Open access • Journal Article • DOI:10.14778/3137628.3137630

Trajectory similarity join in spatial networks — Source link

Shuo Shang, Lisi Chen, Zhewei Wei, Christian S. Jensen ...+2 more authors

Institutions: King Abdullah University of Science and Technology, Renmin University of China, Aalborg University, Soochow University (Suzhou)

Published on: 01 Aug 2017 - Very Large Data Bases

Topics: Similarity (network science), Matching (graph theory), Pruning (decision trees) and Trajectory

Trajectory Similarity Join in Spatial Networks

Shuo Shang
KAUST
jedi.shang@gmail.com

Lisi Chen
HKBU
chenlisi@comp.hkbu.edu.hk

Zhewei Wei
Renmin University of China
zhewei@ruc.edu.cn

Christian S. Jensen
Aalborg University
csj@cs.aau.dk

Kai Zheng
Soochow University
zhengkai@suda.edu.cn

Panos Kalnis
KAUST
panos.kalnis@kaust.edu.sa

ABSTRACT

The matching of similar pairs of objects, called similarity join, is fundamental functionality in data management. We consider the case of trajectory similarity join (TS-Join), where the objects are trajectories of vehicles moving in road networks. Thus, given two sets of trajectories and a threshold θ, the TS-Join returns all pairs of trajectories from the two sets with similarity above θ. This join targets applications such as trajectory near-duplicate detection, data cleaning, ridesharing recommendation, and traffic congestion prediction.

With these applications in mind, we provide a purposeful definition of similarity. To enable efficient TS-Join processing on large sets of trajectories, we develop search space pruning techniques and take into account the parallel processing capabilities of modern processors. Specifically, we present a two-phase divide-and-conquer algorithm. For each trajectory, the algorithm first finds similar trajectories. Then it merges the results to achieve a final result. The algorithm exploits an upper bound on the spatiotemporal similarity and a heuristic scheduling strategy for search space pruning. The algorithm's per-trajectory searches are independent of each other and can be performed in parallel, and the merging has constant cost. An empirical study with real data offers insight in the performance of the algorithm and demonstrates that is capable of outperforming a well-designed baseline algorithm by an order of magnitude.

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], 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 this setting, trajectory similarity join (TS-Join) is fundamental functionality: given sets P and Q of trajectories and a similarity threshold θ, the TS-Join returns all pairs of trajectories from P and Q with a similarity that exceeds θ.

The TS-Join may bring significant benefits to a range of applications, including trajectory near-duplicate detection, data cleaning [2, 19], ridesharing recommendation [16, 17], friend recommendation [17], frequent trajectory based routing [13, 19], and traffic congestion prediction. 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 that have similar trajectories, and social network in services may identify users with similar living trajectories and use this in friend recommendations. We may also use the TS-Join to find frequently traveled trajectories (e.g., trajectory τ joins with m other trajectories, and the travel frequency of τ is $m+1$), 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 similarity in a continuous manner. A linear combination method (e.g., [17, 18]) is adopted 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 classified into two 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 in a continuous manner, 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

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

Proceedings of the VLDB Endowment, Vol. 10, No. 11

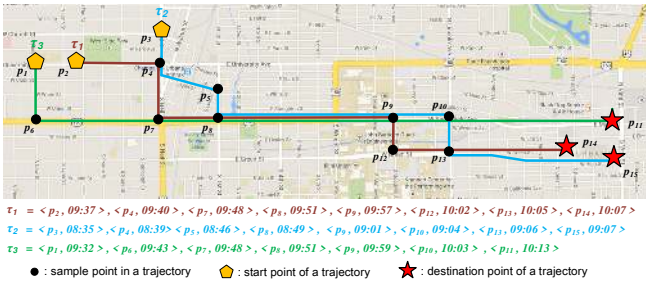


Figure 1: TS-Join Example

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 of the TS-Join is shown in Figure 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-Join returns 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 [2,3,6,10,19], the TS-Join is applied in spatial networks because in many practical scenarios, objects (e.g., commuters and vehicles) move in spatial networks rather than in Euclidean space. In spatial networks, only the network distance can reflect the real distance between two objects, and 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 (spatial domain) according to some map-matching algorithm (e.g., [4, 20]), and we assume that the timestamps of all trajectory sample points are mapped to a time axis with a 24-hour range (temporal domain) [17].

Existing methods cannot process the TS-Join due to three reasons. (i) Different query spaces (Euclidean vs. network): these joins (e.g., [2, 3, 6, 10, 19]) are conducted in Euclidean space rather than in a spatial network. Thus, existing spatial indices (e.g., the R-tree [11]) and accompanying techniques are ineffective here. (ii) Different temporal matching schemes (time interval vs. continuous evaluation): most of the existing trajectory similarity joins are time-interval based (e.g., [2,3,6,10]), and their solutions are inapplicable to continuous temporal matching. They compute different results than the TS-Join (refer to Figure 1). (iii) Parallel processing: an experimental study [12] shows that existing centralized similarity join techniques (that do not take parallel processing into account) are far from efficient in processing very large data sets. Existing centralized trajectory similarity joins (e.g., [2, 3, 6, 10, 19]) can process at most 500 K trajectories (based on their experiments), while the TS-Join is able to process 10 M trajectories with a reasonable runtime (e.g., processing 10 M $\times$ 2 M trajectories for non-self join in 255 seconds and processing 10 M trajectories for self join in 540 seconds). A comparison between the TS-Join and existing studies is shown in Table I.

