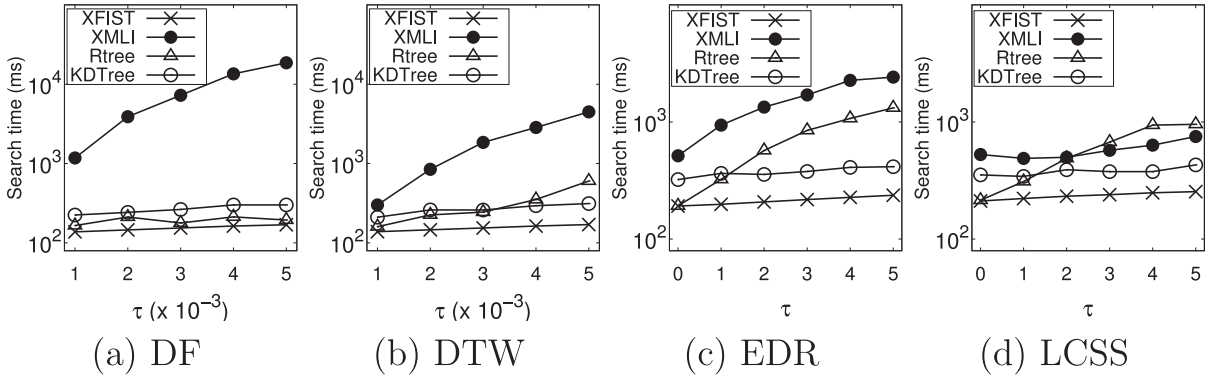


Fig. 8. Evaluation on different trajectory distances with various similarity thresholds on Chengdu dataset.

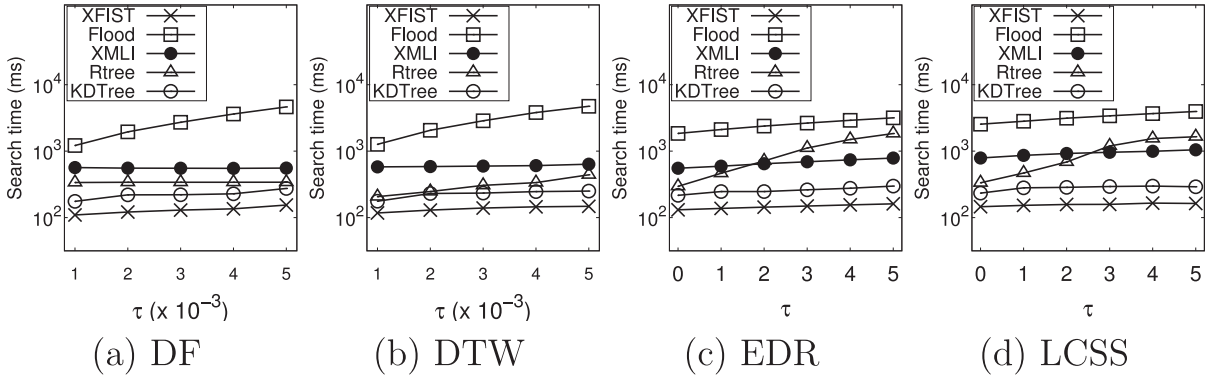


Fig. 9. Evaluation on different trajectory distances with various similarity thresholds on Porto dataset.

