The VLDB Journal (2018) 27:395–420
https://doi.org/10.1007/s00778-018-0502-0

REGULAR PAPER

Parallel trajectory similarity joins in spatial networks

Shuo Shang[1] · Lisi Chen[2] · Zhewei Wei[3] · Christian S. Jensen[4] · Kai Zheng[5] · Panos Kalnis[1]

Received: 28 July 2017 / Revised: 19 February 2018 / Accepted: 14 March 2018 / Published online: 4 April 2018

Abstract
The matching of similar pairs of objects, called similarity join, is fundamental functionality in data management. We consider two cases of trajectory similarity joins (TS-Joins), including a threshold-based join (Tb-TS-Join) and a top-k TS-Join (k-TS-Join), where the objects are trajectories of vehicles moving in road networks. Given two sets of trajectories and a threshold θ, the Tb-TS-Join returns all pairs of trajectories from the two sets with similarity above θ. In contrast, the k-TS-Join does not take a threshold as a parameter, and it returns the top-k most similar trajectory pairs from the two sets. The TS-Joins target diverse applications such as trajectory near-duplicate detection, data cleaning, ridesharing recommendation, and traffic congestion prediction. With these applications in mind, we provide purposeful definitions of similarity. To enable efficient processing of the TS-Joins on large sets of trajectories, we develop search space pruning techniques and enable use of the parallel processing capabilities of modern processors. Specifically, we present a two-phase divide-and-conquer search framework that lays the foundation for the algorithms for the Tb-TS-Join and the k-TS-Join that rely on different pruning techniques to achieve efficiency. For each trajectory, the algorithms first find similar trajectories. Then they merge the results to obtain the final result. The algorithms for the two joins exploit different upper and lower bounds on the spatiotemporal trajectory similarity and different heuristic scheduling strategies for search space pruning. Their per-trajectory searches are independent of each other and can be performed in parallel, and the mergings have constant cost. An empirical study with real data offers insight in the performance of the algorithms and demonstrates that they are capable of outperforming well-designed baseline algorithms by an order of magnitude.

Keywords Trajectory similarity join · Parallel processing · Spatial networks · Spatiotemporal databases

1 Introduction

The continued proliferation of GPS-equipped mobile devices (e.g., vehicle navigation systems and smart phones) and the proliferation of online map-based services (e.g., Bing Maps[1], Google Maps[2], and MapQuest[3]) enable the collection and sharing of trajectories. For example, the sites Bikely[4],

✉ Lisi Chen
chenlisi.cs@gmail.com

Shuo Shang
jedi.shang@gmail.com

Zhewei Wei
zhewei@ruc.edu.cn

Christian S. Jensen
csj@cs.aau.dk

Kai Zheng
zhengkai@uestc.edu.cn

Panos Kalnis
panos.kalnis@kaust.edu.sa

1 CEMSE, King Abdullah University of Science and Technology, Thuwal, Saudi Arabia

2 School of Computer and Information Technology, University of Wollongong, Wollongong, Australia

3 School of Information, Renmin University of China, Beijing, China

4 Department of Computer Science, Aalborg University, Aalborg, Denmark

5 School of Computer Science and Engineering and Big Data Research Center, University of Electronic Science and Technology of China, Chengdu, China

1 https://www.bing.com/maps/.
2 https://maps.google.com/.
3 https://www.mapquest.com.
4 https://www.bikely.com/.

GPS-way-points[5], Share-my-routes[6], and Microsoft Geolife[7] enable such sharing, and more and more social network sites, including Twitter[8], Facebook[9], and Foursquare[10], are starting to support trajectory sharing and search. This development motivates new studies of the management and analysis of massive trajectory data. In these settings, trajectory similarity joins (TS-Joins), including a threshold-based TS-Join (Tb-TS-Join) and a top-k TS-Join (k-TS-Join), constitute fundamental functionality. Given two sets of trajectories as arguments and a similarity threshold θ as a parameter, the Tb-TS-Join returns all pairs of trajectories from the two sets with similarity above θ. In contrast, the k-TS-Join does not take a similarity threshold θ as a parameter and it returns k most similar trajectory pairs from the two sets. The k-TS-Join is useful in cases where a user cannot specify a purposeful similarity threshold.

The TS-Joins may bring significant benefits to diverse applications, including trajectory near-duplicate detection, data cleaning [2,20], ridesharing recommendation [17,18], friend recommendation [18], frequent trajectory-based routing [13,20], and traffic congestion prediction [27,28]. For example, a database may contain several copies of a trajectory or several similar trajectories. We may conduct a TS-Join (self join) on the database to identify duplicate or similar trajectories, thus supporting data cleaning. For example, having found similar trajectory pairs (τ_1,τ_2), (τ_1,τ_3), (τ_1,τ_4), we may choose to retain only the representative trajectory τ_1. The identification of similar trajectories of different commuters is also useful in ridesharing recommendation and friend recommendation. For example, commuters may find potential ridesharing partners among commuters with similar trajectories, and social networking services may identify users with similar living trajectories and use this in friend recommendations. We may also use a TS-Join to find frequently traveled trajectories (e.g., trajectory τ joins with m other trajectories, thus having travel frequency $m + 1$), which may be used for route recommendation and in traffic analyses to predict congestion.

To the best of our knowledge, this is the first study of a trajectory similarity join that takes into account both spatial and temporal aggregate distances. We adopt a linear combination method (e.g., [18,19]) to combine the spatial and temporal similarity into a spatiotemporal similarity metric. In contrast, existing trajectory similarity joins (e.g., [2,3,6,10]) use a time interval threshold to constrain the temporal proximity of two trajectories (in a fixed manner) and can be assigned to two different categories.

Studies in the first category (e.g., [3,10]) eliminate trajectory pairs that are temporally further apart than a threshold. We generalize this category of studies and compute temporal similarity by summarizing temporal proximities of sample point pairs from two trajectories (aggregate-distance matching), thus obviating the need for a time threshold.

Studies in the other category (e.g., [2,6]) utilize a sliding window for all trajectories and eliminate pairs of trajectories with times that fall outside the window. For the remaining pairs, only spatial proximity is considered. However, in the applications that motivate our study, spatial proximity is by itself insufficient to evaluate the relationship between different trajectories. With the approach in the second category, a ridesharing service may recommend a co-traveler with a very different departure time to a traveler. Although the trajectories of the travelers may be spatially close to each other, the travelers may not be satisfied with the recommendation, as their preferences are not fulfilled.

An example similarity join is shown in Fig. 1, where τ_1, τ_2, and τ_3 are trajectories and $P = \{\tau_1\}$ and $Q = \{\tau_2, \tau_3\}$. A trajectory is a sequence of timestamped sample points of a moving object. In the example, p_1, p_2, …, p_{15} are timestamped sample points. Given a time interval (8:30, 10:30), existing sliding window-based trajectory similarity joins (e.g., [2,6]) return trajectory pairs (τ_1, τ_2) and (τ_1, τ_3) because they are spatially close to each other. However, τ_1 and τ_2 have very different departure times, thus rendering a result such as this of little use in ridesharing and traffic congestion prediction. In the applications we target, it is difficult to obtain an appropriate query time interval. The TS-Joins (assuming that $\mathrm{Sim}(\tau_1, \tau_3) \geq \theta$ in the Tb-TS-Join and $k = 1$ in the k-TS-Join) return trajectory pair (τ_1, τ_3) without the need for a query time interval, and the spatial and temporal domains are considered appropriately in the matching.

Next, unlike existing trajectory similarity joins [2,3,6,10,20], the TS-Joins are applied in spatial networks because in many practical scenarios, the objects move in spatial networks rather than in Euclidean space. In spatial networks, network distance is the relevant distance between two objects, and using Euclidean distance may lead to errors. We assume that the sample points of trajectories in sets P and Q have been map matched to the corresponding spatial network (the spatial domain) according to some map-matching algorithm (e.g., [4,22]), and we assume that the timestamps of all trajectory sample points are mapped to a time axis with a 24-hour range (the temporal domain) [18].

Existing methods cannot process the TS-Joins due to three reasons. (i) Different query spaces (Euclidean vs. network): existing joins (e.g., [2,3,6,10,20]) are conducted in Euclidean space rather than in a spatial network. Existing spatial indices (e.g., the R-tree [11]) and accompanying techniques lack

5 https://www.gps-waypoints.net.

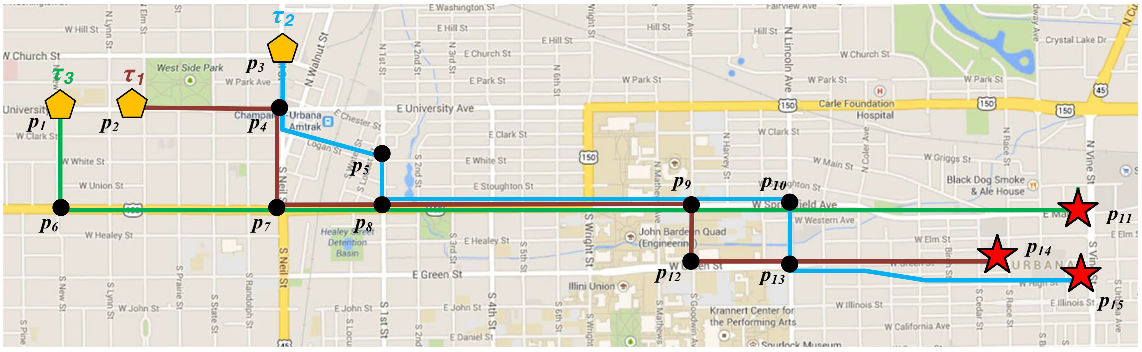
6 https://www.sharemyroutes.com/.

7 https://research.microsoft.com/en-us/projects/geolife/.

8 https://www.twitter.com/.

9 https://www.Facebook.com/.

10 https://www.Foursquare.com/.



$\tau_1 = <p_2, 09{:}37>, <p_4, 09{:}40>, <p_7, 09{:}48>, <p_8, 09{:}51>, <p_9, 09{:}57>, <p_{12}, 10{:}02>, <p_{13}, 10{:}05>, <p_{14}, 10{:}07>$

$\tau_2 = <p_3, 08{:}35>, <p_4, 08{:}39> <p_5, 08{:}46>, <p_8, 08{:}49>, <p_9, 09{:}01>, <p_{10}, 09{:}04>, <p_{13}, 09{:}06>, <p_{15}, 09{:}07>$

$\tau_3 = <p_1, 09{:}32>, <p_6, 09{:}43>, <p_7, 09{:}48>, <p_8, 09{:}51>, <p_9, 09{:}59>, <p_{10}, 10{:}03>, <p_{11}, 10{:}13>$

● : sample point in a trajectory ⬟ : start point of a trajectory ★ : destination point of a trajectory

Fig. 1 TS-Joins example

effectiveness in our setting. (ii) Different temporal matching schemes (time interval vs. aggregate distance): most existing trajectory similarity joins are time interval-based (e.g., [2,3,6,10]), and their solutions are inapplicable of aggregate-distance matching. They compute different results than the TS-Joins (cf. Fig. 1). (iii) Parallel processing: an experimental study [12] shows that existing centralized similarity join techniques (that do not take parallel processing into account) are inefficient when processing very large data sets. Existing centralized trajectory similarity joins (e.g., [2,3,6,10,20]) can process at most 500 K trajectories (based on reported experiments), while the TS-Joins are able to process 10 M trajectories with a reasonable runtime (e.g., processing 10 M $\times$ 2 M trajectories for non-self Tb-TS-Join in 220 s and for non-self k-TS-Join in 820 s. The k-TS-Join takes longer because it does not have a threshold that enables search space pruning). A comparison between the TS-Joins and existing studies is shown in Table 1. Section 7 covers related work in more detail.

We initially propose a relatively straightforward approach to the TS-Joins (Tb-TS-Join and k-TS-Join) called temporal-first matching. We apply a hierarchical grid index in the temporal domain. Then we refine the candidate trajectory pairs in the same leaf node (trajectories in the same node are temporally similar) by computing their spatiotemporal similarities. By merging the results from the leaf nodes toward the root, the join result is obtained when the root is reached. The computations at each index level occur in parallel. Following this framework, we develop two algorithms, called TF-Matching and k-TF-Matching that compute the Tb-TS-Join and the k-TS-Join, respectively. The main difference between TF-Matching and k-TF-Matching is the pruning techniques they employ. The trajectory-similarity lower bound in TF-Matching is defined according to the threshold θ. In contrast, k-TF-Matching has no threshold, so the upper and lower bounds, the pruning within a thread, and the merging among different threads must be reworked.

The two new algorithms are enabled by four specific technical contributions: pruning in leaf nodes, pruning among different nodes, merging, and parallel processing. The only similarity between the proposed algorithms and the sliding window-based trajectory similarity methods [2,6] is that the similarity-join computation in a leaf node is equivalent to the processing of a query issued within a temporal-matching window (the first contribution). The optimization techniques in sliding window-based methods cannot be used in our algorithm because of the different query spaces (Euclidean vs. network) and the different temporal matching schemes (time interval vs. aggregate distance).

The temporal-first matching has three main limitations. First, it is driven by the temporal domain and thus has weak spatial pruning power. As a result, large numbers of pairs must be considered. Second, while having many leaf nodes enables more parallel processing, this also increases the merging cost. Third, it is potentially costly to acquire network distances when computing spatial similarities.

To process the TS-Joins more efficiently, we propose a two-phase divide-and-conquer search framework. In the trajectory-search phase, for each trajectory τ, we explore the spatial and temporal domains concurrently to find trajectories that are similar to τ. In the spatial domain, network expansion [9] is adopted from each sample point of τ, while in the temporal domain, we expand the search from each timestamp of τ. The trajectory-search processes are independent of each other, enabling parallel processing, and the merging cost is constant (uncorrelated to the number of threads used for parallel processing). The two-phase algorithm has a stronger pruning power. The network distances for similarity com-

Table 1 Comparison to existing trajectory similarity joins

Studies	Space	Temporal matching	Parallel	Data
[2]	Euclidean	Sliding window-based	No	50 K
[6]	Euclidean	Sliding window-based	No	250 K
[3]	Euclidean	Time threshold-based	No	150 K
[10]	Euclidean	Time threshold-based	No	2 K
[20]	Euclidean	Spatial join only	No	500 K
TS-Joins	Network	Aggregate-distance matching	Yes	10 M

putation can be derived directly during the trajectory-search process. A time complexity analysis indicates that the two-phase approach is considerably better than the temporal-first matching approach. Following this framework, we develop two specific algorithms, called two-phase and k-two-phase that compute the Tb-TS-Join and the k-TS-Join, respectively. The main difference between two-phase and k-two-phase is their pruning techniques. In two-phase, an upper bound on the spatiotemporal similarity is defined and employed for pruning the search space, and a heuristic scheduling strategy is proposed to schedule multiple so-called query sources in order to improve efficiency. The bound and scheduling are defined according to threshold θ. In contrast, k-two-phase has no threshold, so the upper and lower bounds, the pruning within a thread, the merging among different threads, and the heuristic search strategy must be reworked.

The present paper expands on a previous study [16]. Specifically, we propose a novel top-k TS-Join (k-TS-Join) that retrieves the top-k most similar trajectory pairs from two trajectory sets without taking a threshold θ as a query parameter. Leveraging the frameworks of the TF-Matching and two-phase algorithms, we develop new k-TF-Matching and k-two-phase algorithms that enable parallel processing (cf. Sects. 3.2 and 4.2). Unlike the Tb-TS-Join, the k-TS-Join has no threshold that it can use for pruning. Thus, we develop new upper and lower bounds and rework the pruning and scheduling within a thread and the merging among different threads (cf. Equations 13–17, and 28–31). Next, we add a new trajectory similarity measure that takes the visiting sequence of sample points into account when matching trajectories, and we extend the TF-Matching, k-TF-Matching, two-phase, and k-two-phase algorithms to support this new similarity measure (cf. Sect. 5). We also report on experiments that offer insight into the performance of the two new algorithms (cf. Sect. 6.3, Figs. 10, 11, 12, 13 and 14, including 34 subfigures) and the four extended algorithms (cf. Sect. 6.4, Figures 15, 16, including 12 subfigures) in different settings.

To sum up, the contributions of the paper are as follows.

– We propose two novel network-based trajectory similarity joins, called TB-TS-Join and k-TS-Join, that use aggregate-distance matching to quantify similarity, thus targeting applications such as trajectory near-duplicate detection, ridesharing recommendation, route planning, and traffic congestion prediction.
– The TS-Joins use new metrics to evaluate trajectory similarity in the spatial and temporal domains (Sect. 2).
– We develop two temporal-first baseline algorithms, TF-Matching and k-TF-Mathcing, that enable parallel TB-TS-Join and k-TS-Join processing (Sects. 3.1 and 4.1).
– We develop two-phase and k-two-phase algorithms, each with effective pruning and scheduling techniques that enable parallel TB-TS-Join and k-TS-Join processing (Sects. 3.2 and 4.2).
– We extend the TF-Matching, k-TF-Matching, two-phase, and k-two-phase algorithms to scenarios where the visiting sequence of sample points is to be taken into account when matching trajectories (Sect. 5).
– We conduct extensive experiments on large trajectory sets that offer insight into the performance of the developed algorithms (Sect. 6).

The rest of the paper is organized as follows. Section 2 introduces the spatial network setting and the trajectory similarity metrics used in the paper, and it defines the problem. The temporal-first matching and k-temporal-first matching algorithms are covered in Sect. 3, while the two-phase and k-two-phase algorithms are covered in Sect. 4. The two-phase and k-two-phase algorithms are further extended to support practical scenarios in Sect. 5, which is followed by a presentation of experimental results in Sect. 6. Related work is covered in Sect. 7, and conclusions and research directions are presented in Sect. 8.

2 Preliminaries

2.1 Spatial networks and trajectories

A spatial network is modeled as a connected, undirected graph $G = (V, E, F, W)$, where V is a vertex set and $E \subseteq \{\{v_i, v_j\} | v_i, v_j \in V \wedge v_i \neq v_j\}$ is an edge set. A vertex $v_i \in V$ represents a road intersection or an end of a road, and an edge $e_k = \{v_i, v_j\} \in E$ represents a road segment that enables travel between vertices v_i and v_j. Function $F : V \cup E \rightarrow Geometries$ maps a vertex to the point location of the corresponding road intersection and maps an edge to a polyline representing the corresponding road segment. Function $W : E \rightarrow R$ assigns a real-valued weight $W(e)$ to an edge e that represents the corresponding road segment's length.

The shortest path between two vertices v_i and v_j is a sequence of edges linking v_i and v_j such that the sum of the edge weights is minimal. Such a path is denoted by $SP(v_i, v_j)$, and its length is denoted by $sd(v_i, v_j)$. Euclidean space-based spatial indices (e.g., the R-tree [11]) and accompanying techniques are ineffective in network environments due to loose lower bounds. For simplicity, we assume that the data points considered (e.g., trajectory sample points) are located on vertices. It is straightforward to also support data points on edges. Assume a data point p is on an edge e with given network distances to the two end vertices e_a and e_b. Then, a new vertex is created for p and edge e is replaced by edges (e_a, p) and (p, e_b).