Table I: 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
[19]	Euclidean	None (spatial join only)	No	500 K
TS-Join	Network	Continuous matching	Yes	10 M

We propose a relatively straightforward approach to the TS-Join called temporal-first matching (TF-Matching). Initially, we apply a hierarchical grid index in the temporal domain. Then we refine the 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. Upper and lower bounds are defined to prune the search space in the spatial and temporal domains. The computations at each index level occur in parallel. TF-Matching has four technique contributions: pruning in leaf nodes, pruning among different nodes, merging, and parallel processing. The only similarity between TF-Matching 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 TF-Matching because of the different query spaces (Euclidean vs. network) and the different temporal matching schemes (time interval vs. continuous evaluation). TF-Matching is considered a contribution of the paper.

TF-Matching has three main limitations. First, it is driven by the temporal domain and thus has weak spatial pruning power. As a result, the algorithm needs to consider a large number of pairs. 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-Join efficiently, we propose a two-phase algorithm based on a divide-and-conquer strategy. In the trajectory-search phase, for each trajectory τ, the algorithm explores 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. An upper bound on the spatiotemporal similarity is defined to prune the search space, and a heuristic scheduling strategy is proposed to schedule multiple so-called query sources in order to improve efficiency. The trajectory-search processes are independent of each other, enabling parallel processing, and the merging cost is constant (uncorrelated to number of threads used for parallel processing). The network distances for similarity computation can be derived directly during the trajectory-search process. A time complexity analysis indicates that the two-phase algorithm is considerably better than the temporal-first matching algorithm.

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

- We propose a novel network-based trajectory similarity join, called TS-Join, that takes into account both spatial and temporal similarity in a continuous manner, thus targeting applications such as trajectory near-duplicate detection, ridesharing recommendation, route planning, and traffic congestion prediction.

- The TS-Join uses new metrics to evaluate trajectory similarity in the spatial and temporal domains.
- We develop a temporal-first baseline algorithm that enables parallel TS-Join processing.
- We develop a two-phase algorithm with effective pruning and scheduling techniques that enables parallel TS-Join processing.
- We conduct extensive experiments on large trajectory sets to study the performance of the developed algorithms.

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. Temporal-first matching is covered in Section 3, while the two-phase algorithm is covered in Section 4. The developed algorithms are extended to support non-self joins in Section 5, which is followed by a presentation of experimental results in Section 6. Related work is covered in Section 7, and conclusions and future directions are presented in Section 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, 20]) 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 $\langle v_1, v_2, ..., v_n \rangle$, 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 hours, 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, 17, 18].

2.2 Trajectory Similarity Functions

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

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

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

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

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

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

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 $\mathrm{Sim_S}(\tau_1, \tau_2) = \mathrm{Sim_S}(\tau_2, \tau_1)$ and $\mathrm{Sim_T}(\tau_1, \tau_2) = \mathrm{Sim_T}(\tau_2, \tau_1)$. In contrast, most of existing trajectory similarity measures (e.g., [7, 16, 17]) 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 [16–18] to combine spatial and temporal similarities (Equations 3 and 4), and the spatiotemporal similarity is defined as follows.

$$\mathrm{Sim_{ST}}(\tau_1, \tau_2) = \lambda \cdot \mathrm{Sim_S}(\tau_1, \tau_2) + (1 - \lambda) \cdot \mathrm{Sim_T}(\tau_1, \tau_2) \quad (5)$$

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

Given sets P and Q of trajectories and a threshold θ, the trajectory similarity join (TS-Join) finds a set A of all trajectory pairs from the two sets whose spatiotemporal similarity exceeds θ, i.e., $\forall (\tau_i, \tau_j) \in (P \times Q) \setminus A \, (\mathrm{Sim_{ST}}(\tau_i, \tau_j) < \theta)$.

We initially consider the self-join scenario (i.e., $P = Q$), and then cover the case $P \neq Q$ in Section 5.

3. BASELINE ALGORITHM

3.1 Basic Idea

Temporal-first matching (TF-Matching) is a straightforward baseline approach to computing the 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 (Sections 3.2 and 3.3). By merging the results from the leaf nodes toward the root, the join result is obtained when the root is reached (Section 3.4). 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.

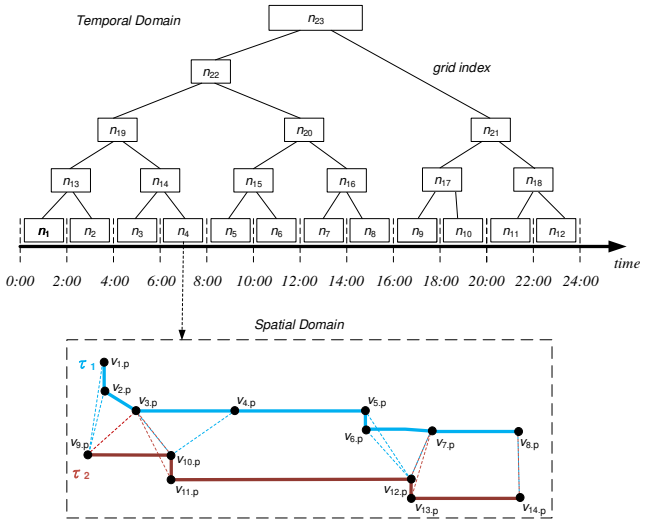


Figure 2: An example of TF-Matching

3.2 Grid Index

The grid index structure [8] is established as follows. First, we partition the temporal domain into m 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 $k = m$). 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 $\lceil log(m) \rceil + 1$. An example is shown in Figure 2, where $n_1, n_2, ..., n_{23}$ are nodes and n_{23} is the root. To find a value for m that yields high performance, we conducted extensive experiments when establishing the grid index.