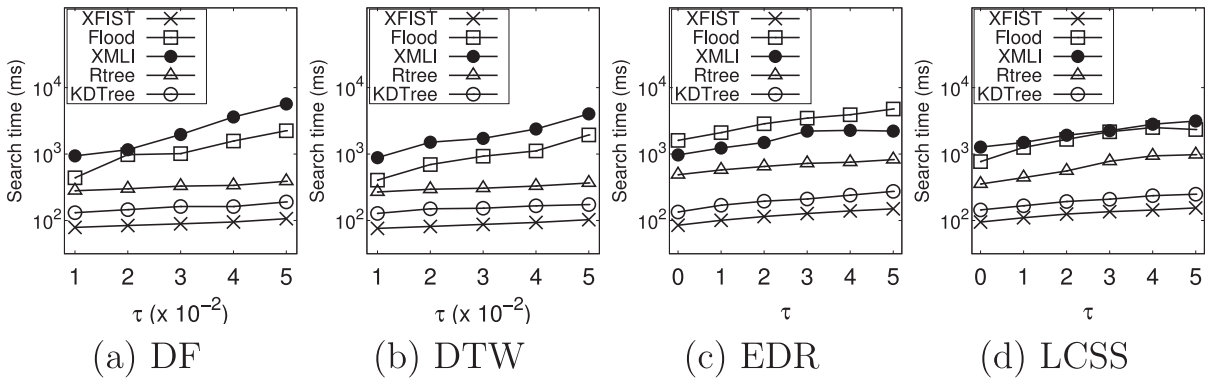


Fig. 10. Evaluation on different trajectory distances with various similarity thresholds on ATC dataset.

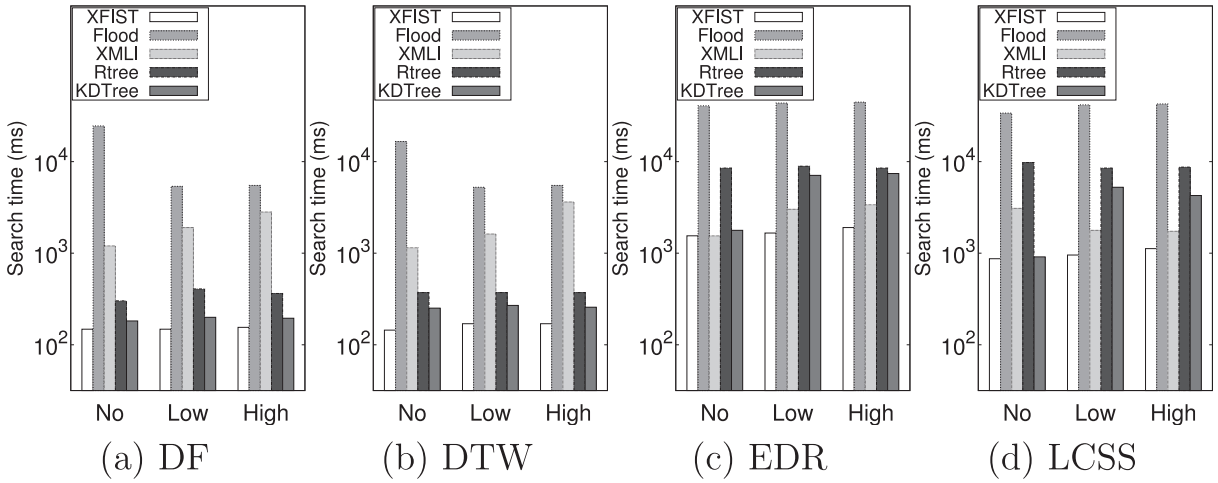


Fig. 11. Similarity search performance on modified Chengdu datasets.

similarity search with EDR and LCSS using $\tau = 0$ and ϵ equal to τ of DF have similar operations in sub-trajectory level but without (EDR) or different (LCSS) special intersection of DF and different trajectory level pruning operations. However, for the same reason, the larger size and narrower range of Chengdu dataset implies that such different operations can impact the search time greatly. Subsequently, the short search time of X-FIST in the ATC dataset shows that the a dimension does not significantly affect the performance. Thus, based on the evaluation using three different real-world datasets and various trajectory distances, we demonstrated that X-FIST performed well in both index construction and trajectory similarity search.

6.2.3. Skewness study

We generate three new trajectory datasets by considering the subset of Chengdu dataset followed by adding random trajectories such that each dataset has different skewness. The skewness of our generated datasets varies with values equal to 0.67, 3.81, and 6.78, which we later denote as 'no', 'low', and 'high' skewed datasets, respectively. The number of trajectories in the generated datasets is limited to 1 million to avoid out of memory error when building the original Flood index.

Fig. 11 depicts the similarity search time with the indices of the dataset with varying skewness. Generally, similarity search with X-FIST does not have any significant difference in all skewness in every distance. However, the performance superiority of X-FIST against other methods is less noticeable because of the smaller dataset size. In contrast, X-FIST is slightly more disadvantageous in the less skewed dataset. In Figs. 11 (c) and (d), XMLI and Kd-tree achieved similar search time with X-FIST in EDR and LCSS distance, respectively.