Raw trajectory samples obtained from GPS devices are typically of the form of (longitude, latitude, time). We assume that all trajectory sample points have already been map matched onto the vertices of the spatial network using some map-matching algorithm (e.g., [4,22]) and that between two adjacent sample points p_a and p_b, the object movement always follows the shortest path connecting p_a and p_b. A trajectory is defined as follows.

Definition: trajectory
A trajectory τ of a moving object is a finite, time-ordered sequence $\langle v_1, v_2, \ldots, v_n\rangle$, where $v_i = (p_i, t_i)$, $i \in [1, n]$, with p_i being a sample point (equal to some vertex in $G.V$) and t_i being a timestamp.

The value of a timestamp is set to be within the range of 24 h, and the date is not taken into account because in many practical scenarios like urban transportation, most movements occur daily.

Notice that the modeling of spatial networks and trajectories align with previous studies [14,15,18,19].

2.2 Trajectory similarity functions

Given a sample point $v = (v.p, v.t)$ and a trajectory τ, the spatial network distance $d(v.p, \tau)$ and the temporal distance $d(v.t, \tau)$ between v and τ are defined as follows.

$$d(v.p, \tau) = \min_{v_i \in \tau}\{sd(v.p, v_i.p)\} \tag{1}$$

$$d(v.t, \tau) = \min_{v_i \in \tau}\{|v.t - v_i.t|\} \tag{2}$$

Given trajectories $\tau_1 = \langle v_1, v_2, \ldots, v_m\rangle$ and $\tau_2 = \langle v_1, v_2, \ldots, v_n\rangle$, the spatial and temporal similarities, $\mathrm{Sim}_\mathrm{S}(\tau_1, \tau_2)$ and $\mathrm{Sim}_\mathrm{T}(\tau_1, \tau_2)$, between them are defined as follows.

$$\mathrm{Sim}_\mathrm{S}(\tau_1, \tau_2) = \frac{\sum_{v_i \in \tau_1} \mathrm{e}^{-d(v_i.p, \tau_2)}}{|\tau_1|} + \frac{\sum_{v_j \in \tau_2} \mathrm{e}^{-d(v_j.p, \tau_1)}}{|\tau_2|} \tag{3}$$

$$\mathrm{Sim}_\mathrm{T}(\tau_1, \tau_2) = \frac{\sum_{v_i \in \tau_1} \mathrm{e}^{-d(v_i.t, \tau_2)}}{|\tau_1|} + \frac{\sum_{v_j \in \tau_2} \mathrm{e}^{-d(v_j.t, \tau_1)}}{|\tau_2|} \tag{4}$$

Here, $|\tau|$ denotes the number of sample points in a trajectory. We extend Euclidean-based trajectory similarity [7] to make it fit into spatial networks. We ensure that the similarity measures are symmetrical, such that $\mathrm{Sim}_\mathrm{S}(\tau_1, \tau_2) = \mathrm{Sim}_\mathrm{S}(\tau_2, \tau_1)$ and $\mathrm{Sim}_\mathrm{T}(\tau_1, \tau_2) = \mathrm{Sim}_\mathrm{T}(\tau_2, \tau_1)$. In contrast, most of existing trajectory similarity measures (e.g., [7,17,18,21]) are asymmetrical; thus, they cannot be used directly in the TS-Join.

Note also that spatial and temporal similarities are in the range $[0, 2]$. Finally, we use a linear combination method [17–19] to combine spatial and temporal similarities (Eqs. 3 and 4), and the spatiotemporal similarity is defined as follows.

$$\mathrm{Sim}_\mathrm{ST}(\tau_1, \tau_2) = \lambda \cdot \mathrm{Sim}_\mathrm{S}(\tau_1, \tau_2) + (1-\lambda) \cdot \mathrm{Sim}_\mathrm{T}(\tau_1, \tau_2) \tag{5}$$

Here, parameter $\lambda \in [0, 1]$ controls the relative importance of the spatial and temporal similarities. We support queries with arbitrary values of λ.

2.3 Problem definition

The two queries considered are defined as follows.

Definition: Tb-TS-Join
Given sets P and Q of trajectories and a threshold θ, the threshold-based trajectory similarity join (Tb-TS-Join) finds the set A of all trajectory pairs from the two sets whose spatiotemporal similarity no smaller than θ, i.e., $\forall(\tau_i, \tau_j) \in (P \times Q)\backslash A\ (\mathrm{Sim}_\mathrm{ST}(\tau_i, \tau_j) < \theta)$.

Definition: k-TS-Join
Given sets P and Q of trajectories, the top-k trajectory similarity join (k-TS-Join) finds a set A of k most similar trajectory pairs from the two sets, i.e., $|A| = k$ and $\forall(\tau_i, \tau_j) \in A$ $(\forall(\tau_i', \tau_j') \in (P \times Q)\backslash A\ (\mathrm{Sim}_\mathrm{ST}(\tau_i, \tau_j) \geq (\tau_i', \tau_j')))$.

The problem addressed is that of processing the two types of join efficiently given the setting and similarity definition provided in Sects. 2.1 and 2.2.

We initially consider the self-join scenario (i.e., $P = Q$) and then cover the case $P \neq Q$ in Sects. 3.1.5 and 4.1.6.

3 Baseline algorithms

We propose two baseline algorithms, TF-Matching and k-TF-Matching, to compute the Tb-TS-Join and k-TS-Join, respectively.

3.1 TF-matching

3.1.1 Basic idea

Temporal-first matching (TF-Matching) is a straightforward baseline approach to computing the Tb-TS-Join. Initially, we index the temporal domain using a hierarchical grid structure. Then we refine trajectory pairs in the same leaf node by computing their spatiotemporal similarities (Sects. 3.1.2 and 3.1.3). By merging the results from the leaf nodes toward the root, the join result is obtained when the root is reached (Sect. 3.1.4). A pair of upper and lower bounds are used to prune the search space in the spatial and temporal domains. The computations at each grid level can be performed in parallel.

3.1.2 Grid index

The grid index structure [8] is established as follows. First, we partition the temporal domain into α equal-sized time slots, each of which corresponds to a leaf node. Next, we build up a tree structure in a bottom-up manner. Assume that there are k nodes at the current level (initially $k = \alpha$). Then we build $\lfloor \frac{k}{2} \rfloor$ parent nodes. We do this recursively until there is one parent, which is the root node. The height of the tree is $\lceil log(\alpha) \rceil + 1$. An example is shown in Fig. 2, where $n_1, n_2, \ldots, n_{23}$ are nodes and n_{23} is the root. To find a value for α that yields high performance, we conducted extensive experiments when establishing the grid index.

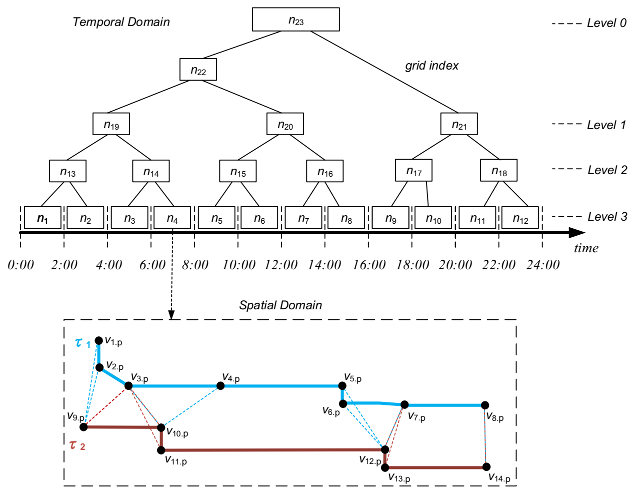


Fig. 2 An example of TF-Matching

The temporal range range($\emptyset$) of a trajectory $\tau = \langle v_1, v_2, \ldots, v_i \rangle$ is defined by the timestamps of its start and end sample points, i.e., range($\emptyset$) $= [v_1.t, v_i.t]$. When we add a new trajectory τ to the index, it is stored in the lowest node n that fully covers its temporal range, i.e., range($\emptyset$) $\subseteq$ range(n) and range($\emptyset$) is not contained in the range of any child of n. If we delete a trajectory from the index, we can simply remove it without any other changes.

Example Consider trajectories τ_1 and τ_2 in Fig. 2, where range($\emptyset_1$) = [6:15, 7:30] and range($\emptyset_2$) = [6 : 20, 7 : 35]. We insert them into the grid index top-down. Both τ_1 and τ_2 are stored in node n_4 (range(n_4) $=$ [6 : 00, 8 : 00]) because range($\emptyset_1$) $\subseteq$ range(n_4), range($\emptyset_2$) $\subseteq$ range(n_4), and n_4 is a leaf node. Given a trajectory τ_3 and range($\emptyset_3$) $=$ [9 : 45, 10 : 30], τ_3 is stored in n_{15} (range(n_{15}) $=$ [8 : 00, 12 : 00]) because range($\emptyset_3$) $\subseteq$ range(n$_{15}$), and range($\emptyset_3$) $\nsubseteq$ range(n$_5$), range($\emptyset_3$) $\nsubseteq$ range(n_6) (n_5 and n_6 are child nodes of n_{15}).

3.1.3 Upper and lower bounds

In the example in Fig. 2, trajectories τ_1 and τ_2 are stored in leaf node n_4, and they are temporally close to each other. We estimate the upper bound on the temporal similarity $\text{Sim}_\text{T}(\tau_1, \tau_2)$ (Eq. 4) as follows.

$$\sum_{v_i \in \tau_1} \mathrm{e}^{-d(v_i.t, \tau_2)} \leq |\tau_1| \quad \text{and} \quad \sum_{v_j \in \tau_2} \mathrm{e}^{-d(v_j.t, \tau_1)} \leq |\tau_2|$$
$$\Rightarrow \text{Sim}_\text{T}(\tau_1, \tau_2).ub = 2 \geq \text{Sim}_\text{T}(\tau_1, \tau_2) \tag{6}$$

Here, $|\tau|$ is the number of sample points in τ. By substituting Eq. 6 into Eq. 5, we have that

$$\lambda \cdot \text{Sim}_\text{S}(\tau_1, \tau_2) + (1 - \lambda) \cdot \text{Sim}_\text{T}(\tau_1, \tau_2) \geq \theta$$
$$\Rightarrow \text{Sim}_\text{S}(\tau_1, \tau_2) \geq \frac{\theta - (1 - \lambda) \cdot 2}{\lambda} = LB_S, \tag{7}$$

where LB_S is a global lower bound on the spatial similarity of all "qualified" trajectory pairs in the same leaf node, and LB_S is suitable for all leaf nodes. If $\text{Sim}_\text{S}(\tau_1, \tau_2) < LB_S$, the spatiotemporal similarity of (τ_1, τ_2) is less than θ, and (τ_1, τ_2) can be pruned safely.

Lemma 1 *Given any two trajectories τ_1 and τ_2, we have that*

$$\forall v \in \tau_1 (d(v.p, \tau_2) \geq \min_{v_i \in \tau_2} \{d(v_i.p, \tau_1)\}). \tag{8}$$

Proof Assume that $d(v.p, \tau_2) = sd(v.p, v'.p)$, where $v'.p$ is the sample point spatially closest to $v.p$ among all sample points in τ_2. According to Eq. 1, for sample point $v'.p$, we have that $d(v'.p, \tau_1) = \min_{v_i \in \tau_1} \{sd(v'.p, v_i.p)\} \leq$

$sd(v.p, v'.p) = d(v.p, \tau_2)$. Therefore, we have that

$$d(v.p, \tau_2) \geq d(v'.p, \tau_1) \geq \min_{v_i \in \tau_2}\{d(v_i.p, \tau_1)\}.$$

□

By substituting Eq. 8 into Eq. 3, we estimate the upper bound $\mathrm{Sim}_\mathrm{S}(\tau_1, \tau_2).ub$ of the spatial similarity between τ_1 and τ_2 as follows.

$$\begin{aligned}
&\sum_{v_i \in \tau_1} \mathrm{e}^{-d(v_i.p,\tau_2)} \leq |\tau_1| \cdot \mathrm{e}^{-\min_{v_i \in \tau_2}\{d(v_i.p,\tau_1)\}} \\
&\Rightarrow \mathrm{Sim}_\mathrm{S}(\tau_1, \tau_2).ub \\
&= \mathrm{e}^{-\min_{v_i \in \tau_2}\{d(v_i.p,\tau_1)\}} + \frac{\sum_{v_j \in \tau_2} \mathrm{e}^{-d(v_j.p,\tau_1)}}{|\tau_2|} \qquad (9)
\end{aligned}$$

According to Eq. 9, we only need to compute half of the exact spatial similarity to get the upper bound. To compute the minimum distance from a sample point v to a trajectory τ, we expand network expansion from v, and the first scanned sample point v' in τ is just the sample point closest to v ($sd(v.p, v'.p) = d(v.p, \tau)$). Assume τ has m sample points, and τ' has n sample points, to compute the exact similarity, we have to conduct $(m+n)$ times network expansions. By using Eq. 9, we only need to conduct m times network expansions to get the upper bound on spatial similarity.

For any trajectory pair (τ_1, τ_2) in the same leaf node, if its spatial-similarity upper bound $\mathrm{Sim}_\mathrm{S}(\tau_1, \tau_2).ub$ (Eq. 9) is less than the global lower bound LB_S of the spatial similarity (Eq. 7), (τ_1, τ_2) cannot have a spatiotemporal similarity that exceeds θ. Hence, (τ_1, τ_2) can be pruned safely. For the remaining trajectory pairs, we compute their exact spatiotemporal similarities, and maintain the qualified pairs, i.e., $\mathrm{Sim}_\mathrm{ST}(\tau_1, \tau_2) \geq \theta$.

Notice that the computations in different leaf nodes are independent. Thus, we can perform these computations in parallel.

3.1.4 Merging

Having computed the spatiotemporal similarities of the trajectory pairs in the leaf nodes, we merge the computation results from the leaf level to the root level iteratively (bottom-up). We merge two leaf nodes n_a and n_b to their parent node n_c (e.g., merging n_3, n_4 to n_{14} in Fig. 2). Besides the qualified trajectory pairs in n_a and n_b, we also need to consider the trajectory pairs (τ, τ') in the following three cases:

(1) $\mathrm{range}(\emptyset) \subseteq \mathrm{range}(n_a) \wedge \mathrm{range}(\emptyset') \subseteq \mathrm{range}(n_c)$
(2) $\mathrm{range}(\emptyset) \subseteq \mathrm{range}(n_b) \wedge \mathrm{range}(\emptyset') \subseteq \mathrm{range}(n_c)$
(3) $\mathrm{range}(\emptyset) \subseteq \mathrm{range}(n_a) \wedge \mathrm{range}(\emptyset') \subseteq \mathrm{range}(n_b)$

In the first two cases, we use the same lower and upper bounds (Eqs. 7 and 9) and pruning techniques as we use for trajectory pairs in the same node (cf. Sect. 3.1.3). The qualified trajectory pairs are stored in n_c.

To explain the third case, we assume that $\tau = \langle v_1, v_2, \ldots, v_i\rangle$, $\tau' = \langle v'_1, v'_2, \ldots, v'_j\rangle$, $\mathrm{range}(n_a) = [t_1, t_2]$, and $\mathrm{range}(n_b) = [t_3, t_4]$. We define the minimum temporal distance $d_T(\tau, n)$ as follows.

$$d_T(\tau, n) = \min\{|\tau.t.lb - n.t.ub|, |\tau.t.ub - n.t.lb|\} \qquad (10)$$

Here, $\tau.t.lb$ and $\tau.t.ub$ are the lower and upper bounds of trajectory τ's timestamps, i.e., $\tau.t.lb = v_1.t$ and $\tau.t.ub = v_i.t$, and $n.t.lb$ and $n.t.ub$ are the lower and upper bounds (left and right boundaries) of $\mathrm{range}(n)$. For example, $\mathrm{range}(n_4) = [6:00, 8:00]$, and $n_4.t.lb = 6:00$ and $n_4.t.lb = 8:00$.

We define the temporal-similarity upper bound $\mathrm{Sim}_\mathrm{T}(\tau, \tau')$ in the following manner. Because $d(v_i.t, \tau') \geq d_T(\tau, n_b)$, we have that $\sum_{v_i \in \tau} \mathrm{e}^{-d(v_i.t,\tau')} \leq |\tau| \cdot \mathrm{e}^{-d_T(\tau,n_b)}$ and $\sum_{v_j \in \tau'} \mathrm{e}^{-d(v_j.t,\tau)} \leq |\tau'| \cdot \mathrm{e}^{-d_T(\tau',n_a)}$. By substituting them into Eq. 4, we have

$$\mathrm{Sim}'_\mathrm{T}(\tau, \tau').ub = \mathrm{e}^{-d_T(\tau,n_b)} + \mathrm{e}^{-d_T(\tau',n_a)}. \qquad (11)$$

By substituting Eq. 11 into Eq. 7, the global lower bound on the spatial similarity for trajectories in the third case is defined as follows.

$$\mathrm{Sim}_\mathrm{S}(\tau, \tau') \geq \frac{\theta - (1-\lambda)(\mathrm{Sim}'_\mathrm{T}(\tau, \tau').ub)}{\lambda} = LB'_S \qquad (12)$$

If $LB'_S > 2$, all trajectory pairs in the third case can be pruned. Otherwise, for a trajectory pair (τ, τ'), if its spatial-similarity upper bound $\mathrm{Sim}_\mathrm{S}(\tau, \tau').ub$ (Eq. 9) is less than the global lower bound LB'_S of the spatial similarity (Eq. 12), (τ, τ') is pruned. For the remaining trajectory pairs, we compute their exact spatiotemporal similarity and store the qualified pairs in n_c. As a result, all the qualified trajectory pairs in $[n_c.lb, n_c.ub]$ are found.

When merging non-leaf nodes (e.g., merging n_{19} and n_{20} to their parent node n_{22}), we propose an approach that aims to further prune the search space. Assume that τ is stored in n_d, that τ' is stored in n_e, and that n_d and n_e are descendant nodes of n_f and n_g (e.g., n_3 and n_8 are descendant nodes of n_{19} and n_{20}). According to Eqs. 11 and 7, the upper and lower bounds of temporal similarity are computed respectively as follows.

$$\mathrm{Sim}'_\mathrm{T}(\tau, \tau').ub = 2\mathrm{e}^{-d_T(n_d,n_e)}$$

$$\mathrm{Sim}_\mathrm{T}(\tau, \tau') \geq \frac{\theta - \lambda \cdot 2}{(1-\lambda)} = LB_T$$

$$\mathrm{Sim}'_\mathrm{T}(\tau, \tau').ub < LB_T \Leftrightarrow d_T(n_d, n_e) > \ln\left(\frac{2-2\lambda}{\theta - 2\lambda}\right)$$

Here, $d_T(n_d, n_e)$ is computed according to Eq. 10. If the minimum distance between n_d and n_e exceeds $\ln(\frac{2-2\lambda}{\theta-2\lambda})$, we prune all trajectory pairs $\{(\tau, \tau') |\ \text{range}(\emptyset) \subseteq \text{range}(n_d) \wedge \text{range}(\tau') \subseteq \text{range}(n_e)\}$.

The merging processes of different node pairs (e.g., n_3 and n_4, n_5, and n_6) at the same level of the tree are independent. Thus, we can again apply parallel processing. Having merged the computation results from the leaf nodes all the way to the root node, the solution in [0:00, 24:00] is found. Notice that during the merging phase, we only follow the partitioning imposed by the index to merge the data of each node; the index structure is not updated.