The temporal range $\text{range}(\tau)$ of a trajectory $\tau = \langle v_1, v_2, ..., v_i \rangle$ is defined by the timestamps of its start and end sample points, i.e., $\text{range}(\tau) = [v_1.t, v_i.t]$. 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., $\text{range}(\tau) \subseteq \text{range}(n)$ and $\text{range}(\tau)$ 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 Figure 2, where $\text{range}(\tau_1)$ = [6:15, 7:30] and $\text{range}(\tau_2)$ = [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 ($\text{range}(n_4)$ = [6:00, 8:00]) because $\text{range}(\tau_1) \subseteq \text{range}(n_4)$, $\text{range}(\tau_2) \subseteq \text{range}(n_4)$, and n_4 is a leaf node. Given a trajectory τ_3 and $\text{range}(\tau_3)$ = [9:45, 10:30], τ_3 is stored in n_{15} ($\text{range}(n_{15})$ = [8:00, 12:00]) because $\text{range}(\tau_3) \subseteq \text{range}(\text{n}_{15})$, and $\text{range}(\tau_3) \nsubseteq \text{range}(\text{n}_5)$, $\text{range}(\tau_3) \nsubseteq \text{range}(n_6)$ (n_5 and n_6 are child nodes of n_{15}).

3.3 Upper and Lower Bounds

In the example in Figure 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 of the temporal similarity $\text{Sim}_\text{T}(\tau_1, \tau_2)$ (Equation 4) as follows.

$$\sum_{v_i \in \tau_1} e^{-d(v_i.t, \tau_2)} \leq |\tau_1| \quad \text{and} \quad \sum_{v_j \in \tau_2} e^{-d(v_j.t, \tau_1)} \leq |\tau_2|$$

$$\Rightarrow \text{Sim}_\text{T}(\tau_1, \tau_2).ub = 2 \geq \text{Sim}_\text{T}(\tau_1, \tau_2) \tag{6}$$

Here, $|\tau|$ is the number of sample points in τ.

By substituting Equation 6 into Equation 5, we have that

$$\lambda \cdot \text{Sim}_\text{S}(\tau_1, \tau_2) + (1 - \lambda) \cdot \text{Sim}_\text{T}(\tau_1, \tau_2) \geq \theta$$

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

where LB_S is a global lower bound of the spatial similarity of all "qualified" trajectory pairs in the same leaf node. 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 Equation 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 Equation 8 into Equation 3, we estimate the upper bound $\text{Sim}_\text{S}(\tau_1, \tau_2).ub$ of the spatial similarity between τ_1 and τ_2 as follows.

$$\sum_{v_i \in \tau_1} e^{-d(v_i.p, \tau_2)} \leq |\tau_1| \cdot e^{-\min_{v_i \in \tau_2} \{d(v_i.p, \tau_1)\}} \Rightarrow$$

$$\text{Sim}_\text{S}(\tau_1, \tau_2).ub = e^{-\min_{v_i \in \tau_2} \{d(v_i.p, \tau_1)\}} + \frac{\sum_{v_j \in \tau_2} e^{-d(v_j.p, \tau_1)}}{|\tau_2|} \tag{9}$$

According to Equation 9, we only need to compute half of the exact spatial similarity to get the upper bound.

For any trajectory pair (τ_1, τ_2) in the same leaf node, if its spatial-similarity upper bound $\text{Sim}_\text{S}(\tau_1, \tau_2).ub$ (Equation 9) is less than the global lower bound LB_S of the spatial similarity (Equation 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., $\text{Sim}_\text{ST}(\tau_1, \tau_2) \geq \theta$.

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

3.4 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 Figure 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) $\text{range}(\tau) \subseteq \text{range}(n_a) \wedge \text{range}(\tau') \subseteq \text{range}(n_c)$

(2) $\text{range}(\tau) \subseteq \text{range}(n_b) \wedge \text{range}(\tau') \subseteq \text{range}(n_c)$

(3) $\text{range}(\tau) \subseteq \text{range}(n_a) \wedge \text{range}(\tau') \subseteq \text{range}(n_b)$

In the first two cases, we use the same lower and upper bounds (Equations 7 and 9) and pruning techniques as we use for trajectory pairs in the same node (cf. Section 3.3). The qualified trajectory pairs are stored in n_c.

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

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

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 range(n). For example, range(n_4) = [6:00, 8:00], and $n_4.t.lb$ = 6:00 and $n_4.t.lb$ = 8:00.