7. Conclusion

In this study, we proposed X-FIST, an extended Flood for efficiently searching similar trajectories in big trajectory datasets. Our approach extended Flood index to learn the distribution of MBR representation of trajectories and sub-trajectories

in the dataset. Consequently, a novel search strategy that eliminates the need to scan every retrieved point in the dataset for every range query was devised. In addition, our index supports multiple trajectory similarity measures. Thus, it is possible to use the same trained parameters whenever the measure changes without changing or retraining the Flood model. The experimental evaluation showed that X-FIST provided superior performance in terms of query time and index size storage over a tree-based index and other learned index approaches.

In future, we aim to extend the usage of X-FIST for various extensions of trajectory search cases, such as k-nearest neighbor similarity search, similarity joins, and semantic trajectory pattern mining.

CRediT authorship contribution statement

Hani Ramadhan: Conceptualization, Methodology, Software, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Joonho Kwon:** Writing – review & editing, Supervision, Project administration, Funding acquisition.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Funding: This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) by the Ministry of Education under Grant NRF-2020R1I1A3072457, in part by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2022-2020-0-01797) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation), and in part by IITP grant funded by the Korea government(MSIT) (No.2020-0-00121, Development of data improvement and dataset correction technology based on data quality assessment).

References

[1] R.S. de Sousa, A. Boukerche, A.A.F. Loureiro, Vehicle trajectory similarity: Models, methods, and applications, ACM Comput. Surv. 53 (5) (2020) 94:1–94:32. doi:10.1145/3406096.
[2] S. Wang, Z. Bao, J.S. Culpepper, G. Cong, A survey on trajectory data management, analytics, and learning, ACM Comput. Surv. 54 (2) (2021) 39:1–39:36. doi:10.1145/3440207.
[3] A. Nishad, S. Abraham, A method for continuous clustering and querying of moving objects, in: International Conference on Advanced Informatics for Computing Research, Springer, 2019, pp. 172–183.
[4] Y. Zheng, L. Zhang, X. Xie, W. Ma, Mining interesting locations and travel sequences from GPS trajectories, in: J. Quemada, G. León, Y.S. Maarek, W. Nejdl (Eds.), Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20–24, 2009, ACM, 2009, pp. 791–800. doi:10.1145/1526709.1526816.
[5] L. Moreira-Matias, J. Gama, M. Ferreira, J. Mendes-Moreira, L. Damas, Predicting taxi-passenger demand using streaming data, IEEE Trans. Intell. Transp. Syst. 14 (3) (2013) 1393–1402, https://doi.org/10.1109/TITS.2013.2262376.
[6] M.I. Ali, F. Gao, A. Mileo, Citybench: A configurable benchmark to evaluate RSP engines using smart city datasets, in: M. Arenas, Ó. Corcho, E. Simperl, M. Strohmaier, M. d'Aquin, K. Srinivas, P. Groth, M. Dumontier, J. Heflin, K. Thirunarayan, S. Staab (Eds.), The Semantic Web - ISWC 2015–14th International Semantic Web Conference, Bethlehem, PA, USA, October 11–15, 2015, Proceedings, Part II, Vol. 9367 of Lecture Notes in Computer Science, Springer, 2015, pp. 374–389. doi:10.1007/978-3-319-25010-6_25.
[7] D. Brscic, T. Kanda, T. Ikeda, T. Miyashita, Person tracking in large public spaces using 3-d range sensors, IEEE Trans. Hum. Mach. Syst. 43 (6) (2013) 522–534, https://doi.org/10.1109/THMS.2013.2283945.
[8] X. Niu, S. Wang, C.Q. Wu, Y. Li, P. Wu, J. Zhu, On a clustering-based mining approach with labeled semantics for significant place discovery, Inf. Sci. 578 (2021) 37–63, https://doi.org/10.1016/j.ins.2021.07.050.
[9] D. Zhang, K. Lee, I. Lee, Semantic periodic pattern mining from spatio-temporal trajectories, Inf. Sci. 502 (2019) 164–189, https://doi.org/10.1016/j.ins.2019.06.035.
[10] J. Zhu, C. Huang, M. Yang, G.P. Cheong Fung, Context-based prediction for road traffic state using trajectory pattern mining and recurrent convolutional neural networks, Inf. Sci. 473 (2019) 190–201, https://doi.org/10.1016/j.ins.2018.09.029.
[11] A. Guttman, R-trees: A dynamic index structure for spatial searching, in: B. Yormark (Ed.), SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, USA, June 18–21, 1984, ACM Press, 1984, pp. 47–57. doi:10.1145/602259.602266.
[12] J.L. Bentley, Multidimensional binary search trees used for associative searching, Commun. ACM 18 (9) (1975) 509–517, https://doi.org/10.1145/361002.361007.
[13] T. Kraska, A. Beutel, E.H. Chi, J. Dean, N. Polyzotis, The case for learned index structures, in: G. Das, C.M. Jermaine, P.A. Bernstein (Eds.), Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10–15, 2018, ACM, 2018, pp. 489–504. doi:10.1145/3183713.3196909.
[14] V. Nathan, J. Ding, M. Alizadeh, T. Kraska, Learning multi-dimensional indexes, in: D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, H.Q. Ngo (Eds.), Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14–19, 2020, ACM, 2020, pp. 985–1000. doi:10.1145/3318464.3380579.
[15] J. Qi, G. Liu, C.S. Jensen, L. Kulik, Effectively learning spatial indices, Proc. VLDB Endow. 13(11) (2020) 2341–2354. URL: http://www.vldb.org/pvldb/vol13/p2341-qi.pdf.
[16] H. Wang, X. Fu, J. Xu, H. Lu, Learned index for spatial queries, in: 20th IEEE International Conference on Mobile Data Management, MDM 2019, Hong Kong, SAR, China, June 10–13, 2019, IEEE, 2019, pp. 569–574. doi:10.1109/MDM.2019.00121.
[17] T. Kraska, M. Alizadeh, A. Beutel, E.H. Chi, A. Kristo, G. Leclerc, S. Madden, H. Mao, V. Nathan, Sagedb: A learned database system, in: 9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13–16, 2019, Online Proceedings, www.cidrdb.org, 2019. URL: http://cidrdb.org/cidr2019/papers/p117-kraska-cidr19.pdf.