Algorithm 1: TF-Matching

```
Data: a grid indexing tree $T_g$, a trajectory set $P$, a threshold $\theta$
Result: $\{(\tau, \tau') | \text{Sim}_{ST}(\tau, \tau') \geq \theta, \forall \tau, \tau' \in P\}$
1  $H \leftarrow \emptyset$;
2  for each leaf node $n$ in $T_g$ do
3      compute $LB_S$;
4      for each trajectory pair $(\tau, \tau')$, range($\emptyset$) $\subseteq$ range($n$) and
       range($\emptyset'$) $\subseteq$ range(n) do
5          compute $\text{Sim}_S(\tau, \tau').ub$;
6          if $\text{Sim}_S(\tau, \tau').ub < LB_S$ then
7              prune $(\tau, \tau')$;
8          compute $\text{Sim}_{ST}(\tau, \tau')$;
9          if $\text{Sim}_{ST}(\tau, \tau') \geq \theta$ then
10             store $(\tau, \tau')$ in $n$;
11     $H.add(n)$;
12 while $H \neq \emptyset$ do
13     if $n, n' \in H$, $n.parent = n'.parent$ then
14         merge $n$, $n'$, and $n.parent$;
15         compute and store qualified trajectory pairs in
           $n.parent$;
16         $H.add(n.parent)$;
17         $H.remove(n)$;
18         $H.remove(n')$;
19     if $|H| = 1$ then
20         return all qualified trajectories;
```

The pseudocode of TF-Matching is shown in Algorithm 1. A set H is used to maintain the processed nodes of the current level of the tree, and the computation is bottom-up. Initially, for each leaf node n, we compute the global lower bound LB_S of the spatial similarity (Eq. 7) for trajectory pairs in n (lines 1–3). Then, for each trajectory pair (τ, τ') in n, we compute its spatial similarity upper bound (Eq. 9), and if its upper bound is less than LB_S, this pair is pruned (lines 4–7). Otherwise, we compute the exact spatiotemporal similarity of (τ, τ'), and if it is no less than θ, we store (τ, τ') in n (lines 8–10). Having refined all trajectory pairs in n, we add n to heap H (line 11). When all leaf nodes have been added to H, we merge the results from the leaf nodes toward the root node. If two nodes n and n' have the same parent node and their child nodes are not in H, we merge the results for n, n', and their parent node (e.g., n_3, n_4, and n_{14} in Fig. 2) and store the qualified trajectory pairs in $n.parent$. Next, we add $n.parent$ to H, and remove n and n' from H (lines 12–18). If $H = 1$, the root node is reached, and all qualified trajectory pairs are returned (lines 19–20).

3.1.5 Complexity analysis

Let $|P|$ denote the cardinality of trajectory set P, and let $|\tau_{avg}|$ denote the average number of samples in a trajectory in P. We use $|V|$ and $|E|$ to denote the numbers of vertices and edges in G. Then $O(|V| \log |V| + |E|)$ is the time complexity of computing the network distance between a vertex and a trajectory. TF-Matching follows the filter-and-refine paradigm, and the time complexity of the filter phase is $O((|V| \log |V| + |E|)|\tau_{avg}|\ |T_{sp}|)$, where T_{sp} is the set of scanned trajectory pairs. Notice that we compute the spatial upper bound (Eq. 9) for most trajectory pairs. Only in the third merging case (see Sect. 3.1.4), if two nodes n_d and n_e are sufficiently far apart temporally, trajectory pairs (τ, τ'), where $\tau \in n_d$ and $\tau' \in n_e$, can be pruned directly (i.e., it is unnecessary to visit these pairs). Next, we have $T_{sp} \cup T_{dp} = P^2$, where T_{dp} is the set of trajectory pairs pruned in the third merging case.

The time complexity of verifying candidates by computing their exact spatiotemporal similarities is $O((|V| \log |V| + |E|)|\tau_{avg}|\ |C|)$, where $|C|$ is the cardinality of the candidate set, $C \subseteq T_{sp} \subseteq P^2$. The total time complexity is $O((|V| \log |V| + |E|)|\tau_{avg}|\ |T_{sp}|) + O((|V| \log |V| + |E|)|\tau_{avg}|\ |C|) = O((|V| \log |V| + |E|)|\tau_{avg}|\ |T_{sp}|)$. In the worst case, i.e., $T_{dp} = \emptyset$, the time complexity is $O((|V| \log |V| + |E|)|\tau_{avg}|\ |P|^2)$.

We proceed to consider the case where $P \neq Q$. First, TF-Matching supports $P \neq Q$ directly. When computing the spatiotemporal similarity in leaf nodes and merging trajectories in different nodes, we only need to select trajectory pairs from P and Q. Let $|\tau'_{avg}|$ denote the average numbers of samples in trajectories in P and Q. Then the time complexity of TF-Matching is $O((|V| \log |V| + |E|)|\tau'_{avg}|\ |T'_{sp}|)$, where $T'_{sp} \cup T'_{dp} = P \times Q$. In the worst case, the time complexity is $O((|V| \log |V| + |E|)|\tau'_{avg}||P|\ |Q|)$.

The computations for nodes at the same tree level occur in parallel. We initially process the leaf nodes and then process $\lceil \log \alpha \rceil$ levels for merging, where α is the number of leaf nodes. Intuitively, given multiple cores and threads, it is possible to accelerate the computation at the leaf level by generating many leaf nodes and processing them in parallel. However, more leaf nodes also lead to more tree levels, which increases the merging cost.

3.2 *k*-TF-matching algorithm

3.2.1 Basic idea

We follow the framework of TF-Matching (Sect. 3.1) and propose a new k-TF-Matching algorithm as a baseline for computing the k-TS-Join. As for TF-Matching, we index the temporal domain using a hierarchical grid structure. We refine trajectory pairs in the same leaf node by computing their spatiotemporal similarities. By merging the results from the leaf nodes toward the root, the join result is obtained when the root is reached. Because the k-TS-Join does not have a threshold to prune the search space (unlike the Tb-TS-Join), we define new upper and lower bounds and rework the pruning in the same grid and the merging among different grids. We initially consider the self-join scenario (i.e., $P = Q$) and it is trivial to support $P \neq Q$. For the k-TF-Matching algorithm, when computing the spatiotemporal similarity in leaf nodes and merging trajectories in different nodes, we only need to select trajectory pairs from P and Q.

3.2.2 Pruning in leaf nodes

Initially, we randomly select a set P_k of k trajectory pairs and compute their similarities. The lowest similarity among them is used as the global top-k lower bound LB_k^{h-1} for the leaf level (h is the height of the tree and level $h-1$ is the leaf level).

$$LB_k^{h-1} = \min_{(\tau_i,\tau_i')\in P_k} \{\mathrm{Sim}_{\mathrm{ST}}(\tau_i, \tau_i')\}, \tag{13}$$

where $P_k = \{(\tau_1, \tau_1'), (\tau_2, \tau_2'), \ldots, (\tau_k, \tau_k')\}$.

Then we refine trajectory pairs in the same leaf node n by computing their spatiotemporal similarities. For a trajectory pair (τ_1, τ_2), if we know its exact spatiotemporal similarity, we put it in set P_s^n. We maintain a top-k heap $P_k^n \subseteq P_k \cup P_s^n$ such that $|P_k^n| = k$ and $\forall(\tau_1, \tau_2) \in P_k^n(\forall(\tau_1', \tau_2') \in P_k \cup P_s^n \backslash P_k^n(\mathrm{Sim}_{\mathrm{ST}}(\tau_1, \tau_2) \geq \mathrm{Sim}_{\mathrm{ST}}(\tau_1', \tau_2')))$. We define a global top-k lower bound LB_k^n of leaf node n.

$$LB_k^n = \min_{(\tau_1,\tau_2)\in P_k^n} \{\mathrm{Sim}_{\mathrm{ST}}(\tau_1, \tau_2)\} \tag{14}$$

The value of LB_k^n changes dynamically during query processing. Initially $P_s^n = \emptyset$ and $LB_k^n = LB_k^{h-1}$.

By substituting Eqs. 14 and 6 into Eq. 7 and by using LB_k^n to replace θ, we estimate the global top-k spatial lower bound LB_{Sk}^n of leaf node n as

$$\mathrm{Sim}_{\mathrm{S}}(\tau_1, \tau_2) \geq \frac{LB_k^n - 2\cdot(1-\lambda)}{\lambda} = LB_{Sk}^n, \tag{15}$$

where τ_1 and τ_2 are two trajectories in the same leaf node (see the example in Fig. 2). For each trajectory pair (τ_1, τ_2), we estimate its spatial upper bound $\mathrm{Sim}_{\mathrm{S}}(\tau_1, \tau_2).ub$ according to Eq. 9. If $\mathrm{Sim}_{\mathrm{S}}(\tau_1, \tau_2).ub < LB_{Sk}^n$, pair (τ_1, τ_2) can be pruned safely. For the remaining trajectory pairs, we compute their exact spatiotemporal similarities and put them in set P_s^n.

Notice that the computations in different leaf nodes are independent. Thus, we can perform these computations in parallel. For each leaf node n, we maintain its top-k heap P_k^n for the following merging operations. Because $|P_k^n| = k$, we can guarantee that no result pair is missing.

3.2.3 Merging

Having computed the spatiotemporal similarities of the trajectory pairs in the leaf nodes, we merge the results from the leaf nodes toward the root. We merge the top-k heaps of all leaf nodes and define a new top-k heap P_k' for level $h-2$ of the tree, such that $\forall(\tau_1, \tau_2) \in P_k'(\forall(\tau_1', \tau_2') \in \bigcup_i^m P_k^{n_i} \backslash P_k'(\mathrm{Sim}_{\mathrm{ST}}(\tau_1, \tau_2) \geq \mathrm{Sim}_{\mathrm{ST}}(\tau_1', \tau_2')))$, where n_i is a leaf node and $i \in [1, m]$. The value of the global top-k lower bound is updated as follows.

$$LB_k^{h-2} = \min_{(\tau_i,\tau_i')\in P_k'} \{\mathrm{Sim}_{\mathrm{ST}}(\tau_i, \tau_i')\} \tag{16}$$

By substituting Eq. 16 into Eqs. 14 and 15 and by using the value of LB_k^{h-2} to replace that of LB_k^{h-1}, we get the global top-k lower bound $LB_k^{n_c}$ and the global top-k spatial lower bound $LB_{Sk}^{n_c}$ of node n_c. The values of $LB_k^{n_c}$ and $LB_{Sk}^{n_c}$ change dynamically during query processing.

We merge two leaf nodes n_a and n_b to their parent node n_c (e.g., merging n_3, n_4 to n_{14} in Fig. 2). In addition to merging their top-k heaps, we also need to consider the trajectory pairs (τ, τ') according to the following three cases:

(1) $\mathrm{range}(\emptyset) \subseteq \mathrm{range}(n_a) \wedge \mathrm{range}(\emptyset') \subseteq \mathrm{range}(n_c)$
(2) $\mathrm{range}(\emptyset) \subseteq \mathrm{range}(n_b) \wedge \mathrm{range}(\emptyset') \subseteq \mathrm{range}(n_c)$
(3) $\mathrm{range}(\emptyset) \subseteq \mathrm{range}(n_a) \wedge \mathrm{range}(\emptyset') \subseteq \mathrm{range}(n_b)$

For the first two cases, we use the spatial-similarity upper bound $\mathrm{Sim}_{\mathrm{S}}(\tau_1, \tau_2).ub$ (Eq. 9) and the global top-k spatial lower bound $LB_{Sk}^{n_c}$ to prune the search space. For the remaining trajectory pairs, we compute their exact spatiotemporal similarity and store them in $P_s^{n_c}$. The top-k heap $P_k^{n_c}$ changes correspondingly.

For the third case, by substituting Eqs. 16 and 11 into Eq. 12 and by using the value of $LB_k^{n_c}$ to replace that of θ, the global lower bound on the spatial similarity for trajectories is defined as follows.

$$\begin{aligned} \mathrm{Sim}_{\mathrm{T}}'(\tau, \tau').ub &= \mathrm{e}^{-d_T(\tau,n_b)} + \mathrm{e}^{-d_T(\tau',n_a)} \\ \mathrm{Sim}_{\mathrm{S}}(\tau, \tau') &\geq \frac{LB_k^{n_c} - (1-\lambda)(\mathrm{Sim}_{\mathrm{T}}'(\tau, \tau').ub)}{\lambda} \\ &= LB_{Sk}^{n_c\prime} \end{aligned} \tag{17}$$

If $LB_{Sk}^{n_c\prime} > 2$, all trajectory pairs in the third case can be pruned. Otherwise, for a trajectory pair (τ, τ'), if its spatial-similarity upper bound $\mathrm{Sim}_{\mathrm{S}}(\tau, \tau').ub$ (Eq. 9) is less than the global spatial lower bound $LB_{Sk}^{n_c{}'}$ (Eq. 17), (τ, τ') is pruned. For the remaining trajectory pairs, we compute their exact spatiotemporal similarity and store them in $P_s^{n_c}$. The top-k heap $P_k^{n_c}$ changes correspondingly.

When merging non-leaf nodes (e.g., merging n_{19} and n_{20} into their parent node n_{22} in Fig. 2), we propose an approach that aims to further prune the search space. Assume that τ is stored in n_d, that τ' is stored in n_e, that n_d and n_e are descendant nodes of n_f and n_g, and that n_h is the parent node of n_f and n_g (e.g., n_3 and n_8 are descendant nodes of n_{19} and n_{20}, and n_22 is the parent node of n_{19} and n_{20}). According to Eqs. 11 and 14, the upper and lower bounds of temporal similarity are computed as follows.

$$\mathrm{Sim_T}(\tau, \tau').ub = 2\mathrm{e}^{-d_T(n_d,n_e)}$$

$$\lambda \cdot \mathrm{Sim_S}(\tau_1, \tau_2) + (1-\lambda) \cdot \mathrm{Sim_T}(\tau_1, \tau_2) \geq LB_k^{n_h}$$

$$\Rightarrow \mathrm{Sim_T}(\tau, \tau') \geq \frac{LB_k^{n_h} - \lambda \cdot 2}{(1-\lambda)} = LB_T^{n_h}$$

$$\mathrm{Sim_T}(\tau, \tau').ub < LB_T^{n_h} \Leftrightarrow d_T(n_d, n_e) > \ln\left(\frac{2-2\lambda}{LB_T^{n_h} - 2\lambda}\right)$$

Thus, if the minimum distance between n_d and n_e exceeds $\ln(\frac{2-2\lambda}{LB_k^{n_h}-2\lambda})$, we prune all trajectory pairs $\{(\tau, \tau') | \mathrm{range}(\emptyset) \subseteq \mathrm{range}(n_d) \wedge \mathrm{range}(\tau') \subseteq \mathrm{range}(n_e)\}$.

The merging processes of different node pairs (e.g., pair n_3 and n_4, and pair n_5 and n_6) at the same level of the tree are independent. Thus, we can again apply parallel processing. Having merged the results from the leaf nodes all the way to the root node, the top-k solution is found.

The pseudocode of k-TF-Matching is shown in Algorithm 2. The computation is bottom-up, and h is the current level of computation. Initially, we randomly select k trajectory pairs and compute their spatiotemporal similarities. The minimum value among them is used as the global lower bound LB_k^h (Eq. 13) (lines 1–2). For each trajectory pair (τ, τ') in the same leaf node n, we compute its spatial similarity upper bound (Eq. 9), and if its upper bound is less than LB_{Sk}^n, this pair is pruned (lines 4–7). Otherwise, we compute the exact spatiotemporal similarity of (τ, τ'), and if it exceeds LB_{Sk}^n, we put it in P_k^n and update the values of LB_k^n and LB_{Sk}^n (lines 8–10). Having computed all nodes in level h, we merge the top-k heaps of all nodes at level h and compute the value of LB_k^{h-1} (Eq. 16). If two nodes n_1 and n_2 at level h have the same parent node n_3, we merge the results for n_1, n_2, and n_3 by computing $P_k^{n_3}$, LB_k^n, and LB_{Sk}^n (lines 11–15). If $h - 1 = 0$, we reach the root n_3, and the top-k heap $P_k^{n_3}$ is returned. Otherwise, we compute level $h - 1$ (lines 16–18).

Algorithm 2: k-TF-Matching

```
   Data: a grid indexing tree $T_g$, a trajectory set $P$
   Result: top-$k$ trajectory pairs
1  $h \leftarrow T_g.hight - 1$;
2  compute $LB_k^h$;
3  for each leaf node $n$ in $T_g$ do
4      for each trajectory pair $(\tau, \tau')$, where
       range($\emptyset$) $\subseteq$ range($n$) and range($\emptyset'$) $\subseteq$ range(n) do
5          compute $\mathrm{Sim_S}(\tau, \tau').ub$;
6          if $\mathrm{Sim_S}(\tau, \tau').ub < LB_{Sk}^n$ then
7              prune $(\tau, \tau')$;
8          compute $\mathrm{Sim_{ST}}(\tau, \tau')$;
9          if $\mathrm{Sim_{ST}}(\tau, \tau') \geq LB_{Sk}^n$ then
10             update $P_k^n$, $LB_k^n$, and $LB_{Sk}^n$;
11 while true do
12     compute $\bigcup_{n \in levelh} P_k^n$ and $LB_k^{h-1}$;
13     if $n_1, n_2 \in level\ h$, $n_1.parent = n_2.parent = n_3$ then
14         merge $n_1$, $n_2$, and $n_3$;
15         compute $P_k^{n_3}$, $LB_k^n$, and $LB_{Sk}^n$;
16     if $h - 1 = 0$ then
17         return $P_k^{n_3}$;
18     $h \leftarrow h - 1$;
```

3.2.4 Complexity

Let P_θ denote the scanned trajectory set for each trajectory search, and let $|T'_{sp}|$ denote the cardinality of the scanned trajectory pairs. The time complexity of the k-two-phase algorithm is $O((|V| \log |V| + |E|)\tau_{avg}||T'_{sp}|)$, where $T'_{sp} \subseteq P \times P$ for self join, and $T'_{sp} \subseteq P \times Q$ for non-self join. The detailed procedure is the same as Sect. 3.1.5.

Correctness: The k-TF-Matching algorithm follows the "filter-and-refine" paradigm. We define a spatial upper bound $\mathrm{Sim_S}(\tau, \tau').ub$ and a global top-k lower bound LB_{Sk}^n (cf. Eqs. 9 and 15) to prune the search space in each node. When $LB_{Sk}^n > \mathrm{Sim_S}(\tau, \tau').ub$, pair (τ, τ') is pruned. It is clear that the pruned pairs cannot be a solution because their upper bounds are less than the global lower bound. Next, we refine the candidates by computing their exact similarities, and we find the result by merging the top-k results of all nodes. Because (1) the trajectory pairs pruned cannot be a solution, (2) the computation in the refinement is exact, and (3) the global top-k result is a subset of the union of the top-k results of all nodes, the k-TF-Matching algorithm computes a correct solution to the k-TS-Join.

4 Two-phase search

We propose a two-phase and a k-two-phase algorithms to compute the TB-TS-Join and k-TS-Join efficiently.