We define the upper bound of temporal similarity $\mathrm{Sim_T}(\tau, \tau')$ in the following manner. As $\sum_{v_i \in \tau} e^{-d(v_i.t, \tau')} \leq |\tau| \cdot e^{-d_T(\tau, n_b)}$ and $\sum_{v_j \in \tau'} e^{-d(v_j.t, \tau)} \leq |\tau'| \cdot e^{-d_T(\tau', n_a)}$, we have that

$$\mathrm{Sim'_T}(\tau, \tau').ub = e^{-d_T(\tau, n_b)} + e^{-d_T(\tau', n_a)}. \quad (11)$$

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

$$\mathrm{Sim_S}(\tau, \tau') \geq \frac{\theta - (1-\lambda)(\mathrm{Sim'_T}(\tau, \tau').ub)}{\lambda} = LB'_S \quad (12)$$

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 $\mathrm{Sim_S}(\tau, \tau').ub$ (Equation 9) is less than the global lower bound LB'_S of the spatial similarity (Equation 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 Equations 11 and 7, the upper and lower bounds of temporal similarity are computed respectively as follows.

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

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

$$\mathrm{Sim'_T}(\tau, \tau').ub < LB_T \Leftrightarrow d_T(n_d, n_e) > ln(\frac{2-2\lambda}{\theta - 2\lambda})$$

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

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.

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 (Equation 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 (Equation 9), and if its upper bound is less than LB_S, this pair is pruned (lines 4–7). Otherwise, we compute