[18] J. Ding, V. Nathan, M. Alizadeh, T. Kraska, Tsunami: A learned multi-dimensional index for correlated data and skewed workloads, CoRR abs/2006.13282. arXiv:2006.13282.
[19] A. Davitkova, E. Milchevski, S. Michel, The ml-index: A multidimensional, learned index for point, range, and nearest-neighbor queries, in: A. Bonifati, Y. Zhou, M.A.V. Salles, A. Böhm, D. Olteanu, G.H.L. Fletcher, A. Khan, B. Yang (Eds.), Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020, OpenProceedings.org, 2020, pp. 407–410. doi:10.5441/002/edbt.2020.44.
[20] Z. Shang, G. Li, Z. Bao, DITA: distributed in-memory trajectory analytics, in: G. Das, C.M. Jermaine, P.A. Bernstein (Eds.), Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10–15, 2018, ACM, 2018, pp. 725–740. doi:10.1145/3183713.3183743.
[21] J. Xu, H. Lu, R.H. Güting, Range queries on multi-attribute trajectories, IEEE Trans. Knowl. Data Eng. 30 (6) (2018) 1206–1211, https://doi.org/10.1109/TKDE.2017.2787711.
[22] R. Zhang, Y. Rong, Z. Wu, Y. Zhuo, Trajectory similarity assessment on road networks via embedding learning, in: 6th IEEE International Conference on Multimedia Big Data, BigMM 2020, New Delhi, India, September 24–26, 2020, IEEE, 2020, pp. 1–8. doi:10.1109/BigMM50055.2020.00012.
[23] H. Zhang, X. Zhang, Q. Jiang, B. Zheng, Z. Sun, W. Sun, C. Wang, Trajectory similarity learning with auxiliary supervision and optimal matching, in: C. Bessiere (Ed.), Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, ijcai.org, 2020, pp. 3209–3215. doi:10.24963/ijcai.2020/444.
[24] L. Chen, M.T. Özsu, V. Oria, Robust and fast similarity search for moving object trajectories, in: F. Özcan (Ed.), Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14–16, 2005, ACM, 2005, pp. 491–502. doi:10.1145/1066157.1066213.
[25] N. Pliakis, E. Tiakas, Y. Manolopoulos, Indexing and progressive top-k similarity retrieval of trajectories, World Wide Web 24 (1) (2021) 51–83, https://doi.org/10.1007/s11280-020-00831-w.
[26] J. Ding, U.F. Minhas, J. Yu, C. Wang, J. Do, Y. Li, H. Zhang, B. Chandramouli, J. Gehrke, D. Kossmann, D.B. Lomet, T. Kraska, ALEX: an updatable adaptive learned index, in: D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, H.Q. Ngo (Eds.), Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14–19, 2020, ACM, 2020, pp. 969–984. doi:10.1145/3318464.3389711.
[27] J. Wu, Y. Zhang, S. Chen, Y. Chen, J. Wang, C. Xing, Updatable learned index with precise positions, Proc. VLDB Endow. 14 (8) (2021) 1276–1288. URL: http://www.vldb.org/pvldb/vol14/p1276-wu.pdf.