4.1 Two-phase algorithm

4.1.1 Basic idea

TF-Matching has three main drawbacks. First, it is driven by the temporal domain and so has weak spatial pruning power. The algorithm has to process a large number of trajectory pairs, which adversely affects the performance. Second, more leaf nodes (more threads) lead to a higher merging cost, which counts against parallel processing. Third, it may need additional computation to acquire network distances to compute spatial similarities (Eqs. 1 and 3), again decreasing performance.

To process the Tb-TS-Join more efficiently, we develop a two-phase algorithm based on a divide-and-conquer strategy (see Fig. 3a). (1) In the trajectory-search phase, for each trajectory $\tau \in P$, we explore the spatial and temporal domains concurrently and search for trajectories close to τ. In the spatial domain, network expansion [9] from each trajectory sample point is used to explore the spatial network, while in the temporal domain, we expand the search from each timestamp of τ. An upper bound on the spatiotemporal similarity is defined to enable pruning of the search space in the spatial and temporal domains. Moreover, a heuristic scheduling strategy is proposed to schedule multiple so-called query sources (sample points in the spatial domain, and timestamps in the temporal domain) effectively, which aims to further enhance efficiency. Compared to TF-Mathcing, the two-phase algorithms have a stronger pruning power. The search process of different trajectories is independent, so the trajectory searches can be processed in parallel. In addition, the network distances for the similarity computation can be derived directly during the trajectory-search processes. (2) In the merging phase, we combine the computation results of all trajectories and find the solution to the Tb-TS-Join. In contrast to TF-Matching, the merging cost is now uncorrelated to the thread count. The two-phase algorithm has better time complexity than the temporal-first matching algorithm.

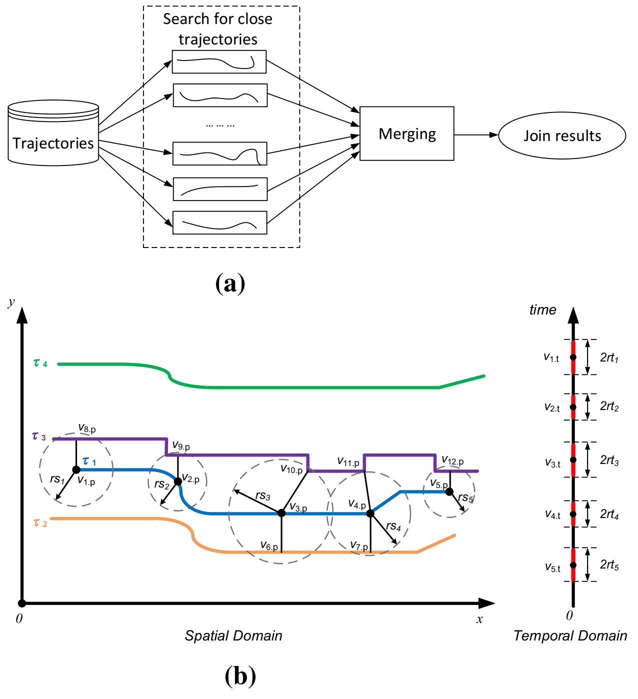


Fig. 3 An example of the two-phase algorithm. **a** Parallel mechanism, **b** trajectory search

4.1.2 Expansion search

Consider the example in Figure 3b, where τ_1, τ_2, τ_3, and τ_4 are trajectories, and we search for the trajectories close to τ_1 in the spatial and temporal domains (τ_1 is the "query trajectory"). Trajectory $\tau_1 = \langle v_1, v_2, \ldots, v_5\rangle$, sample points $\{v_6, v_7\} \in \tau_2$, and $v_6.p$ and $v_7.p$ are the samples closest to $v_3.p$ and $v_4.p$. Sample points $\{v_8, v_9, \ldots, v_{12}\} \in \tau_3$, and $v_8.p$, $v_9.p$,…,$v_{12}.p$ are the samples closest to $v_1.p$, $v_2.p$,…,$v_5.p$.

In the spatial domain, network expansion is performed from each sample point $v_i.p \in \tau_1$ using Dijkstra's algorithm [9]. The explored space is a "circular" region $(v_i.p, rs_i)$, where the radius rs_i is the network distance from the center $v_i.p$ to the expansion boundary. Dijkstra's algorithm always selects the vertex with the minimum distance label for expansion (initially $rs_i = 0$). Hence, if $v'.p \in \tau'$ is the first sample point scanned by the expansion from $v.p$, $v'.p$ is the sample point closest to $v.p$, i.e., $d(v.p, \tau') = sd(v.p, v'.p)$. For example, in Figure 3(b), $d(v_3.p, \tau_2) = sd(v_3.p, v_6.p)$, and $d(v_4.p, \tau_2) = sd(v_4.p, v_7.p)$.

In the temporal domain, we expand the search from each timestamp $v_i.t \in \tau_1$. The explored space is a time range $[v_i.t - rt_i, v_i.t + rt_i]$, where rt_i is the radius of the range. Initially $rt_i = 0$, and then it is increased by one second (the minimum scale of the time axis) at each time, step by step, to form a larger scanned range until the targets are found. Similar to the Dijkstra's algorithm, if $v'.t \in \tau'$ is the first timestamp scanned by the expansion from $v.t$, $v'.t$ is the timestamp closest to $v.t$, i.e., $d(v.t, \tau') = |v.t - v'.t|$.

If a trajectory τ is scanned by the expansions from all sample points in τ_1, we compute the spatial similarity of (τ, τ_1) according to Eq. 3; this type of trajectory is called "spatially fully scanned," e.g., τ_3. If a trajectory is scanned by the expansions from a part of sample points in τ_1, it is called "spatially partly scanned," e.g., τ_2. If a trajectory is unscanned by the expansions from any sample points in τ_1, it is called "spatially unscanned," e.g., τ_4. Similarly, in the temporal domain, such trajectories are called "temporally fully scanned," "temporally partly scanned," and "temporally unscanned."

4.1.3 Upper bound computation

If a trajectory is spatially partly scanned (e.g., τ_2 in Fig. 3b) or spatially unscanned (e.g., τ_4), for a sample point $v_i.p \in \tau_1$, the lower bound on network distance between $v_i.p$ and τ_2 is defined as follows.

$$d(v_i.p, \tau_2) \geq d(v_i.p, \tau_2).lb = \begin{cases} sd(v_i.p, v_i'.p) & \text{if Case 1} \\ rs_i & \text{if Case 2} \end{cases} \tag{18}$$

Case 1: τ_2 has been scanned by the expansion from $v_i.p$, and $v_i'.p \in \tau_2$ is the closest point to $v_i.p$.
Case 2: τ_2 has not been scanned by the expansion from $v_i.p$.

By substituting Eq. 18 into Eq. 8, for any sample point $v_i'.p \in \tau_1$, we have that

$$d(v_i'.p, \tau_2) \geq \min_{v_i \in \tau_1}\{d(v_i.p, \tau_2).lb\}. \tag{19}$$

Then we merge Eqs. 18 and 19 into Eq. 3, and the spatial similarity upper bound $\text{Sim}_\text{S}(\tau_1, \tau_2).ub$ is derived.

$$\begin{aligned} \sum_{v_i \in \tau_1} \mathrm{e}^{-d(v_i.p,\tau_2)} &\leq \sum_{v_i \in \tau_1} \mathrm{e}^{-d(v_i.p,\tau_2).lb} \\ \sum_{v_i' \in \tau_2} \mathrm{e}^{-d(v_i'.p,\tau_1)} &\leq |\tau_2| \cdot \min_{v_i \in \tau_1}\{d(v_i.p, \tau_2).lb\} \\ &\Rightarrow \text{Sim}_\text{S}(\tau_1, \tau_2).ub \\ &= \frac{\sum_{v_i \in \tau_1} \mathrm{e}^{-d(v_i.p,\tau_2).lb}}{|\tau_1|} \\ &\quad + \mathrm{e}^{-\min_{v_i \in \tau_1}\{d(v_i.p,\tau_2).lb\}} \end{aligned} \tag{20}$$

Similarly, in the temporal domain, if a trajectory τ_2 is temporally partly scanned or temporally unscanned, for a timestamp $v_i.t \in \tau_1$, the lower bound on the distance between $v_i.t$ and τ_2 is defined as follows.

$$d(v_i.t, \tau_2) \geq d(v_i.t, \tau_2).lb = \begin{cases} |v_i.t - v_i'.t| & \text{if Case 3} \\ rt_i & \text{if Case 4} \end{cases} \tag{21}$$

Case 3: τ_2 has been scanned by the expansion from $v_i.t$, and $v_i'.t \in \tau_2$ is the point closest to $v_i.t$.
Case 4: τ_2 has not been scanned by the expansion from $v_i.t$.

We then extend Lemma 1 (Eq. 8) to apply to the temporal domain. Specifically, by substituting Eq. 21 into Eq. 8, for any sample point $v_i'.t \in \tau_2$, we have that

$$d(v_i'.t, \tau_2) \geq \min_{v_i \in \tau_1}\{d(v_i.t, \tau_2).lb\}. \tag{22}$$

Then, we merge Eqs. 21 and 22 into Eq. 4, and the temporal similarity upper bound $\text{Sim}_\text{T}(\tau_1, \tau_2).ub$ is derived.

$$\begin{aligned} \sum_{v_i \in \tau_1} \mathrm{e}^{-d(v_i.t,\tau_2)} &\leq \sum_{v_i \in \tau_1} \mathrm{e}^{-d(v_i.t,\tau_2).lb} \\ \sum_{v_i' \in \tau_2} \mathrm{e}^{-d(v_i'.t,\tau_1)} &\leq |\tau_2| \cdot \min_{v_i \in \tau_1}\{d(v_i.t, \tau_2).lb\} \\ &\Rightarrow \text{Sim}_\text{T}(\tau_1, \tau_2).ub \\ &= \frac{\sum_{v_i \in \tau_1} \mathrm{e}^{-d(v_i.t,\tau_2).lb}}{|\tau_1|} \\ &\quad + \mathrm{e}^{-\min_{v_i \in \tau_1}\{d(v_i.t,\tau_2).lb\}} \end{aligned} \tag{23}$$

Next, we combine the spatial and temporal similarity upper bounds (Eqs. 20 and 23). Thus, if a trajectory τ_2 is not both spatially and temporally fully scanned, we compute the upper bound on the spatiotemporal similarity $\text{Sim}_\text{ST}(\tau_1, \tau_2).ub$ as follows.

$$\begin{aligned} &\text{Sim}_\text{ST}(\tau_1, \tau_2).ub \\ &\quad = \lambda \cdot \text{Sim}_\text{S}(\tau_1, \tau_2).ub + (1-\lambda) \cdot \text{Sim}_\text{T}(\tau_1, \tau_2).ub \end{aligned} \tag{24}$$

For all partly scanned trajectories, we define a global upper bound UB as follows.

$$UB = \max_{\tau_2 \in P_{ps}}\{\text{Sim}_\text{ST}(\tau_1, \tau_2).ub\}, \tag{25}$$

where $P_{ps} \subseteq P$ is the current set of partly scanned trajectories. The value of UB changes during query processing.

If a trajectory is unscanned in both the spatial and temporal domains, we do not maintain its spatiotemporal similarity upper bound to reduce the computation and storage costs. Assume that trajectory τ_1 is the query trajectory, τ_2 is partly scanned, and τ_4 is unscanned in both domains. According to Eqs. 18 and 21, we have that $\forall v_i \in \tau_1(d(v_i.p, \tau_2).lb \leq d(v_i.p, \tau_4).lb)$ and $\forall v_i \in \tau_1(d(v_i.t, \tau_2).lb \leq d(v_i.t, \tau_4).lb)$.

Referring to Eqs. 20, 23, and 24, we have $\text{Sim}_\text{ST}(\tau_1, \tau_2).ub \geq \text{Sim}_\text{ST}(\tau_1, \tau_4).ub$. Therefore, $\text{Sim}_\text{ST}(\tau_1, \tau_4).ub$ cannot be the global upper bound UB, and it is not necessary to maintain the spatiotemporal similarity upper bound on τ_4.

4.1.4 Scheduling strategy

We propose a heuristic strategy to schedule the expansions from different sample points and timestamps (so-called query sources) in the spatial and temporal domains in order to make the search focus on trajectories that are most likely to be in the result.

Assume $\tau = \langle v_1, v_2, \ldots, v_m \rangle$ is the query trajectory. We give each query source $q \in \{v_1.p, v_2.p, \ldots, v_m.p\} \cup$

$\{v_1.t, v_2.t, \ldots, v_m.t\}$ a priority label q.label and maintain a heap H of descending order on the value of q.label on the query sources. The values of priority labels change during the search in the two domains. We search the top-ranked query source until a new query source takes its place. The priority label is defined as follows.

$$q.\text{label} = \sum_{\tau' \in P_{ps} \setminus q.s} \{\text{Sim}_{\text{ST}}(\tau, \tau').ub\} \tag{26}$$

Here, $P_{ps} \subseteq P$ is the set of spatially and temporally partly scanned trajectories, and $q.s$ is the set of trajectories that have been scanned from query source q. For example, in Fig. 3b, τ_1 is a query trajectory and $v_1.p, v_2.p, \ldots, v_5.p$ are query sources in the spatial domain. We have that $v_1.p.s = \{\tau_3\}$, $v_2.p.s = \{\tau_3\}$, $v_3.p.s = \{\tau_2, \tau_3\}$, $v_4.p.s = \{\tau_2, \tau_3\}$, and $v_5.p.s = \{\tau_3\}$. Trajectory τ_2 is spatially partly scanned, τ_3 is spatially fully scanned and temporally partly scanned, and τ_4 is temporally partly scanned. Thus, $P_{ps} = \{\tau_2, \tau_3, \tau_4\}$. For query source $v_1.p.s$, $P_{ps} \setminus v_1.p.s = \{\tau_2, \tau_3, \tau_4\} \setminus \{\tau_3\} = \{\tau_2, \tau_4\}$, and for query source $v_3.p.s$, $P_{ps} \setminus v_3.p.s = \{\tau_2, \tau_3, \tau_4\} \setminus \{\tau_2, \tau_3\} = \{\tau_4\}$.

The priority label represents the significance of a query source during search. The main goal of the scheduling strategy is to transform trajectories from "partly scanned" to "fully scanned" as soon as possible [17,18]. Thus, the priority $q.s$ of a query source should be proportional to its "margin," i.e., the size of $P_{ps} \setminus q.s$. For example, in Fig. 3b, $P_{ps} \setminus v_1.p.s = \{\tau_2, \tau_4\}$; thus, the margin of $v_1.p$ is 2. Moreover, a trajectory with a higher spatiotemporal-similarity upper bound (Eq. 24) is more likely to be the solution. So, $\forall \tau \in P_{ps} \setminus q.s$, the value of $\text{Sim}_{\text{ST}}(\tau_1, \tau).ub$ is proportional to the priority of query source q.

4.1.5 Filter, refine, and merging

If the global upper bound UB of the partly scanned trajectories is smaller than the value of threshold θ, the expansion in the spatial and temporal domains terminates, and all trajectories that are not fully scanned in the two domains can be pruned safely. For each fully scanned trajectory τ, we have the exact values of $d(v_i.p, \tau)$ and $d(v_i.t, \tau)$ for all sample points v_i in τ_1; thus, we can further refine the spatial, temporal, and spatiotemporal upper bounds (refer to Eqs. 20, 23, and 24).

We place all fully scanned trajectories in a candidate set $C(\tau_1)$ for trajectory τ_1. For each trajectory $\tau \in C(\tau_1)$, (τ, τ_1) is a potential qualified trajectory pair. For (τ_1, τ), we maintain a parameter defined as follows.

$$V(\tau_1, \tau) = \frac{\sum_{v_i \in \tau_1} \mathrm{e}^{-d(v_i.p, \tau)}}{|\tau_1|} + \frac{\sum_{v_i \in \tau_1} \mathrm{e}^{-d(v_i.t, \tau)}}{|\tau_1|}$$

Notice that the value of $V(\tau_1, \tau)$ can be derived from Eqs. 20 and 23 directly.

Having processed the nearest neighbor searches for all trajectories in P, we merge the results. For each trajectory $\tau \in P$, we maintain a candidate set $C(\tau)$. For a trajectory pair (τ_1, τ_2), if $\tau_1 \in C(\tau_2)$ and $\tau_2 \in C(\tau_1)$, we compute their exact spatiotemporal similarity:

$$\begin{aligned} \text{Sim}_{\text{ST}}(\tau_1, \tau_2) &= V(\tau_1, \tau_2) + V(\tau_2, \tau_1) \\ &= \frac{\sum_{v_i \in \tau_1} \mathrm{e}^{-d(v_i.p, \tau_2)}}{|\tau_1|} + \frac{\sum_{v_i \in \tau_1} \mathrm{e}^{-d(v_i.t, \tau_2)}}{|\tau_1|} \\ &\quad + \frac{\sum_{v_j \in \tau_2} \mathrm{e}^{-d(v_j.p, \tau_1)}}{|\tau_2|} + \frac{\sum_{v_j \in \tau_2} \mathrm{e}^{-d(v_j.t, \tau_1)}}{|\tau_2|} \end{aligned} \tag{27}$$

Then we compare $\text{Sim}_{\text{ST}}(\tau_1, \tau_2)$ to threshold θ. If $\text{Sim}_{\text{ST}}(\tau_1, \tau_2) \geq \theta$, (τ_1, τ_2) is a qualified pair. Otherwise, we prune it. For other cases, i.e., $\tau_1 \notin C(\tau_2)$ or $\tau_2 \notin C(\tau_1)$, (τ_1, τ_2) cannot be a qualified trajectory, and it is pruned.

The two-phase algorithm is based on a divide-and-conquer strategy. First, for each trajectory τ in P, we retrieve the trajectories spatiotemporally close to τ. The trajectory-search phase is detailed in Algorithm 3. Since the search processes for different trajectories are independent, we can process the searches in parallel. Second, we merge the results of the individual searches, i.e., candidate sets, to obtain the final result. Unlike for TF-Matching, the merging cost of the two-phase algorithm is uncorrelated to the thread count. The merging process is detailed in Algorithm 4.

In Algorithm 3, the query arguments are a trajectory τ and a threshold θ, and the query result is a candidate set for τ. Initially, we select the top-ranked item q from heap H as the current-search query source. Then we search using q. Each newly scanned trajectory τ' (τ' has not been scanned by the expansion from q) is added to a scanned trajectory set $q.s$. If τ' is unscanned, we also add it to the partly scanned trajectory set P_{ps} (lines 1–9). Next, we update the spatiotemporal similarity upper bound $\text{Sim}_{\text{ST}}(\tau, \tau').ub$ (refer to Eq. 24). If τ' is not fully scanned in the two domains, then if $\text{Sim}_{\text{ST}}(\tau, \tau').ub > UB$, we update the value of UB to that of $\text{Sim}_{\text{ST}}(\tau, \tau').ub$ (lines 10–13). If τ' is fully scanned, we remove it from P_{ps}. If $\text{Sim}_{\text{ST}}(\tau, \tau').ub$ was used as UB before, we also update the value of UB. If $\text{Sim}_{\text{ST}}(\tau, \tau').ub \geq \theta$, we add τ' to the candidate set for τ (lines 14–18). If $UB < \theta$, the query returns the candidate set $C(\tau)$ (lines 19–20). If q is not the top-ranked query source in H, we update it so that this is the case (lines 21–22).