Algorithm 1: TF-Matching

Data: a grid indexing tree T_g, a trajectory set P, a threshold θ
Result: $\{(\tau, \tau') | \mathrm{Sim_{ST}}(\tau, \tau') \geq \theta, \forall \tau, \tau' \in P\}$

```
1  H ← ∅;
2  for each leaf node n in T_g do
3      compute LB_S;
4      for each trajectory pair (τ, τ'), range(τ) ⊆ range(n) and
       range(τ') ⊆ range(n) do
5          compute Sim_S(τ, τ').ub;
6          if Sim_S(τ, τ').ub < LB_S then
7              prune (τ, τ');
8          compute Sim_ST(τ, τ');
9          if Sim_ST(τ, τ') ≥ θ then
10             store (τ, τ') in n;
11     H.add(n);
12 while H ≠ ∅ do
13     if n, n' ∈ H, n.parent = n'.parent, and n.child ∉ H,
       n'.child ∉ H 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 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 towards 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 Figure 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).

3.5 Complexity Analysis

Let $|P|$ denote the cardinality of trajectory set P, and $|\tau_{avg}|$ 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 two vertices. TF-Matching follows the filter-and-refine paradigm, and the time complexity of the filter phase is $O((|V| \log |V| + |E|)|\tau_{avg}|^2 |P|^2)$.

The time complexity to verify the candidates by computing their exact spatiotemporal similarities is $O((|V| \log |V| + |E|)|\tau_{avg}|^2 |C|)$, where $|C|$ is the cardinality of the candidate set and $C \subseteq P \times P$. The total time complexity is $O((|V| \log |V| + |E|)|\tau_{avg}|^2 |P|^2) + O((|V| \log |V| + |E|)|\tau_{avg}|^2 |C|)$

$$= O((|V| \log |V| + |E|)|\tau_{avg}|^2 |P|^2),$$

which does not depend on the candidate set size.

The computations for nodes at the same tree level are processed in parallel. Initially, we process the leaf nodes and then we process $\lceil \log m \rceil$ levels for merging, where m is the number of leaf nodes. Intuitively, if we have 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 leads to more tree levels, which will increase the merging cost.

4. TWO-PHASE ALGORITHM

4.1 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) leads to a higher merging cost, which counts against parallel processing. Third, it may need additional computation to acquire network distances to compute spatial similarities (Equations 1 and 3), again decreasing performance.

To process the TS-Join more efficiently, we develop a two-phase algorithm based on a divide-and-conquer strategy (see Figure 3(a)). (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. The search process of different trajectories are 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 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.

4.2 Expansion Search

Consider the example in Figure 3(b), 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 $\tau_1 = \langle v_1, v_2, ..., v_5 \rangle$, 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 $\{v_8, v_9, ..., v_{12}\} \in \tau_3$, 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. 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. 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 Equation 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."

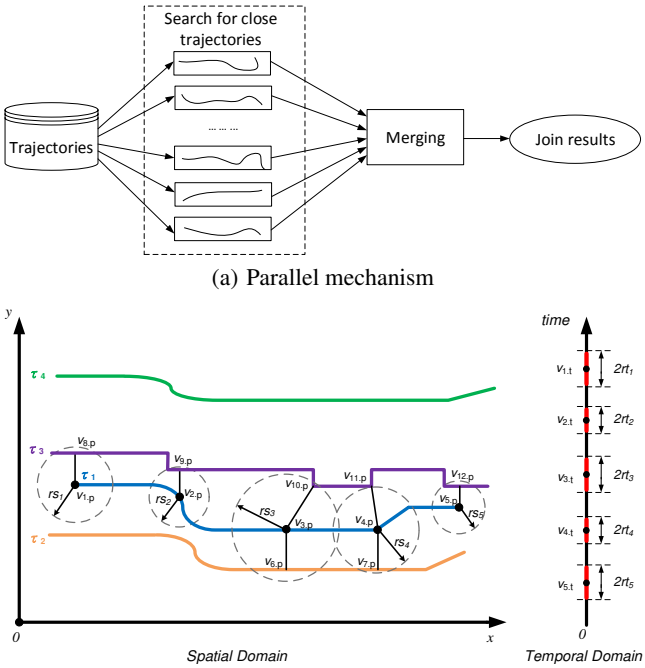


Figure 3: An example of the two-phase algorithm

4.2.1 *Upper Bound Computation*

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

$$d(v_i.p, \tau_2) \geq d(v_i.p, \tau_2).lb = \begin{cases} sd(v_i.p, v_i'.p) & \text{if Case 1} \\ rs_i & \text{if Case 2} \end{cases} \tag{13}$$

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 Equation 13 into Equation 8, for any sample point $v_i'.p \in \tau_2$, we have that

$$d(v_i'.p, \tau_2) \geq \min_{v_i \in \tau_1} \{d(v_i.p, \tau_2).lb\}. \tag{14}$$

Then we merge Equations 13 and 14 into Equation 3, and the spatial similarity upper bound $\mathrm{Sim_S}(\tau_1, \tau_2).ub$ is derived.

$$\sum_{v_i \in \tau_1} e^{-d(v_i.p, \tau_2)} \leq \sum_{v_i \in \tau_1} e^{-d(v_i.p, \tau_2).lb}$$

$$\sum_{v_i' \in \tau_2} e^{-d(v_i'.p, \tau_1)} \leq |\tau_2| \cdot \min_{v_i \in \tau_1} \{d(v_i.p, \tau_2).lb\}$$

$$\Rightarrow \mathrm{Sim_S}(\tau_1, \tau_2).ub = \frac{\sum_{v_i \in \tau_1} e^{-d(v_i.p, \tau_2).lb}}{|\tau_1|} + e^{-\min_{v_i \in \tau_1} \{d(v_i.p, \tau_2).lb\}} \tag{15}$$

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 of the distance between $v_i.t$ and τ_2 is defined as:

$$d(v_i.t, \tau_2) \geq d(v_i.t, \tau_2).lb = \begin{cases} |v_i.t - v_i'.t| & \text{if Case 3} \\ rt_i & \text{if Case 4} \end{cases} \quad (16)$$

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 (Equation 8) to apply to the temporal domain. Specifically, by substituting Equation 16 into Equation 8, for any sample point $v_i'.t \in \tau_2$, we have that

$$d(v_i'.t, \tau_2) \geq \min_{v_i \in \tau_1} \{d(v_i.t, \tau_2).lb\}. \quad (17)$$

Then, we merge Equations 16 and 17 into Equation 4, and the temporal similarity upper bound $\mathrm{Sim_T}(\tau_1, \tau_2).ub$ is derived.

$$\sum_{v_i \in \tau_1} e^{-d(v_i.t, \tau_2)} \leq \sum_{v_i \in \tau_1} e^{-d(v_i.t, \tau_2).lb}$$

$$\sum_{v_i' \in \tau_2} e^{-d(v_i'.t, \tau_1)} \leq |\tau_2| \cdot \min_{v_i \in \tau_1} \{d(v_i.t, \tau_2).lb\}$$

$$\Rightarrow \mathrm{Sim_T}(\tau_1, \tau_2).ub = \frac{\sum_{v_i \in \tau_1} e^{-d(v_i.t, \tau_2).lb}}{|\tau_1|} + e^{-\min_{v_i \in \tau_1}\{d(v_i.t, \tau_2).lb\}} \quad (18)$$

Next, we combine the spatial and temporal similarity upper bounds (Equations 15 and 18). Thus, if a trajectory τ_2 is not both spatially and temporally fully scanned, we compute the upper bound of the spatiotemporal similarity $\mathrm{Sim_T}(\tau_1, \tau_2).ub$ as follows.

$$\mathrm{Sim_{ST}}(\tau_1, \tau_2).ub = \lambda \cdot \mathrm{Sim_S}(\tau_1, \tau_2).ub + (1-\lambda) \cdot \mathrm{Sim_T}(\tau_1, \tau_2).ub \quad (19)$$

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

$$UB = \max_{\tau_2 \in P_{ps}} \{\mathrm{Sim_{ST}}(\tau_1, \tau_2).ub\}, \quad (20)$$

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 Equations 13 and 16, 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 Equations 15, 18, and 19, we have $\mathrm{Sim_{ST}}(\tau_1, \tau_2).ub \geq \mathrm{Sim_{ST}}(\tau_1, \tau_4).ub$. Therefore, $\mathrm{Sim_{ST}}(\tau_1, \tau_4).ub$ cannot be the global upper bound UB, and it is not necessary to maintain the spatiotemporal similarity upper bound of τ_4.

4.2.2 Scheduling Strategy

We propose a heuristic strategy to scheduling 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 $\tau = \langle v_1, v_2, ..., v_m \rangle$ is the query trajectory. We give each query source $q \in \{v_1.p, v_2.p, ..., v_m.p\} \cup \{v_1.t, v_2.t, ..., v_m.t\}$ a priority label $q.\mathrm{label}$ and maintain a heap H of descending order on the value of $q.\mathrm{label}$ 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.

$$q.\mathrm{label} = \sum_{\tau' \in P_{ps} \setminus q.s} \{\mathrm{Sim_{ST}}(\tau, \tau').ub\} \quad (21)$$

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 Figure 3(b), τ_1 is a query trajectory and $v_1.p, v_2.p, ..., v_5.p$ are query sources in the spatial domain. We have that $v_1.p.s = \{\tau_2\}$, $v_2.p.s = \{\tau_2\}$, $v_3.p.s = \emptyset$, $v_4.p.s = \emptyset$, and $v_5.p.s = \emptyset$. Trajectory τ_2 is spatially partly scanned, τ_3 is spatially fully 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 [16, 17]. 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 Figure 3(b), $P_{ps} \setminus v_1.p.s = \{\tau_2, \tau_4\}$; thus, the margin of $v_1.p$ is 2. Moreover, a trajectory with a higher spatiotemporal-similarity upper bound (Equation 19) is more likely to be the solution. So, $\forall \tau \in P_{ps} \setminus q.s$, the value of $\mathrm{Sim_{ST}}(\tau_1, \tau).ub$ is proportional to the priority of query source q.

4.3 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 Equations 15, 18, and 19).

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.

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

Notice that the value of $V(\tau_1, \tau)$ can be derived from Equations 15 and 18 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:

$$\mathrm{Sim_{ST}}(\tau_1, \tau_2) = V(\tau_1, \tau_2) + V(\tau_2, \tau_1) \quad (22)$$

$$= \frac{\sum_{v_i \in \tau_1} e^{-d(v_i.p, \tau_2)}}{|\tau_1|} + \frac{\sum_{v_i \in \tau_1} e^{-d(v_i.t, \tau_2)}}{|\tau_1|} + \frac{\sum_{v_j \in \tau_2} e^{-d(v_j.p, \tau_1)}}{|\tau_2|} + \frac{\sum_{v_j \in \tau_2} e^{-d(v_j.t, \tau_1)}}{|\tau_2|}$$

Then we compare $\mathrm{Sim_{ST}}(\tau_1, \tau_2)$ to threshold θ. If $\mathrm{Sim_{ST}}(\tau_1, \tau_2) \geq \theta$, (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 2. 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 3.

Algorithm 2: Trajectory Search Algorithm