[28] M. Antol, J. Ol'ha, T. Slaninákoá, V. Dohnal, Learned metric index–proposition of learned indexing for unstructured data, Inf. Syst. 100 (2021) 101774.
[29] E.T. Zacharatou, A. Kipf, I. Sabek, V. Pandey, H. Doraiswamy, V. Markl, The case for distance-bounded spatial approximations, CoRR abs/2010.12548. arXiv:2010.12548.
[30] X. Li, K. Zhao, G. Cong, C.S. Jensen, W. Wei, Deep representation learning for trajectory similarity computation, in: 34th IEEE International Conference on Data Engineering, ICDE 2018, IEEE Computer Society, 2018, pp. 617–628, https://doi.org/10.1109/ICDE.2018.00062.
[31] D. Yao, G. Cong, C. Zhang, J. Bi, Computing trajectory similarity in linear time: A generic seed-guided neural metric learning approach, in: 35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8–11, 2019, IEEE, 2019, pp. 1358–1369. doi:10.1109/ICDE.2019.00123.
[32] P. Han, J. Wang, D. Yao, S. Shang, X. Zhang, A graph-based approach for trajectory similarity computation in spatial networks, in: F. Zhu, B.C. Ooi, C. Miao (Eds.), KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14–18, 2021, ACM, 2021, pp. 556–564. doi:10.1145/3447548.3467337.
[33] Z. Wang, C. Long, G. Cong, Y. Liu, Efficient and effective similar subtrajectory search with deep reinforcement learning, Proc. VLDB Endow. 13 (11) (2020) 2312–2325. URL: http://www.vldb.org/pvldb/vol13/p2312-wang.pdf.
[34] T. Eiter, H. Mannila, Computing discrete fréchet distance (Tech. rep.), Citeseer, 1994.
[35] B. Yi, H.V. Jagadish, C. Faloutsos, Efficient retrieval of similar time sequences under time warping, in: S.D. Urban, E. Bertino (Eds.), Proceedings of the Fourteenth International Conference on Data Engineering, Orlando, Florida, USA, February 23–27, 1998, IEEE Computer Society, 1998, pp. 201–208. doi:10.1109/ICDE.1998.655778.
[36] K. Toohey, M. Duckham, Trajectory similarity measures, ACM SIGSPATIAL Special 7 (1) (2015) 43–50, https://doi.org/10.1145/2782759.2782767.
[37] G. Roh, J. Roh, S. Hwang, B. Yi, Supporting pattern-matching queries over trajectories on road networks, IEEE Trans. Knowl. Data Eng. 23 (11) (2011) 1753–1758, https://doi.org/10.1109/TKDE.2010.189.
[38] L. Chen, R.T. Ng, On the marriage of lp-norms and edit distance, in: M.A. Nascimento, M.T. Özsu, D. Kossmann, R.J. Miller, J.A. Blakeley, K.B. Schiefer (Eds.), (e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB 2004, Toronto, Canada, August 31 - September 3 2004, Morgan Kaufmann, 2004, pp. 792–803. doi:10.1016/B978-012088469-8.50070-X. URL: http://www.vldb.org/conf/2004/RS21P2.PDF.
[39] M.G. Bulmer, Principles of statistics, Courier Corporation (1979) 61–63.