Algorithm 4 merges the candidate sets iteratively. For each trajectory τ' in $C(\tau)$, we check whether τ belongs to $C(\tau')$. If so, we compute the exact spatiotemporal similarity $\text{Sim}_{\text{ST}}(\tau, \tau')$ (refer to Eq. 27), and then we remove τ from $C(\tau')$. If $\text{Sim}_{\text{ST}}(\tau, \tau') \geq \theta$, we add the pair (τ, τ') to result set A. Finally, result set A is returned.

Algorithm 3: Trajectory Search Algorithm

```
Data: a trajectory $\tau$ and a threshold $\theta$
Result: candidate set $C(\tau)$
1  $H \leftarrow \{v_1.p, v_2.p, \ldots, v_{|\tau|}.p\} \cup \{v_1.t, v_2.t, \ldots, v_{|\tau|}.t\}$;
2  $\forall q \in H$ ($q.label \leftarrow 0$), $UB \leftarrow 0$;
3  $q \leftarrow H.top$;
4  while true do
5      search($q$);
6      for each newly scanned trajectory $\tau'$ do
7          $q.s.add(\tau')$;
8          if $\tau' \notin P_{ps}$ then
9              $P_{ps}.add(\tau')$;
10         update $\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau').ub$;
11         if $\tau'$ is not fully scanned then
12             if $\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau').ub > UB$ then
13                 $UB \leftarrow \mathrm{Sim}_{\mathrm{ST}}(\tau, \tau').ub$;
14         if $\tau'$ is fully scanned then
15             $P_{ps}.remove(\tau)$;
16             update $UB$;
17             if $\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau').ub \geq \theta$ then
18                 $C(\tau).add(\tau')$
19         if $UB < \theta$ then
20             return $C(\tau)$;
21     if $q \neq H.top$ then
22         $q \leftarrow H.top$;
```

Algorithm 4: Merging Algorithm

```
Data: $\{C(\tau)|\forall\tau \in P\}, \theta$
Result: $A = \{(\tau, \tau')|\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau') \geq \theta, \forall\tau, \tau' \in P\}$
1  for each trajectory $\tau$ in $P$ do
2      for each $\tau'$ in $C(\tau)$ do
3          if $\tau \in C(\tau')$ then
4              compute $\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau')$;
5              $C(\tau').remove(\tau)$;
6              if $\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau') \geq \theta$ then
7                  $A.add(\tau, \tau')$;
8              else
9                  break;
10 return $A$;
```

4.1.6 Complexity analysis

Let P_θ denote the set of scanned, partly or fully, trajectories for each trajectory search. Let $|\tau_{avg}|$ denote the average number of samples in a trajectory in P. Then $O(|V|\log|V|+|E|)$ is the time complexity of computing the network distance between a sample point and a trajectory by using Dijkstra's algorithm [9].

According to Eqs. 20, 23, and 24, the maximum spatial and temporal expansion radiuses rs and rt are inversely proportional to threshold θ. Assuming the trajectories are uniformly distributed in the spatial and temporal domains, it follows that $|P_\theta|$ is inversely proportional to threshold θ. Thus, $|P_\theta|$ is sensitive to the value of threshold θ and the pruning effectiveness.

The time complexity of the trajectory search phase is $O(((|V|\log|V|+|E|)|\tau_{avg}|+|P_\theta|)|P|) = O((|V|\log|V|+|E|)|\tau_{avg}|\ |P| + |P|\ |P_\theta|)$. For each trajectory, the trajectory search complexity is $O((|V|\log|V|+|E|)|\tau_{avg}|)$, and $|P_\theta|$ is the number of scanned trajectories. There are $|P|$ trajectories in total. The time complexity of the merging phase is $O(|P||C|)$, where $|C|$ is the cardinality of the candidate set for each trajectory. Since $C \subseteq P_\theta \subseteq P$, the time complexity of the two-phase algorithm is $O((|V|\log|V| + |E|)|\tau_{avg}|\ |P|+|P|\ |P_\theta|)+O(|P|\ |C|) = O((|V|\log|V|+|E|)|\tau_{avg}|\ |P|+|P|\ |P_\theta|)$. If θ is sufficiently large, the time complexity is close to $O((|V|\log|V|+|E|)|\tau_{avg}|\ |P|)$.

We proceed to consider the case where $P \neq Q$. The two-phase algorithm conducts trajectory searches and maintains candidate sets for all trajectories in P and Q. The time complexity of the trajectory-search phase is $O((|V|\log|V| + |E|)|\tau_{avg}|\ |P| + |P|\ |P_\theta| + (|V|\log|V| +|E|)|\tau_{avg}|\ |Q| + |Q|\ |Q_\theta|) = O((|V|\log|V| + |E|)|\tau_{avg}|\ (|P| + |Q|) + |P|\ |P_\theta| + |Q|\ |Q_\theta|)$. For the merging phase, the time complexity remains $O(|P|\ |C_p|)$ (or $O(|Q|\ |C_q|)$), $C_p \subseteq Q_\theta \subseteq Q$, and $C_q \subseteq P_\theta \subseteq P$. The time complexity of the two-phase algorithm is then $O((|V|\log|V| + |E|)|\tau_{avg}|(|P| + |Q|) + |P|\ |P_\theta| + |Q|\ |Q_\theta| + |P|\ |C_p|) = O((|V|\log|V| + |E|)|\tau_{avg}|(|P| + |Q|) + |P|\ |P_\theta| + |Q|\ |Q_\theta|)$, which is sensitive to the pruning effectiveness.

In the worst case, the time complexity of the two-phase algorithm is $O((|V|\log|V|+|E|)|\tau_{avg}|\ |P|+|P|^2)$, which is better than that of TF-Matching, which is $O(|V|\log|V|+|E|)|\tau_{avg}||P|^2)$. In addition, the superiority of the two-phase algorithm stems from the following two points.

First, in the filter phase, TF-Matching has to visit and compute spatial upper bounds for the most trajectory pairs, including all trajectory pairs in leaf nodes. Only in the third merging case (see Sect. 3.1.4), if two nodes n_d and n_e are sufficiently far apart temporally, trajectory pairs (τ, τ'), where $\tau \in n_d$ and $\tau' \in n_e$, can be pruned directly (i.e., these pairs need not be visited). All other trajectory pairs must be visited for bound computation. The pruning in TF-Matching simply serves to save computations when computing exact similarities between pairs of trajectories—the algorithm still needs to visit most of them. Next, in the two-phase algorithm, $|P_\theta|$ is sensitive to the value of threshold θ and the pruning effectiveness. Here, it is not necessary to compute bounds for unscanned trajectories (no need to visit them). Moreover, TF-Matching is driven by the temporal domain and has weak spatial pruning power, while the two-phase algorithm exploits effective spatiotemporal bounds. The resulting pruning effectiveness is shown in Tables 3, 4, 5 and 6 (Sects. 6.2.1 and 6.3.1).

Second, TF-Matching only partially supports parallel processing, i.e., only the computations for the nodes at the same tree level can be processed in parallel. Initially, we process the leaf nodes and then process $\lceil log(\alpha) \rceil$ upper levels for merging, where α is the number of leaf nodes. Intuitively, given multiple cores and threads, it is possible to accelerate the computation at the leaf level by generating many leaf nodes and processing them in parallel. However, more leaf nodes also yields more tree levels, which increases the merging cost (the computation is done at each tree level, and there are $(\lceil log(\alpha) \rceil + 1)$ levels). In contrast, the trajectory-search processes of the two-phase algorithm are independent of each other, and the merging cost is constant (uncorrelated to the number of threads used for parallel processing). Therefore, the trajectory-search processes of the two-phase algorithm can be performed fully in parallel (the computation is conducted only at one time).

4.2 *k*-two-phase algorithm

4.2.1 Basic idea

k-TF-Matching has similar drawbacks to TF-Matching: weak spatial pruning power, a higher merging cost, and additional computation to acquire network distances to compute spatial similarities. To process the k-TS-Join more efficiently, we thus follow the framework of the two-phase algorithm (Sect. 4.1) and develop a new k-two-phase algorithm (see Fig. 3a). We define a pair of new upper and lower bounds to prune the search space effectively in the spatial and temporal domains. The network distances for the similarity computation can be derived directly during the trajectory-search processes. The search process of different trajectories is independent, so the trajectory searches can be processed in parallel. In contrast to k-TF-Matching, it is not necessary to compute the spatiotemporal similarity of k randomly selected trajectories (refer to Eq. 13) in the k-two-phase algorithm. In addition, the merging cost of k-two-phase is uncorrelated to the thread count. The k-two-phase algorithm has better time complexity than the k-TF-Matching algorithm.

We initially consider the self-join scenario (i.e., $P = Q$) and it is trivial to support $P \neq Q$. We only need to conduct trajectory searches for all trajectories in P and Q and to maintain candidate sets for all of them.

4.2.2 Lower bound

Let $|P|$ denote the number of trajectories in set P, and let m denote the number of threads. Then each thread will process $\lceil \frac{|P|}{m} \rceil$ or $\lceil \frac{|P|}{m} \rceil - 1$ trajectory searches. Trajectory search is performed in the spatial and temporal domains (see Sect. 4.1.2). Assuming that τ is a "query trajectory" (e.g., τ_1 in Fig. 3b) then if a trajectory τ' is fully scanned in the spatial and temporal domains (e.g., τ_3 in Fig. 3b), we compute its spatial lower bound $\mathrm{Sim}_\mathrm{S}(\tau, \tau').lb$ as follows.

$$\begin{aligned} \mathrm{Sim}_\mathrm{S}(\tau, \tau') &= \frac{\sum_{v_i \in \tau} \mathrm{e}^{-d(v_i.p, \tau')}}{|\tau|} + \frac{\sum_{v_j \in \tau'} \mathrm{e}^{-d(v_j.p, \tau)}}{|\tau'|} \\ &\text{and } \frac{\sum_{v_j \in \tau'} \mathrm{e}^{-d(v_j.p, \tau)}}{|\tau'|} > 0 \\ \Rightarrow \mathrm{Sim}_\mathrm{S}(\tau, \tau') &> \frac{\sum_{v_i \in \tau} \mathrm{e}^{-d(v_i.p, \tau')}}{|\tau|} = \mathrm{Sim}_\mathrm{S}(\tau, \tau').lb \end{aligned} \tag{28}$$

$$\begin{aligned} \mathrm{Sim}_\mathrm{T}(\tau, \tau') &= \frac{\sum_{v_i \in \tau} \mathrm{e}^{-d(v_i.t, \tau')}}{|\tau|} + \frac{\sum_{v_j \in \tau'} \mathrm{e}^{-d(v_j.t, \tau)}}{|\tau'|} \\ &\text{and } \frac{\sum_{v_j \in \tau'} \mathrm{e}^{-d(v_j.t, \tau)}}{|\tau'|} > 0 \\ \Rightarrow \mathrm{Sim}_\mathrm{T}(\tau, \tau') &> \frac{\sum_{v_i \in \tau} \mathrm{e}^{-d(v_i.t, \tau')}}{|\tau|} = \mathrm{Sim}_\mathrm{T}(\tau, \tau').lb, \end{aligned} \tag{29}$$

where v_i and v_j are sample points in trajectories τ and τ'. By combining the spatial and temporal lower bounds, the spatiotemporal lower bound is defined as follows.

$$\mathrm{Sim}_\mathrm{ST}(\tau, \tau').lb = \lambda \cdot \mathrm{Sim}_\mathrm{S}(\tau, \tau').lb + (1-\lambda) \cdot \mathrm{Sim}_\mathrm{T}(\tau, \tau').lb \tag{30}$$

For each thread c, we store fully scanned trajectory pairs (e.g., (τ, τ')) in set P_f. We maintain a top-k heap P_k^c to contain the trajectory pairs with top-k lower bounds such that $|P_k^c| = k$ and $\forall(\tau_i, \tau_i') \in H(\forall(\tau_j, \tau_j') \in P_f \backslash P_k^c(\mathrm{Sim}_\mathrm{ST}(\tau_i, \tau_i').lb > \mathrm{Sim}_\mathrm{ST}(\tau_j, \tau_j').lb))$. We define a global spatiotemporal lower bound LB_c of thread c as follows.

$$LB_c = \min_{(\tau, \tau') \in P_k^c} \{\mathrm{Sim}_\mathrm{ST}(\tau, \tau').lb\}, \tag{31}$$

where LB_c changes dynamically during query processing. Notice that LB_c is only valid when $|H| = k$, to guarantee that no solution is missing.

For partly scanned trajectories, we compute their global upper bound UB according to Eq. 25. If the value of UB is less than that of LB_c, the expansions in the spatial and temporal domains terminate, and all trajectories that are not fully scanned in the two domains can be pruned safely. Then, we merge the results according to the approach of the two-phase algorithm (see Sect. 4.1.5).

The k-two-phase algorithm is based on a divide-and-conquer strategy. For each trajectory τ in P, we retrieve the trajectories spatiotemporally close to τ. Because the search processes for different trajectories are independent, we can process the searches in parallel. The trajectory search in the same thread is detailed in Algorithm 5. The merging phase

Algorithm 5: Trajectory Search in A Thread

```
Data: query trajectories in thread $c$;
Result: candidate sets $C(\tau), \forall\tau \in c$
1  $P_k^c \leftarrow \emptyset$; $LB_c \leftarrow 0$; $LB \leftarrow LB_k$;
2  for each query trajectory $\tau \in c$ do
3      while true do
4          search($\tau$);
5          $\tau'$ is a newly scanned trajectory;
6          if $\tau'$ is not fully scanned then
7              if $\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau').ub > UB$ then
8                  $UB \leftarrow \mathrm{Sim}_{\mathrm{ST}}(\tau, \tau').ub$;
9          if $\tau'$ is fully scanned then
10             update $UB$;
11             compute $\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau').lb$;
12             if $\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau').lb > LB_c$ then
13                 update $P_k^c$, and $LB_c$;
14             if $|P_k^c| = k$ then
15                 if $\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau').ub \geq LB_c$ then
16                     $C(\tau).add(\tau')$
17         if $UB < LB_c$ then
18             break;
19 return candidate sets $C(\tau), \forall\tau \in c$
```

is the same as that of the two-phase algorithm (see Algorithm 4).

In Algorithm 5, the query arguments are the query trajectories in thread c, and the query results are the candidate sets for all query trajectories. Initially, the top-k heap P_k^c is set to $\emptyset$, LB_c is set to 0, and the value of LB is set to that of LB_k (line 1). We search each query trajectory τ in thread c. For each newly scanned trajectory τ', we compute its spatiotemporal upper bound $\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau').ub$, and if its value exceeds that of UB, UB is updated to $\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau').ub$ (Eqs. 24 and 25) (lines 2–8). If τ' is fully scanned, we update the value of UB and compute the value of $\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau').lb$ (Eq. 30). If $\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau').lb$ exceeds LB_c, we update top-k set P_k^c (lines 9–13). If the size of P_k^c reaches k, LB_c is valid. If $\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau').ub \geq LB_c$, we add τ' to the candidate set for τ (lines 14–16). If $UB < LB_c$, search process for query trajectory τ terminates (lines 17–18). Finally, candidate sets $C(\tau), \forall\tau \in c$ are returned (line 19).

4.2.3 Complexity

Let P_θ denote the scanned trajectory set for each trajectory search, and let $|C|$ denote the cardinality of the candidate set for each query trajectory. The time complexity of the k-two-phase algorithm is $O((|V|\log|V|+|E|)|\tau_{avg}||P|+|P||P_s|)$ for self join, and is $O((|V|\log|V|+|E|)|\tau_{avg}|(|P|+|Q|)+|P||P_s|+|Q||Q_s|)$ for non-self join, where P_s and Q_s are set of scanned trajectories for each trajectory search.

Correctness Similar to the k-TF-Matching algorithm, the k-two-phase algorithm follows the "filter-and-refine" paradigm. We define a global upper bound UB and a global lower bound LB_c (cf. Eqs. 25 and 31) of the spatiotemporal similarity to prune the search space. When $LB_c > UB$, the search terminates. It is clear that not fully-scanned trajectories cannot be a solution because their global upper bound is less than the global lower bound of the fully scanned trajectories, meaning that they can be pruned safely. Second, we refine the candidates by computing their exact similarities, and we obtain the result by merging the top-k results of all threads. Because (1) the trajectory pairs pruned in the filtering cannot be in the result, (2) the computation in the refinement is exact, and (3) the global top-k result is a subset of the union of the top-k results of all threads, the k-two-phase algorithm computes the k-TS-Join correctly.

5 Extension

We first propose a new sequence similarity measure that takes the visiting sequence of sample points into account when matching trajectories. Then we extend the TF-Matching, k-TF-Matching, two-phase, and k-two-phase algorithms to support the new measure.

5.1 Sequence similarity measure

Given trajectories $\tau_1 = \langle v_1, v_2, \ldots, v_m\rangle$ and $\tau_2 = \langle v_1, v_2, \ldots, v_n\rangle$, the spatial and temporal aggregate distances $S_{dist}(\tau_1, \tau_2)$ and $T_{dist}(\tau_1, \tau_2)$ (taking the visiting sequence of trajectory sample points into account) [7,17,18] from τ_1 to τ_2 are defined as follows.

$$S_{dist}(\tau_1, \tau_2) = \max \begin{cases} e^{-sd(\tau_1.head.p, \tau_2.head.p)} \\ +S_{dist}(\tau_1.tail, \tau_2) \\ \\ S_{dist}(\tau_1, \tau_2.tail) \end{cases} \tag{32}$$

$$T_{dist}(\tau_1, \tau_2) = \max \begin{cases} e^{-|\tau_1.head.t - \tau_2.head.t|} \\ +T_{dist}(\tau_1.tail, \tau_2) \\ \\ T_{dist}(\tau_1, \tau_2.tail) \end{cases} \tag{33}$$

Here $*.head$ is the first sample point of $*$, (e.g., $\tau_1.head = v_1$) and $*.tail$ is the trajectory obtained by removing the head from $*$. (e.g., $\tau_1.tail = \langle v_2, v_3, \ldots, v_m\rangle$). The distances $S_{dist}(\tau_1, \tau_2)$ and $T_{dist}(\tau_1, \tau_2)$ are asymmetrical.

By combining the spatial distances $S_{dist}(\tau_1, \tau_2)$ and $S_{dist}(\tau_2, \tau_1)$ and the temporal distances $T_{dist}(\tau_1, \tau_2)$ and $T_{dist}(\tau_2, \tau_1)$, we define the spatial and temporal similarities $\mathrm{Sim}_{\mathrm{S}}^{\mathrm{o}}(\tau_1, \tau_2)$ and $\mathrm{Sim}_{\mathrm{T}}^{\mathrm{o}}(\tau_1, \tau_2)$ as follows.

$$\mathrm{Sim}_{\mathrm{S}}^{\mathrm{o}}(\tau_1, \tau_2) = \frac{S_{dist}(\tau_1, \tau_2)}{|\tau_1|} + \frac{S_{dist}(\tau_2, \tau_1)}{|\tau_2|} \tag{34}$$

$$\mathrm{Sim}_{\mathrm{T}}^{\mathrm{o}}(\tau_1, \tau_2) = \frac{T_{dist}(\tau_1, \tau_2)}{|\tau_1|} + \frac{T_{dist}(\tau_2, \tau_1)}{|\tau_2|} \tag{35}$$

These similarity measures are symmetrical, i.e., $\mathrm{Sim}_{\mathrm{S}}^{\mathrm{o}}(\tau_1, \tau_2) = \mathrm{Sim}_{\mathrm{S}}^{\mathrm{o}}(\tau_2, \tau_1)$ and $\mathrm{Sim}_{\mathrm{T}}^{\mathrm{o}}(\tau_1, \tau_2) = \mathrm{Sim}_{\mathrm{T}}^{\mathrm{o}}(\tau_2, \tau_1)$.