```
   Data: a trajectory τ and a threshold θ
   Result: candidate set C(τ)
1  H ← {v1.p, v2.p, ..., v|τ|.p} ∪ {v1.t, v2.t, ..., v|τ|.t};
2  ∀q ∈ H (q.label ← 0), UB ← 0;
3  q ← H.top;
4  while true do
5      search(q);
6      for each newly scanned trajectory τ' do
7          q.s.add(τ');
8          if τ' ∉ Pps then
9              Pps.add(τ');
10         update Sim_ST(τ, τ').ub;
11         if τ' is not fully scanned then
12             if Sim_ST(τ, τ').ub > UB then
13                 UB ← Sim_ST(τ, τ').ub;
14         if τ' is fully scanned then
15             Pps.remove(τ);
16             update UB;
17             if Sim_ST(τ, τ').ub ≥ θ then
18                 C(τ).add(τ')
19         if UB < θ then
20             return C(τ);
21     if q ≠ H.top then
22         q ← H.top;
```

In Algorithm 2, 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 $\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau').ub$ (refer to Equation 19). If τ' is not fully scanned in the two domains, then if $\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau').ub > UB$, we update the value of UB to that of $\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau').ub$ (lines 10–13). If τ' is fully scanned, we remove it from P_{ps}. If $\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau').ub$ was used as UB before, we also update the value of UB. If $\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau').ub \geq \theta$, 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 3 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 $\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau')$ (refer to Equation 22), and then we remove τ from $C(\tau')$. If $\mathrm{Sim}_{\mathrm{ST}}(\tau, \tau') \geq \theta$, we add the pair (τ, τ') to result set A. Finally, result set A is returned.

4.4 Complexity Analysis

Let P_θ denote the scanned trajectory set for each trajectory search, which includes the partly and fully scanned trajectories.

Algorithm 3: Merging Algorithm