By substituting Eqs. 34 and 35 into Equation 5, the spatiotemporal sequence similarity $\mathrm{Sim}_{\mathrm{ST}}^{\mathrm{o}}(\tau_1, \tau_2)$ that takes the visiting sequence into account is defined as follows.

$$\mathrm{Sim}_{\mathrm{ST}}^{\mathrm{o}}(\tau_1, \tau_2) = \lambda \cdot \mathrm{Sim}_{\mathrm{S}}^{\mathrm{o}}(\tau_1, \tau_2) + (1 - \lambda) \cdot \mathrm{Sim}_{\mathrm{T}}^{\mathrm{o}}(\tau_1, \tau_2) \tag{36}$$

Our search framework can support all aggregate-distance-based similarity measures in spatial networks, including BCT [7], NNT [21], network-based BCT [17], and network-based spatiotemporal BCT with a sequence [18], because the framework is based on network expansion and uses bounds calculated using network distances. Variants of these similarity measures, such as sequenced BCT or NNT, can also be supported. Notice that these similarity measures cannot be used in the TS-Join directly because (1) the original BCT and NNT are based on Euclidean space, and (2) all of them are asymmetrical. We extend them into spatial networks and make them symmetrical according to Eqs. 3 and 4.

5.2 Temporal-first matching

5.2.1 TF-matching algorithm

For each leaf node n, we compute the spatial similarity $\mathrm{Sim}_{\mathrm{S}}^{\mathrm{o}}(\tau_1, \tau_2)$ for each trajectory pair $(\tau_1, \tau_2) \in n$ (Eq. 34). By substituting the value of $\mathrm{Sim}_{\mathrm{S}}^{\mathrm{o}}(\tau_1, \tau_2)$ into Eq. 7, we have that

$$\mathrm{Sim}_{\mathrm{S}}^{\mathrm{o}}(\tau, \tau') \geq \frac{\theta - (1 - \lambda) \cdot 2}{\lambda} = LB_S^o \tag{37}$$

If $\mathrm{Sim}_{\mathrm{S}}^{\mathrm{o}}(\tau_1, \tau_2) < LB_S^o$, trajectory pair (τ_1, τ_2) is pruned. Otherwise, we compute the exact spatiotemporal similarity $\mathrm{Sim}_{\mathrm{ST}}^{\mathrm{o}}(\tau_1, \tau_2)$ and compare its value to threshold θ.

The TF-Matching algorithm that takes into account the visiting sequence of trajectory sample points is obtained by applying Eqs. 32–37 in Algorithm 1.

5.2.2 *k*-TF-matching

Initially, we randomly select a set P_k of k trajectory pairs and compute their similarities. We then use the minimum similarity as the global top-k lower bound LB_k^{h-1}. By substituting Eq. 36 into Eq. 14, we have that

$$LB_k^n = \min_{(\tau_1, \tau_2) \in P_k^n} \{\mathrm{Sim}_{\mathrm{ST}}^{\mathrm{o}}(\tau_1, \tau_2)\} \tag{38}$$

We use the value of LB_k^n to replace that of θ in Eq. 37 and have that

$$\mathrm{Sim}_{\mathrm{S}}^{\mathrm{o}}(\tau, \tau') \geq \frac{LB_k^n - (1 - \lambda)(\mathrm{Sim}_{\mathrm{T}}^{\mathrm{o}}(\tau, \tau').ub)}{\lambda} = LB_S^{o'} \tag{39}$$

For each leaf node n, we compute the spatial similarity $\mathrm{Sim}_{\mathrm{S}}^{\mathrm{o}}(\tau_1, \tau_2)$ for each trajectory pair $(\tau_1, \tau_2) \in n$ (Eq. 34). If $\mathrm{Sim}_{\mathrm{S}}^{\mathrm{o}}(\tau_1, \tau_2) < LB_S^{o'}$, pair (τ_1, τ_2) is pruned. Otherwise, we compute the exact spatiotemporal similarity $\mathrm{Sim}_{\mathrm{ST}}^{\mathrm{o}}(\tau_1, \tau_2)$ and compare its value to threshold LB_k^n.

The k-TF-Matching algorithm that takes into account the visiting sequence of trajectory sample points is obtained by applying Eqs. 32–39 in Algorithm 2.

5.3 Two-phase search

5.3.1 Two-phase algorithm

Given two trajectory sample points $v_i \in \tau$ and $v_j \in \tau'$, the lower bounds of the network and temporal distances between $v_i.p$ and $v_j.p$ are defined as follows.

$$sd(v_i.p, v_j.p).lb = \begin{cases} sd(v_i.p, v_j.p) & \text{if Case 5} \\ rs_i & \text{if Case 6} \end{cases} \tag{40}$$

Case 5: v_j has been scanned by the expansion from $v_i.p$.
Case 6: v_j has not been scanned by the expansion from $v_i.p$.

$$d(v_i.t, v_j.t).lb = \begin{cases} |v_i.t - v_j.t| & \text{if Case 7} \\ rt_i & \text{if Case 8} \end{cases} \tag{41}$$

Case 7: v_j has been scanned by the expansion from $v_i.t$.
Case 8: v_j has not been scanned by the expansion from $v_i.t$.

By substituting Eqs. 40 and 41 into Eqs. 32 and 33, we have the upper bounds on the spatial and temporal aggregate distances $S_{dist}(\tau_1, \tau_2)$ and $T_{dist}(\tau_1, \tau_2)$.

$$S_{dist}(\tau_1, \tau_2).ub = \max \begin{cases} \mathrm{e}^{-sd(\tau_1.head.p, \tau_2.head.p).lb} \\ +S_{dist}(\tau_1.tail, \tau_2) \\ \\ S_{dist}(\tau_1, \tau_2.tail) \end{cases} \tag{42}$$

$$T_{dist}(\tau_1, \tau_2) = \max \begin{cases} \mathrm{e}^{-d(v_i.t, v_j.t).lb} \\ +T_{dist}(\tau_1.tail, \tau_2) \\ \\ T_{dist}(\tau_1, \tau_2.tail) \end{cases} \tag{43}$$

By substituting Eqs. 42 and 43 into Eqs. 24 and 25, the upper bound on spatiotemporal similarity $\mathrm{Sim}^{o}_{\mathrm{ST}}(\tau_1, \tau_2).ub$ and the global upper bound UB^{o} are derived.

The two-phase algorithm that takes into account the visiting sequence of sample points is obtained by applying Eqs. 32–36 and 40–43 in Algorithms 3 and 4.

5.3.2 *k*-two-phase algorithm

By submitting Eqs. 34 and 35 into Eqs. 28, 29, and 30, the lower bounds on the spatial, temporal, and spatiotemporal similarities are obtained as follows.

$$\mathrm{Sim}^{o}_{\mathrm{S}}(\tau, \tau').lb = \frac{S_{dist}(\tau_1, \tau_2)}{|\tau_1|} \tag{44}$$

$$\mathrm{Sim}^{o}_{\mathrm{T}}(\tau, \tau').lb = \frac{T_{dist}(\tau_1, \tau_2)}{|\tau_1|} \tag{45}$$

$$\mathrm{Sim}^{o}_{\mathrm{ST}}(\tau, \tau').lb = \lambda \cdot \mathrm{Sim}^{o}_{\mathrm{S}}(\tau, \tau').lb + (1-\lambda) \cdot \mathrm{Sim}^{o}_{\mathrm{T}}(\tau, \tau').lb \tag{46}$$

The *k*-two-phase algorithm that takes into account the visiting sequence of sample points is obtained by applying Eqs. 32–36 and 40–46 in Algorithms 4 and 5.

6 Experimental study

We report on experiments with real trajectory data that offer insight into the properties of the developed algorithms.

6.1 Settings

We use two spatial networks, namely the Beijing Road Network (BRN) and the New York Road Network (NRN)[11], which contain 28,342 vertices and 27,690 edges, and 95,581 vertices and 260,855 edges, respectively. The graphs are stored using adjacency lists. In BRN, we use a real taxi trajectory data set collected by the T-drive project [24,25] [12], while in NRN, we use a real taxi trajectory data set from New York[11]. Each trajectory in NRN denotes a taxi trip, and their average length (number of vertices) is ~ 80. The original trajectories in BRN are very long, often lasting days. We divide these trajectories into hour-long sub-trajectories, giving them an average length of ~ 72. The intent is to create trips with a realistic length and duration.

In the experiments, the indexing structure of the two TF-Matching algorithms (cf. Sect. 3) and the spatial networks of the two two-phase algorithms (when running Dijkstra's expansion [9], cf. Sect. 4) are memory resident, as the memory occupied are 42 MB and 57 MB for BRN and 51 MB and 68 MB for NRN. Trajectories are also memory resident for both algorithms, and they occupy 506 MB for BRN and 3.9 GB for NRN. All algorithms are implemented in Java and run on a cluster with 10 data nodes. Each node is equipped with two Intel® Xeon® Processors E5-2620 v3 (2.4GHz) and 128GB RAM. To account for the case where the trajectory data does not fit in main memory, we also consider a disk-resident approach and report its performance in Fig. 5 (Figs. 4, and 6, 7, 8, 9 concern the memory-based algorithms).

For the TF-Matching algorithms, for each node we store the ids (entries) of the trajectories that overlap the timespan indicated by the node. For the two-phase algorithms, for each vertex in the network, we store the ids (entries) of the trajectories that contain the vertex. The ids in each node are stored in ascending order in an ArrayList. We use a B+-tree to index trajectories on their ids. When we visit a node/vertex, we first traverse the corresponding ArrayList and retrieve the ids of the trajectories stored in the node/vertex. Next, we traverse the B+-tree and load all of the pages that contain the trajectories stored in the node/vertex. To improve the loading efficiency, we use a 1GB LRU buffer to store the retrieved pages.

Unless stated otherwise, experimental results are averaged over 10 independent trails using different query inputs. The main performance metrics are runtime and the number of trajectory visits. The number of trajectory visits (during query processing) is used as a metric since it reflects the number of data accesses. Since a trajectory may be visited several times, the number of trajectory visits may exceed the value of $|P|$ or $|Q|$. In multi-threaded executions, the runtime is the maximum runtime among all individual threads.

We study the performance of the non-self joins, i.e., $P \neq Q$, in Sects. 6.2.2–6.2.5 and 6.3.2–6.2.5 and of self joins in Sects. 6.2.5–6.2.6 and 6.2.5–6.3.6. Trajectories in P and Q are selected randomly from real data sets. The parameter settings are listed in Table 2. Because computing network distances online is time-consuming, we pre-computed the all-pair shortest path distances in the graph (for the TF-Matching algorithms only, not for the two-phase algorithms). The precomputation is processed in parallel with 120 threads. For BRN, the computation runtime is ~ 5 min, and the storage space of computation results is ~ 4 G. For NRN, the computation runtime is $\sim$2 h, and the storage space of computation results is $\sim$40 G. We denote the accelerated TF-Matchings (Sect. 3) by "TF-A" and "*k*-TF-A" in subsequent figures. The two-phase algorithms (Sect. 4) are denoted by "two-phase" and "*k*-two-phase," and the two-phase algorithms without the heuristic scheduling strategy are denoted by "two-phase-w/o-h" and "*k*-two-phase-w/o-h."

By default, the grid indexes in TF-Matching and *k*-TF-Matching employ a uniform leaf node partitioning scheme.

[11] https://publish.illinois.edu/dbwork/open-data/.

[12] https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample/.

Table 2 Parameter settings

	NRN	BRN
Trajectory cardinality $\|P\|$	1,000,000–10,000,000/default 1,000,000	50,000–200,000/ default 100,000
Trajectory cardinality $\|Q\|$	500,000–2,000,000/default 500,000	25,000–100,000/ default 50,000
Threshold θ	1.3–1.9/default 1.9	1.3–1.9/default 1.9
Preference parameter λ	0.1–0.9/default 0.5	0.1–0.9/default 0.5
Thread count m	24–144/default 24 for the Tb-TS-Join, 72–144/default 120 for the k-TS-Join	24–144/default 24
k	10–50/ default 10	10–50/ default 10

We conduct experiments to find the best such partitioning. When a leaf node is set to 1 h in BRN and to 15 min in NRN, and each node (including leaf and non-leaf nodes) contains at most 6560 trajectories in BRN and at most 15,265 trajectories in NRN, the index performs the best. We also consider a balanced partitioning scheme, where each leaf node contains the same or similar numbers of trajectories. When the index contains 32 leaf nodes in BRN and 196 leaf nodes in NRN, and each leaf node contains 1032 trajectories in BRN and 3875 trajectories in NRN on average, the index performs the best. The algorithms using the balanced partitioning method are denoted by "TF-A-balance" and "k-TF-A-balance." According to the experimental results in Figs. 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 and 14, the performance of TF-Matching is improved by around 20% when using balanced partitioning.

Table 3 Pruning effectiveness for non-self Tb-TS-Join

	TF	Two-phase	Two-phase-w/o-h
Candidate ratio (BRN)	0.17	0.10	0.14
Pruning ratio (BRN)	0.83	0.90	0.86
Candidate ratio (NRN)	0.12	0.04	0.06
Pruning ratio (NRN)	0.88	0.96	0.94

Table 4 Pruning effectiveness for self Tb-TS-Join

	TF	two-phase	two-phase-w/o-h
Candidate ratio (BRN)	0.11	0.06	0.09
Pruning ratio (BRN)	0.89	0.94	0.91
Candidate ratio (NRN)	0.08	0.03	0.04
Pruning ratio (NRN)	0.92	0.97	0.96

6.2 Performance of the Tb-TS-Join

6.2.1 Pruning effectiveness

First, we study the pruning effectiveness of the algorithms using the default settings. The experimental results are shown in Tables 3 (non-self join) and 4 (self join), with the reported candidate and pruning ratios defined as follows.

$$\begin{aligned} Candidate\ ratio &= \begin{cases} \frac{2|C|}{|P|^2} & \text{if self join} \\ \frac{|C|}{|P||Q|} & \text{if non}-\text{self join} \end{cases} \\ Pruning\ ratio &= 1 - Candidate\ ratio, \end{aligned} \tag{47}$$

where C is the candidate set. Comparing the pruning and candidate ratios of TF-Matching to those of the two-phase algorithm, we see that the candidate ratio of the two-phase algorithm is only 54.5–58.8% of that of TF-Matching and that, with the help of the heuristic scheduling strategy, the candidate ratio is improved by a factor of 40–50%.

6.2.2 Effect of trajectory cardinalities

Figure 4 shows the effect of trajectory cardinalities $|P|$ and $|Q|$ on the performance of the algorithms. Intuitively, a larger $|P|$ (or $|Q|$) causes more trajectory pairs to be processed (refer to the complexity analysis in Sects. 3.1.5 and 4.1.6), meaning that the runtime and the number of trajectory visits are expected to be higher for all algorithms. We see that the two-phase algorithm outperforms TF-Matching (TF-A and TF-A-balance) by almost an order of magnitude; and we see that the heuristic strategy can further improve the two-phase algorithm by almost a factor of 50% in terms of both runtime and the number of trajectory visits. The two-phase algorithm is able to process 1 M trajectories ($|P| = 1$ M and $|Q| = 0.5$M) in 38 s and 10 M trajectories ($|P| = 10$ M and $|Q| = 0.5$ M) in 252 s on the default 24 threads (see Fig. 4b).

The runtime is not fully aligned with the number of trajectory visits because the algorithms expend computational effort on maintaining the bounds used to prune the search

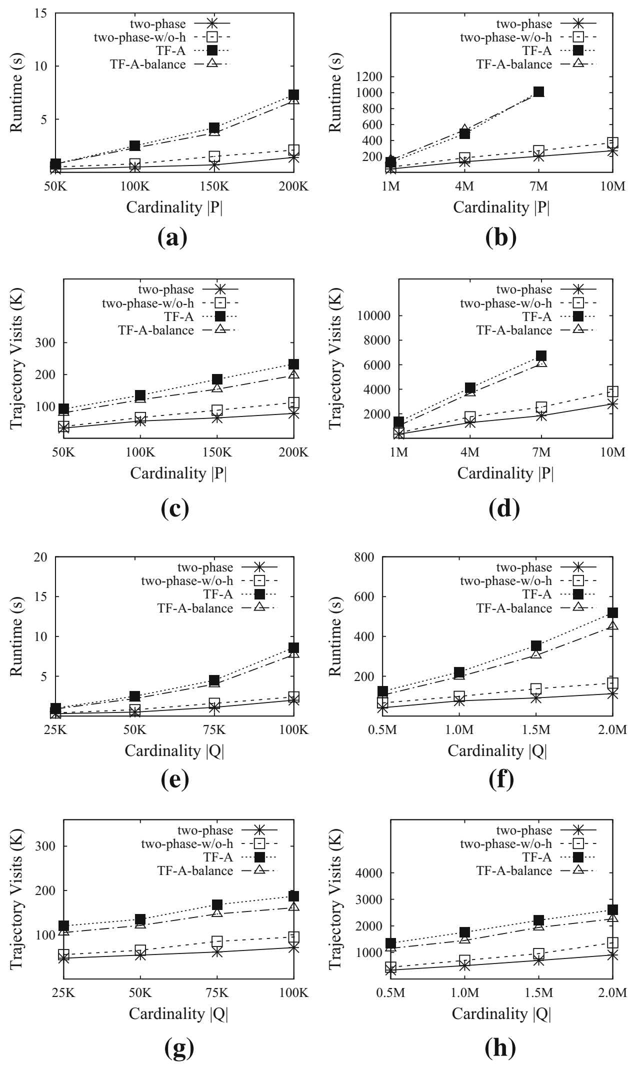


Fig. 4 Effect of trajectory cardinalities $|P|$ and $|Q|$. **a** BRN, **b** NRN, **c** BRN, **d** NRN, **e** BRN, **f** NRN, **g** BRN, **h** NRN

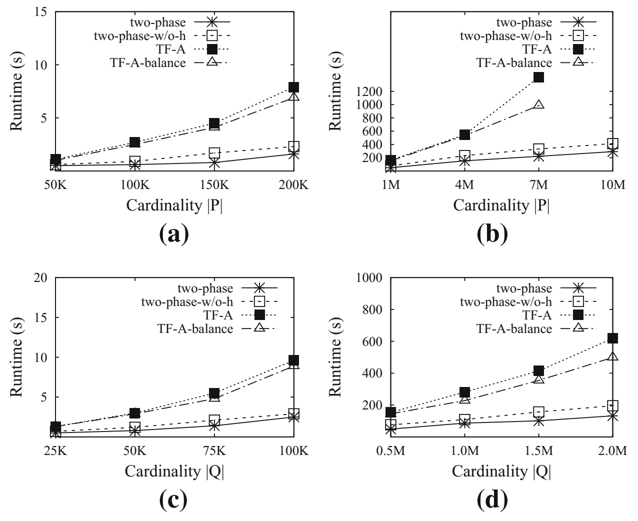


Fig. 5 Effect of disk-based storage. **a** BRN, **b** NRN, **c** BRN, **d** NRN

space. The resulting cost may offset the benefits of the reduction in the number of trajectory visits. In particular, the filter phase of TF-Matching computes and maintain bounds for almost all trajectory pairs.

Figure 5 shows the performance of the disk-based algorithms. Their performance patterns are similar to those of the memory-based algorithms (Fig. 4). The disk-based algorithms may need longer runtime because of disk I/O, but the query can still be processed in reasonable runtime (e.g., processing 10 M $\times$ 0.5 M trajectories on the default 24 threads in 300 s, see Fig. 5b). Notice that the number of trajectory visits is independent of where the data are stored.

6.2.3 Effect of threshold θ

Next, we vary the threshold θ in Fig. 6. For the two-phase algorithm, a larger θ leads to higher pruning effectiveness (refer to Eq. 4.1.6). Thus, the larger θ becomes, the smaller the search space becomes, and the required runtime and the number of trajectory visits are expected to decrease correspondingly. When $\theta = 1.9$, the two-phase algorithm is able to process 1 M trajectories ($|P| = 1$ M and $|Q| = 0.5$ M) in 38 s. In TF-Matching, a larger θ does not help prune the search space (refer to Sect. 3.1.5), and only slightly fewer trajectories are visited when θ increases. In contrast, a larger θ is useful in reducing the similarity computation (see Eq. 7). Thus, the runtime of TF-A and TF-A-balance decrease when θ increases.

6.2.4 Effect of λ

Figure 7 shows the effect of varying the preference parameter λ on efficiency. Parameter λ enables adjusting the relative preference of spatial and temporal similarity (see Eq. 5).

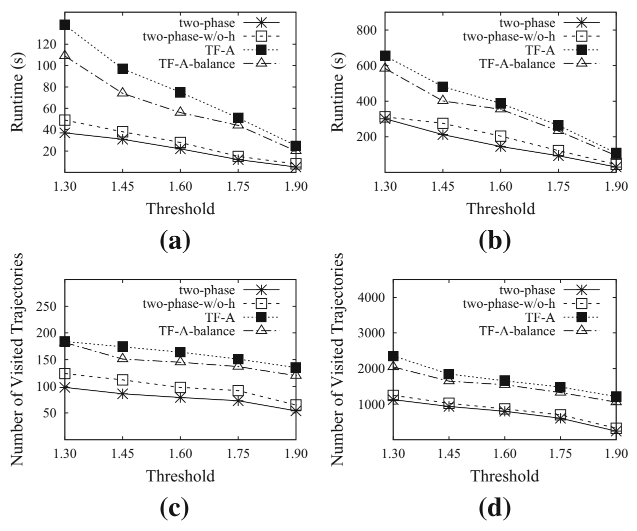


Fig. 6 Effect of threshold θ. **a** BRN, **b** NRN, **c** BRN, **d** NRN

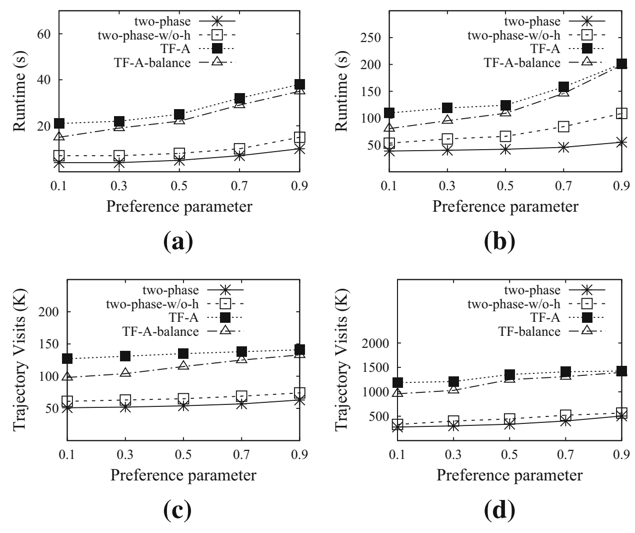


Fig. 7 Effect of λ. **a** BRN, **b** NRN, **c** BRN, **d** NRN

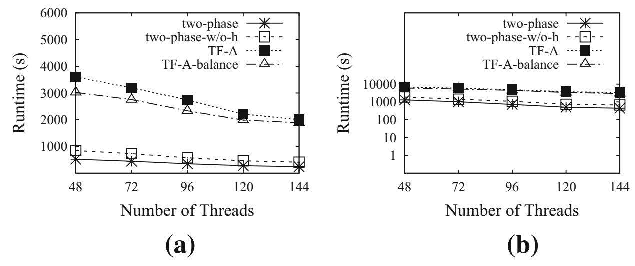


Fig. 8 Effect of thread count m. **a** Non-self join, **b** self join

When $\lambda = 1$, the TS-Join is in the spatial domain only, and when $\lambda = 0$, only temporal similarity is considered. Fig. 7 shows that the spatial domain needs more search effort than the temporal domain.

6.2.5 Effect of thread count m

We study the effect of thread count m on the efficiency of the algorithms using large trajectory data sets in NRN ($|P| = 10$ M and $|Q| = 2$ M for non-self Tb-TS-Join and $|P| = 10$ M for self Tb-TS-Join). The results are shown in Fig. 8.

We see that the two-phase algorithm outperforms TF-Matching by almost an order of magnitude in term of runtime. For the non-self join, the two-phase algorithm is able to process 10 M × 2 M trajectories with 144 threads in 220 s, while for the self join, the two-phase algorithm is able to process 10 M × 10 M trajectories with 144 threads in 480 s.

In Fig. 8, we increase the thread count m from 48 to 144 (3 times). This improves the runtime of the two-phase algorithm by a factor of 2.3–2.6, while the runtime of TF-Matching (TF-A and TF-A-balance) is improved by a factor of around 1.9. The main reason for the smaller improvement is that more threads (more leaf nodes) lead to a higher merging cost (cf. Sect. 3.1.5).

6.2.6 Performance of the self Tb-TS-Join

Figure 9 shows the runtime and number of trajectory visits for the self Tb-TS-Join when varying the trajectory cardinality, the similarity threshold, and the preference parameter. The trends of the figures are similar to those of the non-self Tb-TS-Join. The two-phase algorithm outperforms TF-Matching (TF-A and TF-A-balance) by almost an order of magnitude in terms of both runtime and the number of trajectory visits, and the heuristic search strategy improves the efficiency by almost a factor of 50%.

6.3 Performance of the *k*-TS-Join

6.3.1 Pruning effectiveness

We study the pruning effectiveness of the algorithms using the default settings. The experimental results are shown in Tables 5 (non-self k-TS-Join) and 6 (self k-TS-Join). Comparing the pruning and candidate ratios of k-TF-Matching to those of the k-two-phase algorithm, we see that the candidate ratio of the k-two-phase algorithm is only 47.3–63.6% of that of k-TF-Matching and that, with the help of the heuristic scheduling strategy, the candidate ratio is improved by a factor of 21.4–38.8%. The pruning effectiveness of k-TS-Join is a little bit weaker than that of Tb-TS-Join (cf. Tables 3 and 4) because k-TS-Join has no user-specified threshold to help the pruning.

6.3.2 Effect of k

Figure 10 shows the effect of k on the performance of the algorithms. Intuitively, a larger k causes a larger candidate set and more trajectory pairs to be processed, meaning that the runtime and the number of trajectory visits are expected to increase for all algorithms. We see that k-two-phase outperforms k-TF-Matching (k-TF-A and k-TF-A-balance) by almost an order of magnitude; and we see that the heuristic strategy can improve the two-phase algorithm by almost a factor of 40% in terms of both runtime and the number of trajectory visits.

6.3.3 Effects of $|P|$, $|Q|$, and λ

Figure 11 shows the effects of $|P|$, $|Q|$, and λ on the performance of the k-TS-Join. The trends are similar to those of the Tb-TS-Join, and it is evident that the k-two-phase algorithm has a clear advantage over the other algorithms. The k-two-phase algorithm is able to process 1 M trajectories ($|P| = 1$ M and $|Q| = 0.5$ M) in 142 s and 10 M trajectories

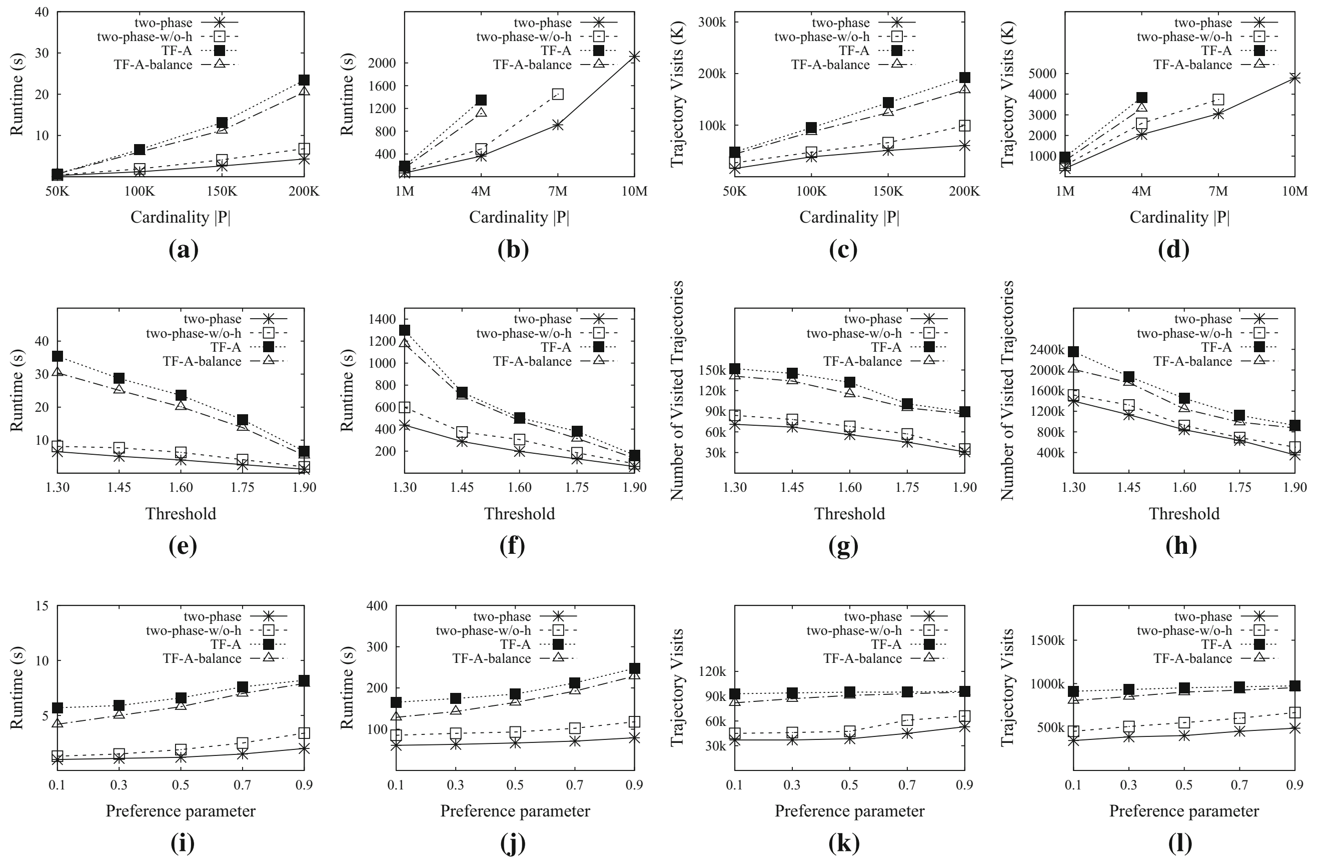


Fig. 9 Performance of the self Tb-TS-Join. **a** BRN, **b** NRN, **c** BRN, **d** NRN, **e** BRN, **f** NRN, **g** BRN, **h** NRN, **i** BRN, **j** NRN, **k** BRN, **l** NRN

Table 5 Pruning effectiveness for non-self *k*-TS-Join

	k-TF	*k*-two-phase	*k*-two-phase-w/o-h
Candidate ratio (BRN)	0.38	0.18	0.25
Pruning ratio (BRN)	0.62	0.82	0.75
Candidate ratio (NRN)	0.31	0.16	0.20
Pruning ratio (NRN)	0.69	0.84	0.80

Table 6 Pruning effectiveness for self *k*-TS-Join

	k-TF	*k*-two-phase	*k*-two-phase-w/o-h
Candidate ratio (BRN)	0.22	0.14	0.17
Pruning ratio (BRN)	0.78	0.86	0.83
Candidate ratio (NRN)	0.17	0.09	0.11
Pruning ratio (NRN)	0.83	0.91	0.89

($|P| = 10$ M and $|Q| = 0.5$ M) in 971 s using 120 threads (see Fig. 11b).

6.3.4 Effect of disk-based storage

Figure 12 shows the performance of the disk-based algorithms. Their performance patterns are similar to those of the memory-based algorithms (Fig. 11). The disk-based algorithms need more runtime because of disk I/O, but the query can still be processed in reasonable time (e.g., processing 10 M × 0.5 M trajectories in around 1184 s using 120 threads, see Fig. 12b).

6.3.5 Effect of thread count *m*

We study the effect of thread count *m* on the efficiency of the algorithms using large trajectory data sets in NRN ($|P| = 10$ M and $|Q| = 2$ M for the non-self *k*-TS-Join and

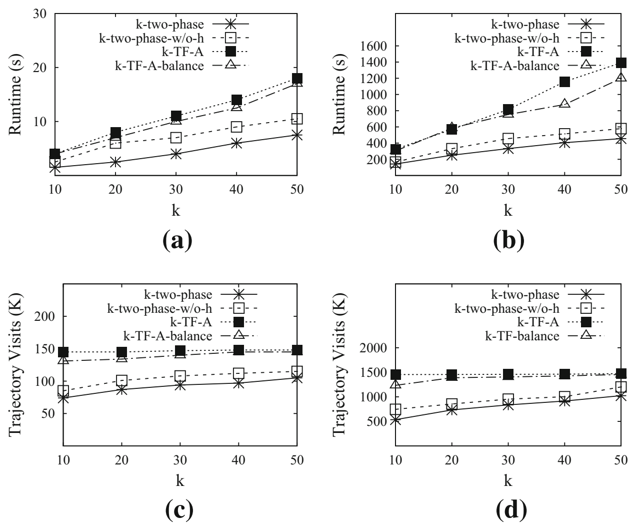


Fig. 10 Effect of k. **a** BRN, **b** NRN, **c** BRN, **d** NRN

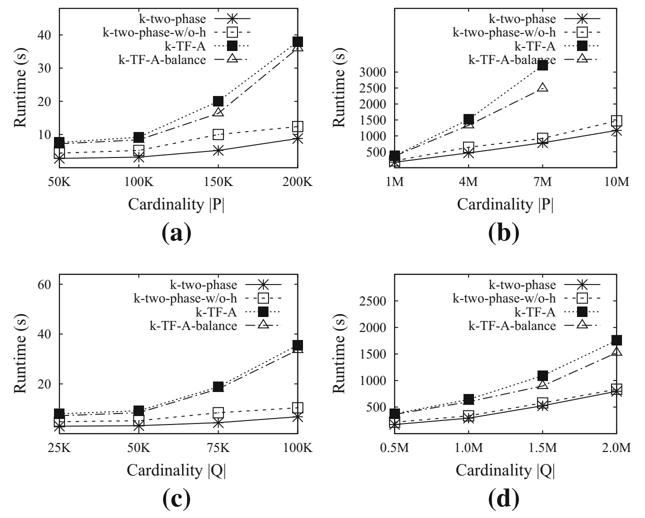


Fig. 12 Effect of disk-based storage. **a** BRN, **b** NRN, **c** BRN, **d** NRN

$|P| = 10$ M for the self k-TS-Join). The results are shown in Fig. 13 (note the logarithmic y-axis in Fig. 13b). We see that the k-two-phase algorithm outperforms k-TF-Matching by almost an order of magnitude in term of runtime. For the non-self join, the two-phase algorithm is able to process 10 M × 2 M trajectories with 144 threads in 820 s, while for the self join, the k-two-phase algorithm is able to process 10 M × 10 M trajectories with 144 threads in 1750 s.

In Fig. 13, we increase the thread count m from 72 to 144 (2 times). This improves the runtime of the k-two-phase algorithm by a factor of 1.62–1.66, while the runtime of k-TF-Matching (k-TF-A and k-TF-A-balance) is improved by a factor of around 1.2. The main reason for the smaller

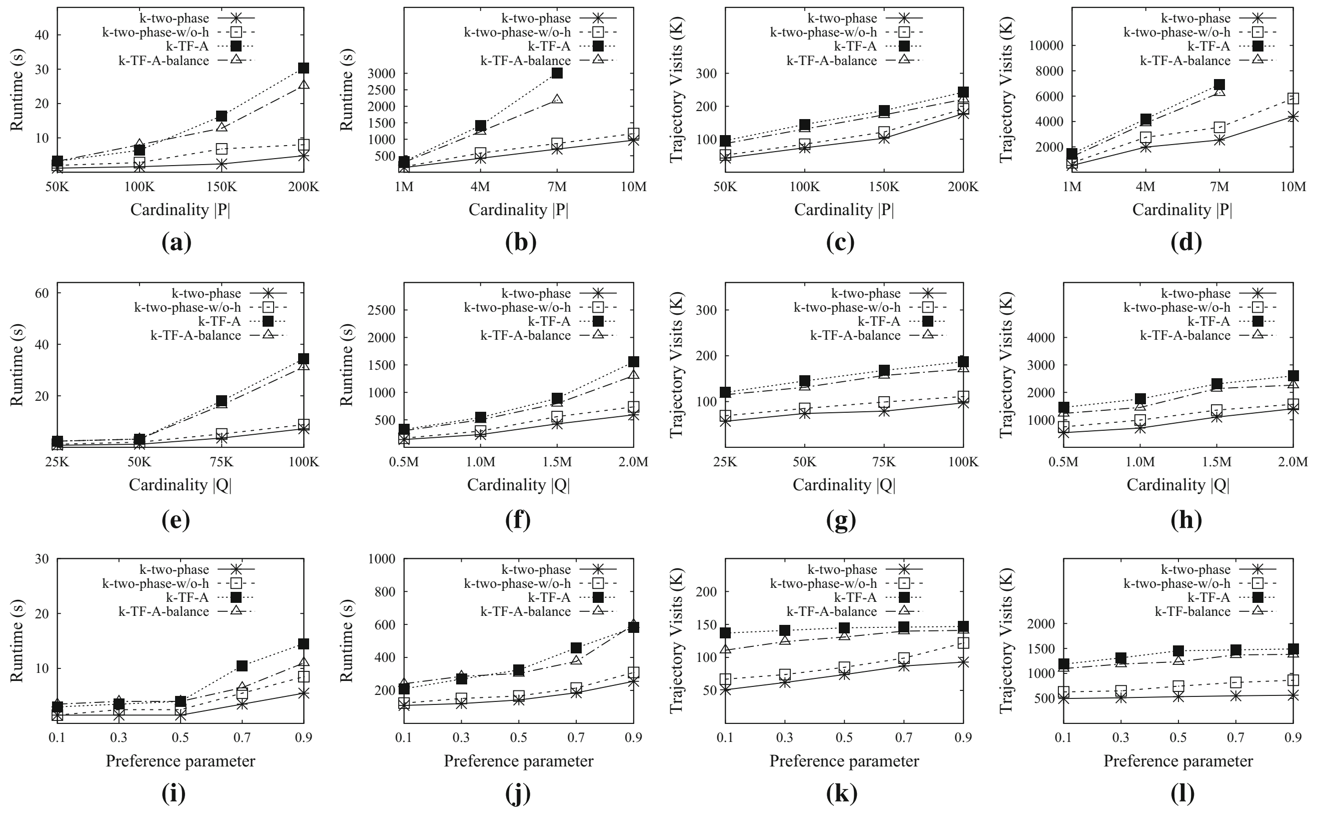


Fig. 11 Effects of $|P|$, $|Q|$, and λ. **a** BRN, **b** NRN, **c** BRN, **d** NRN, **e** BRN, **f** NRN, **g** BRN, **h** NRN, **i** BRN, **j** NRN, **k** BRN, **l** NRN

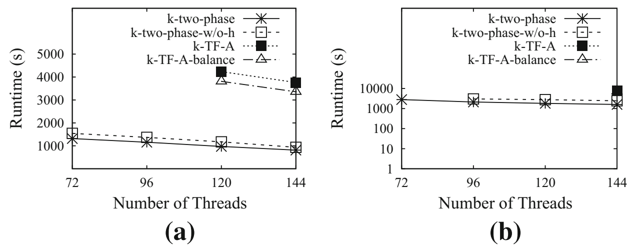


Fig. 13 Effect of thread count m. **a** Non-self join, **b** self join

improvement is that more threads (more leaf nodes) leads to a higher merging cost (cf. Sects. 3.1.5 and 3.2.3).