```
   Data: {C(τ)|∀τ ∈ P}, θ
   Result: A = {(τ, τ')|Sim_ST(τ, τ') ≥ θ, ∀τ, τ' ∈ P }
1  for each trajectory τ in P do
2      for each τ' in C(τ) do
3          if τ ∈ C(τ') then
4              compute Sim_ST(τ, τ');
5              C(τ').remove(τ);
6              if Sim_ST(τ, τ') ≥ θ then
7                  A.add(τ, τ');
8              else
9                  break;
10 return A;
```

According to Equations 15, 18, and 19, 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 $O(|P||P_\theta|)$. The time complexity for the merging phase is $O(|P||C|)$, where $|C|$ is the cardinality of candidate set for each trajectory. Since $C \subseteq P_\theta \subseteq P$, the time complexity of the two-phase algorithm is $O(|P||P_\theta|) + O(|P||C|) = O(|P||P_\theta|)$. If θ is sufficiently large, the time complexity is close to $O(|P|)$.

5. EXTENSION

We proceed to explain how to extend the proposed algorithms to support the case where $P \neq Q$. First, TF-Matching can support $P \neq Q$ 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 $|\tau_{avg}|$ and $|\tau'_{avg}|$ denote the average numbers of samples in trajectories in P and Q, respectively. Then, the time complexity of TF-Matching is

$$O\left(\sum_{\tau \in P, \tau' \in Q} |\tau||\tau'|(|V|\log|V| + |E|)\right)$$
$$= O(|P||Q||\tau_{avg}||\tau'_{avg}|(|V|\log|V| + |E|)),$$

which is not sensitive to the pruning effectiveness.

Second, the two-phase algorithm conducts trajectory searches for all trajectories in P and Q and maintains candidate sets for all of them. The time complexity of the trajectory-search phase is $O(|P||P_\theta| + |Q||Q_\theta|)$. For the merging phase, the time complexity is still $O(|P||C_p|)$ (or $O(|Q||C_q|)$), $C_p \subseteq Q_s \subseteq Q$, and $C_q \subseteq P_s \subseteq P$. The time complexity of the two-phase algorithm is then $O(|P||P_\theta| + |Q||Q_\theta| + |P||C_p|) = O(|P||P_\theta| + |Q||Q_\theta|)$, which is sensitive to the pruning effectiveness.

6. EXPERIMENTAL RESULTS

We report on extensive 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]. They contain 28,342 vertices and 27,690 edges, and 95,581 vertices and 260,855

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

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[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 baseline algorithm (cf. Section 3) and the spatial networks of the two-phase algorithm (when running Dijkstra's expansion [9], cf. Section 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 test its performance in Figure 5 (Figures 4, and 6–9 are for memory-based algorithms). For the baseline algorithm, for each node we store the ids (entries) of the trajectories that overlap the timespan indicated by the node. For the two-phase algorithm, for each vertex in the network we store the ids (entries) of the trajectories that contains 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 trajectories stored in the node/vertex. Next, we traverse the B+-tree and load all of the pages that contains 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 CPU time and the number of visited trajectories. The number of visited trajectories is used as a metric since it reflects the number of data accesses. In multi-threaded execution, the total runtime is the maximum runtime among all individual threads.

We study the performance of the non-self joins, i.e., $P \neq Q$, in Sections 6.3–6.5 and of self joins in Section 6.7. Trajectories in P and Q are selected randomly from the real data sets. The parameter settings are listed in Table II. Because computing network distances online is time-consuming, we pre-computed the all-pair shortest path distances in the graph (for TF-Matching only, not for the two-phase algorithm) and denote the accelerated TF-Matching (Section 3) by "TF-A" in subsequent figures. The two-phase algorithm (Section 4) is denoted by "two-phase," and the two-phase algorithm without the heuristic scheduling strategy is denoted by "two-phase-w/o-h."