6.3.6 Performance of the self *k*-TS-Join

Figure 14 shows the runtime and number of trajectory visits for the self *k*-TS-Join when varying trajectory cardinality, *k*, and preference parameter α. The trends of the figures are similar to those for the non-self *k*-TS-Join (cf. Figs. 10– 12). The *k*-two-phase algorithm outperforms *k*-TF-Matching (*k*-TF-A and *k*-TF-A-balance) by almost an order of magnitude in terms of both runtime and trajectory visits, and the heuristic search strategy improves the efficiency by almost a factor of 40%.

6.4 TS-Join performance with sequence similarity

We conducted experiments to study the performance of processing the sequential TB-TS-Join and *k*-TS-Join (when using the sequence similarity measure presented in Sect. 5). The experimental results are shown in Figs. 15 and 16. Compared to the original TB-TS-Join and *k*-TS-Join, the sequential TB-TS-Join and *k*-TS-Join needs more computational efforts to compute the upper and lower bounds and the priority labels, which is due to the more complex distance measures. Therefore, more time and trajectory accesses are incurred. However, the trends observed in Figs. 15 and 16 are still similar to those observed for the original TB-TS-Join and *k*-TS-Join in Figs. 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 and 14. In Figures 15 and 16, the two-phase (*k*-two-phase) algorithm is still able to outperform the TF-Matching (*k*-TF-Matching) algorithm by almost an order of magnitude in term of both runtime and trajectory visits. The sequence similarity two-phase and the sequence similarity *k*-two-phase algorithms are able to process 10 M $\times$ 2 M trajectories with 144 threads in 250 and 840 s.

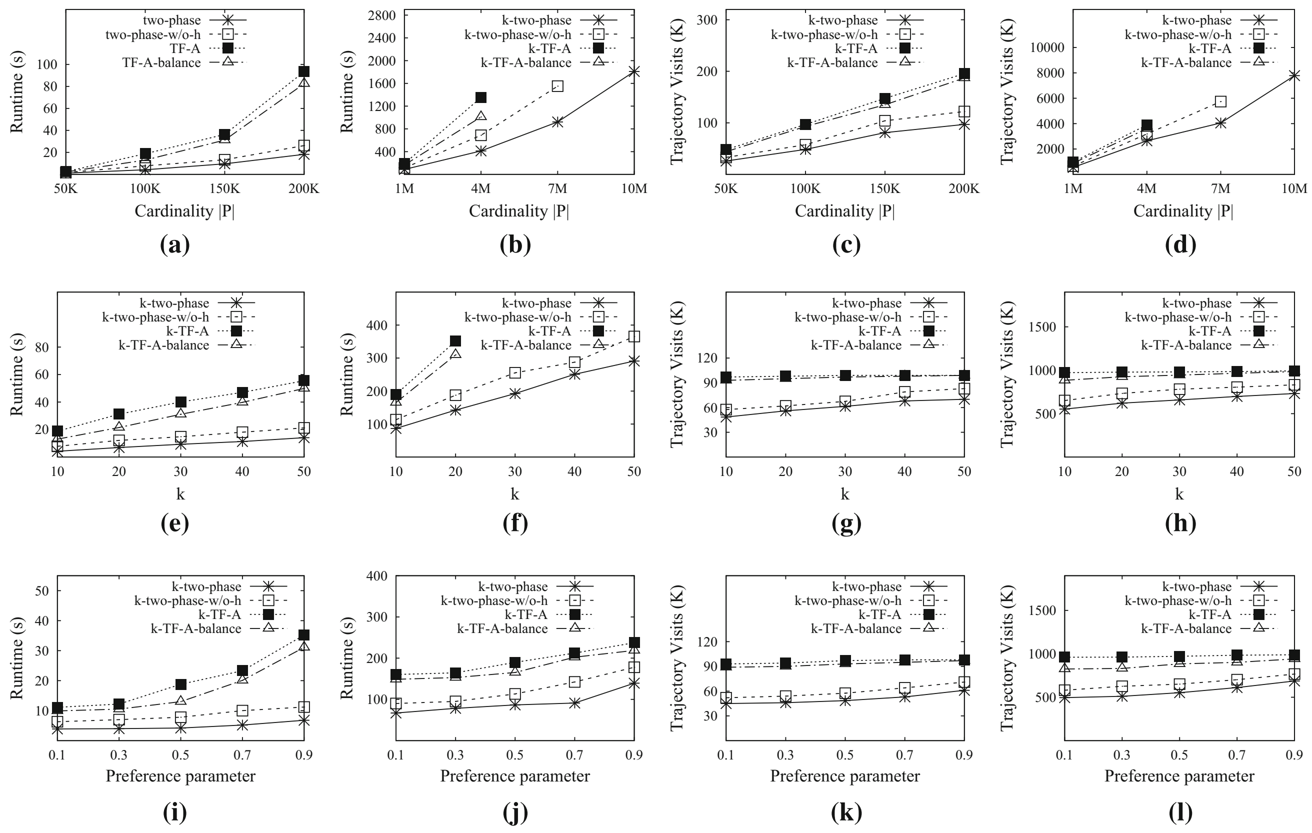


Fig. 14 Performance of the self *k*-TS-Join. **a** BRN, **b** NRN, **c** BRN, **d** NRN, **e** BRN, **f** NRN, **g** BRN, **h** NRN, **i** BRN, **j** NRN, **k** BRN, **l** NRN

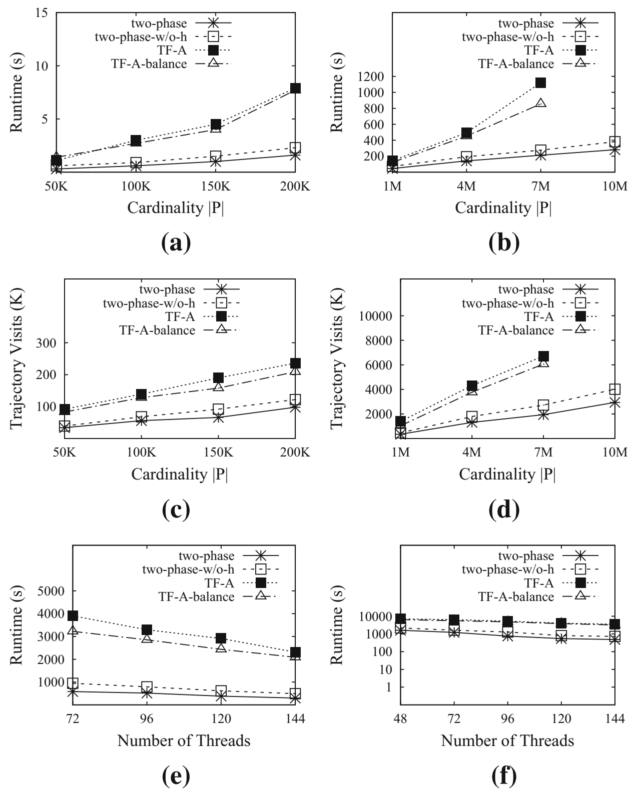


Fig. 15 Performance for TB-TS-Join with a sequence. **a** BRN, **b** NRN, **c** BRN, **d** NRN, **e** non-self join, **f** self join

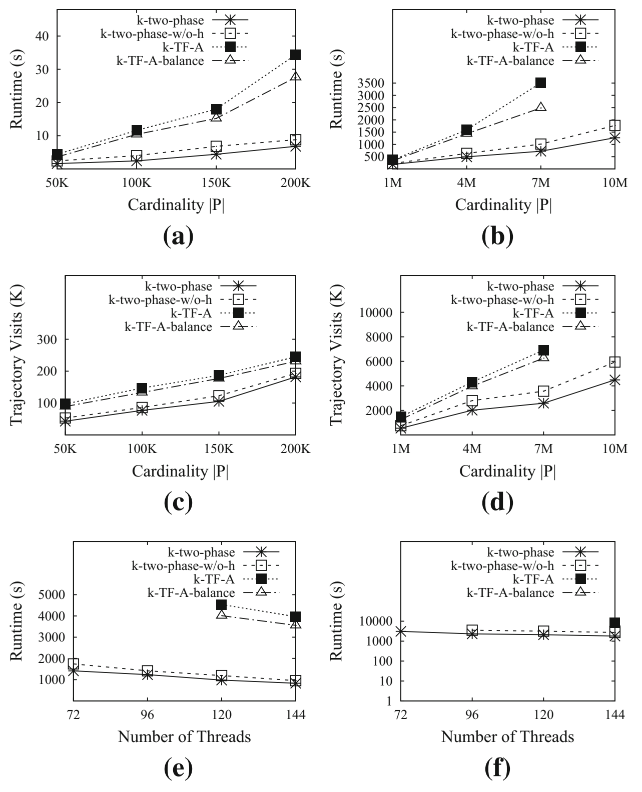


Fig. 16 Performance for *k*-TS-Join with a sequence. **a** BRN, **b** NRN, **c** BRN, **d** NRN, **e** non-self join, **f** self join

6.5 Summary

An empirical study with real data offers insight in the performance of the algorithms (cf. Algorithm 1–5) and provides evidence that the two-phase and *k*-two-phase algorithms (cf. Algorithms 3, 4, and 5) typically are capable of outperforming well-designed baseline algorithms (TF-Matching and *k*-TF-Matching, cf. Algorithms 1 and 2) by an order of magnitude.

7 Related work

7.1 Trajectory similarity search

Trajectory similarity search [7,17,18,26] typically involves a definition step and a query processing step. First, a similarity function is defined to evaluate the spatial and temporal similarities between two trajectories, typically taking into account spatial proximity and curve similarity. Second, an efficient algorithm is developed to retrieve trajectories spatiotemporally close to a query trajectory. Several trajectory similarity functions are proposed for different applications. For example, *BCT* [7] considers trajectory search in Euclidean space, and *UOTS* [17] and *ATSQ* [26] extend these to the spatial and textual domains, while *PTM* [18] extends them into spatial and temporal domains. Next, several similarity functions exist for trajectory or time-series data, including Dynamic Time Warping [23], Longest Common Subsequence [1], and Edit Distance on Real sequence [5]. The definition of *BCT* [7] is most similar to the one we use. Both studies target routing and ridesharing/carpooling. We extend the Euclidean-based *BCT* to spatial networks, and we also offer a symmetrical definition. In contrast, most existing trajectory similarity functions [7,17,18] are asymmetrical; thus, they cannot be used directly in the TS-Joins.

7.2 Trajectory similarity join

Most existing studies on trajectory similarity join (e.g., [2,3,6,10]) use a time interval threshold to constrain the temporal proximity of two trajectories and can be classified into two categories. Studies in the first category (e.g., [3,10]) eliminate trajectory pairs with sample point pairs with time intervals that exceed the threshold. Our study generalizes studies in this category in that we eliminate the time-interval threshold. Studies in the other category (e.g., [2,6]) apply a sliding window to all trajectories. Here, pairs of trajectories must fall into a sliding window to be candidate join results. In contrast, the TS-Joins use spatiotemporal similarity, taking into account both spatial and temporal aggregate distances. Thus, the existing time interval-based solutions are not suitable for the TS-Joins (e.g., the temporal-first matching, cf. Sect. 3). Moreover, in contrast to most existing trajectory join studies (e.g., [2,3,6,10,20]), the TS-Joins are applied in spatial networks because in many practical scenarios, objects (e.g., commuters and vehicles) move in spatial networks (e.g., road networks) rather than in a Euclidean space. Thus, spatial indices (e.g., the R-tree [11]) and corresponding optimizations are not effective in our setting. In addition, existing trajectory similarity join studies (e.g., [2,3,6,10,20]) are not taking steps to exploit the parallelism in modern processors. According to an experimental study [12], most existing similarity join algorithms cannot achieve high performance for really large data sets, making it relevant to pursue parallel algorithms for very large data sets. To address this issue, we introduce parallelism to the temporal-first matching and the two-phase algorithm to process the TS-Joins efficiently on very large trajectory data sets.

8 Conclusion and future work

We present and study novel trajectory similarity joins (TS-Joins) in spatial networks, including a threshold-based TS-Join (Tb-TS-Join) and a top-k TS-Join (k-TS-Join), that target diverse applications such as trajectory duplicate detection, data cleaning, ridesharing/carpooling recommendation, and traffic congestion prediction. To process the TS-Joins efficiently, two-phase and k-two-phase algorithms are developed based on a divide-and-conquer strategy. The algorithms are equipped with upper and lower bounds and a heuristic scheduling strategy that enable effective pruning of the search space. The performance of the TS-Joins is investigated through extensive experiments on very large trajectory data. Two research directions are of particular interest. The first is to study how to select a larger initial lower bound for the k-TS-Join online. The second is to study system-level optimizations in the TS-Joins.

References

1. Agrawal, R., Lin, K., Sawhney, H.S., Shim, K.: Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In: VLDB, pp. 490–501 (1995)
2. Bakalov, P., Hadjieleftheriou, M., Keogh, E.J., Tsotras, V.J.: Efficient trajectory joins using symbolic representations. In: MDM, pp. 86–93 (2005)
3. Bakalov, P., Tsotras, V.J.: Continuous spatiotemporal trajectory joins. In: GSN, pp. 109–128 (2006)
4. Brakatsoulas, S., Pfoser, D., Salas, R., Wenk, C.: On map-matching vehicle tracking data. In: VLDB, pp. 853–864 (2005)
5. Chen, L., Özsu, M.T., Oria, V.: Robust and fast similarity search for moving object trajectories. In: SIGMOD, pp. 491–502 (2005)
6. Chen, Y., Patel, J.M.: Design and evaluation of trajectory join algorithms. In: ACM-GIS, pp. 266–275 (2009)
7. Chen, Z., Shen, H.T., Zhou, X., Zheng, Y., Xie, X.: Searching trajectories by locations: an efficiency study. In: SIGMOD, pp. 255–266 (2010)
8. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications. Springer, Berlin (2008)
9. Dijkstra, E.W.: A note on two problems in connection with graphs. Numer. Math. **1**, 269–271 (1959)
10. Ding, H., Trajcevski, G., Scheuermann, P.: Efficient similarity join of large sets of moving object trajectories. In: TIME, pp. 79–87 (2008)
11. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: SIGMOD, pp. 47–57 (1984)
12. Jiang, Y., Li, G., Feng, J., Li, W.: String similarity joins: an experimental evaluation. PVLDB **7**(8), 625–636 (2014)
13. Luo, W., Tan, H., Chen, L., Ni, L.M.: Finding time period-based most frequent path in big trajectory data. In: SIGMOD, pp. 713–724 (2013)
14. Shang, S., Chen, L., Jensen, C.S., Wen, J., Kalnis, P.: Searching trajectories by regions of interest. IEEE Trans. Knowl. Data Eng. **29**(7), 1549–1562 (2017)
15. Shang, S., Chen, L., Wei, Z., Jensen, C.S., Wen, J., Kalnis, P.: Collective travel planning in spatial networks. IEEE Trans. Knowl. Data Eng. **28**(5), 1132–1146 (2016)
16. Shang, S., Chen, L., Wei, Z., Jensen, C.S., Zheng, K., Kalnis, P.: Trajectory similarity join in spatial networks. PVLDB **10**(11), 1178–1189 (2017)
17. Shang, S., Ding, R., Yuan, B., Xie, K., Zheng, K., Kalnis, P.: User oriented trajectory search for trip recommendation. In: EDBT, pp. 156–167 (2012)
18. Shang, S., Ding, R., Zheng, K., Jensen, C.S., Kalnis, P., Zhou, X.: Personalized trajectory matching in spatial networks. VLDB J. **23**(3), 449–468 (2014)
19. Shang, S., Zheng, K., Jensen, C.S., Yang, B., Kalnis, P., Li, G., Wen, J.: Discovery of path nearby clusters in spatial networks. IEEE Trans. Knowl. Data Eng. **27**(6), 1505–1518 (2015)
20. Ta, N., Li, G., Xie, Y., Li, C., Hao, S., Feng, J.: Signature-based trajectory similarity join. IEEE Trans. Knowl. Data Eng. **29**(4), 870–883 (2017)
21. Tang, L.A., Zheng, Y., Xie, X., Yuan, J., Yu, X., Han, J.: Retrieving k-nearest neighboring trajectories by a set of point locations. In: SSTD, pp. 223–241 (2011)
22. Wenk, C., Salas, R., Pfoser, D.: Addressing the need for map-matching speed: localizing global curve-matching algorithms. In: SSDBM, pp. 379–388 (2006)
23. Yi, B., Jagadish, H.V., Faloutsos, C.: Efficient retrieval of similar time sequences under time warping. In: ICDE, pp. 201–208 (1998)
24. Yuan, J., Zheng, Y., Xie, X., Sun, G.: Driving with knowledge from the physical world. In: SIGKDD, pp. 316–324 (2011)
25. Yuan, J., Zheng, Y., Zhang, C., Xie, W., Xie, X., Sun, G., Huang, Y.: T-drive: driving directions based on taxi trajectories. In: ACM SIGSPATIAL, pp. 99–108 (2010)
26. Zheng, K., Shang, S., Yuan, N.J., Yang, Y.: Towards efficient search for activity trajectories. In: ICDE, pp. 230–241 (2013)
27. Zheng, K., Zheng, Y., Yuan, N.J., Shang, S., Zhou, X.: Online discovery of gathering patterns over trajectories. IEEE Trans. Knowl. Data Eng. **26**(8), 1974–1988 (2014)
28. Zhou, J., Tung, A.K.H.,Wu, W., Ng, W.S.: A "semi-lazy" approach to probabilistic path prediction. In: SIGKDD, pp. 748–756 (2013)