By default, TF-Matching uses a uniform leaf-node partitioning scheme. We conduct experiments to find the optimal partitioning. When a leaf node is set to 1 hour in BRN and to 15 minutes in NRN, and each node (including leaf and non-leaf nodes) contains at most 6,560 trajectories in BRN and at most 15,265 trajectories in NRN, the index achieves the highest performance. We also consider a balanced partitioning scheme where each leaf node contains the same/similar numbers of trajectories. When the index contains 32 leaf nodes in BRN and 196 leaf nodes in NRN, and each leaf node contains 1,032 trajectories in BRN and 3,875 trajectories in NRN on average, the index achieves the highest performance. The balanced partitioning method is denoted by "TF-A-balance." According to the experimental results in Figures 4–9, the performance of TF-Matching is improved by around 20% when using balanced partitioning.

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

Table II: 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	1.8–1.95/default 1.9	1.8–1.95/default 1.9
Preference parameter λ	0.1–0.9/default 0.5	0.1–0.9/default 0.5
Thread count m	24–120/default 24	24–120/default 24

6.2 Pruning Effectiveness

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

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

$$Pruning\ ratio = 1 - Candidate\ ratio,$$

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 1/2 – 1/3 of that of TF-Matching and that, with the help of the heuristic scheduling strategy, the candidate ratio is improved by a factor of around 1.5.

Table III: Pruning Effectiveness for Non-Self Join

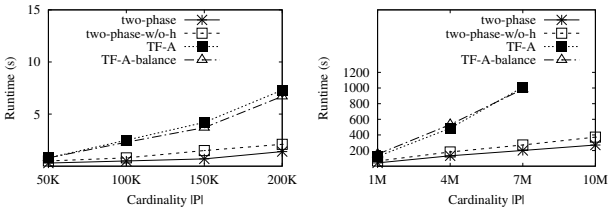
	TF	two-phase	two-phase-w/o-h
Candidate ratio (BRN)	0.17	0.10	0.14
Pruning ratio (BRN)	0.83	0.90	0.86
Candidate ratio (NRN)	0.12	0.04	0.06
Pruning ratio (NRN)	0.88	0.96	0.94

Table IV: Pruning Effectiveness for Self Join

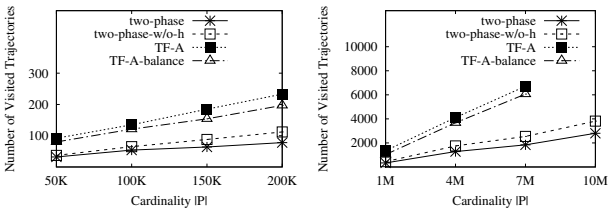
	TF	two-phase	two-phase-w/o-h
Candidate ratio (BRN)	0.11	0.06	0.09
Pruning ratio (BRN)	0.89	0.94	0.91
Candidate ratio (NRN)	0.08	0.03	0.04
Pruning ratio (NRN)	0.92	0.97	0.96

6.3 Effect of Trajectory Cardinalities

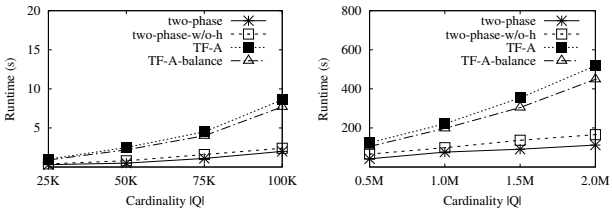
Figure 4 shows the effect of trajectory cardinalities $|P|$ and $|Q|$ on the performance of the algorithms. Intuitively, a larger $|P|$ (or $|Q|$) causes more trajectory pairs to be processed (refer to the complexity analysis in Sections 3.5 and 4.4), meaning that the CPU time and the number of visited trajectories 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 1.5 in terms of both CPU time and the number of visited trajectories. The two-phase algorithm is able to process 1 M trajectories ($|P|$ = 1 M and $|Q|$ = 0.5 M) in 38 seconds and 10 M trajectories ($|P|$ = 10 M and $|Q|$ = 0.5 M) in 252 seconds on default 24 threads (see Figure 4(b)).



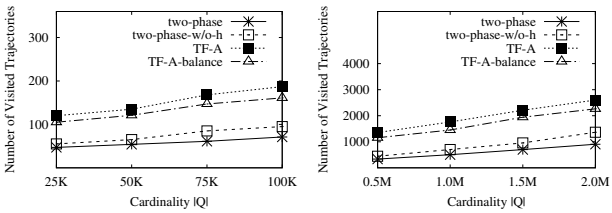
(a) BRN (b) NRN



(c) BRN (d) NRN

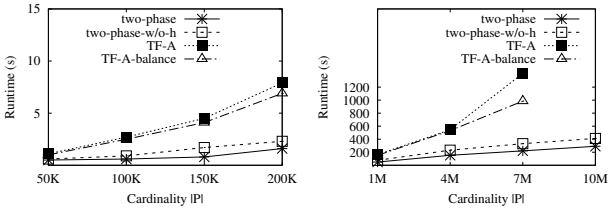


(e) BRN (f) NRN

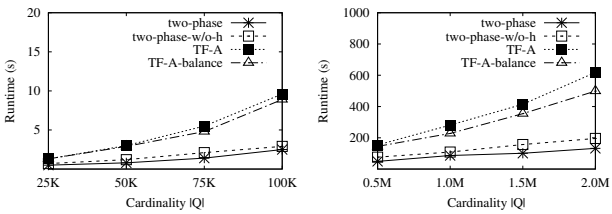


(g) BRN (h) NRN

Figure 4: Effect of trajectory cardinalities $|P|$ and $|Q|$

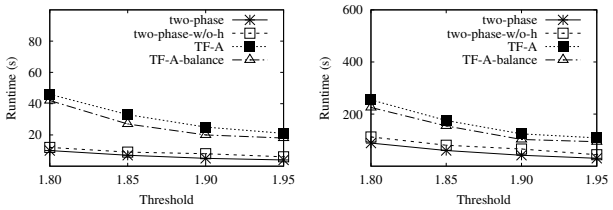


(a) BRN (b) NRN

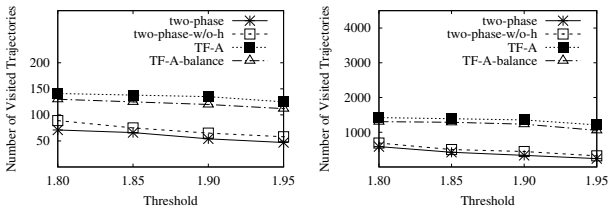


(c) BRN (d) NRN

Figure 5: Effect of disk-based approach

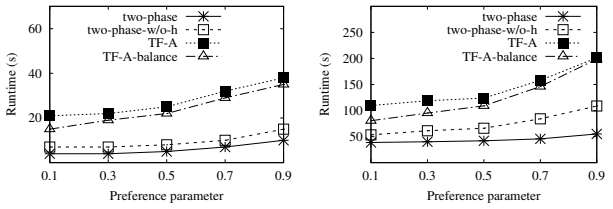


(a) BRN (b) NRN



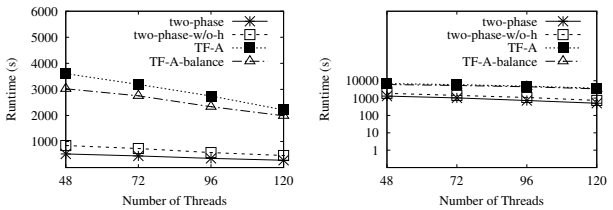
(c) BRN (d) NRN

Figure 6: Effect of threshold θ



(a) BRN (b) NRN (c) BRN (d) NRN

Figure 7: Effect of λ



(a) Non-self join (b) Self join

Figure 8: Effect of thread count m

The CPU time is not fully aligned with the number of visited trajectories because the algorithms expend computational effort on maintaining the bounds used to prune the search space. The resulting cost may offset the benefits of the reduction in the number of visited trajectories. 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 that of memory-based algorithms (Figure 4). Disk-based algorithms may need more CPU

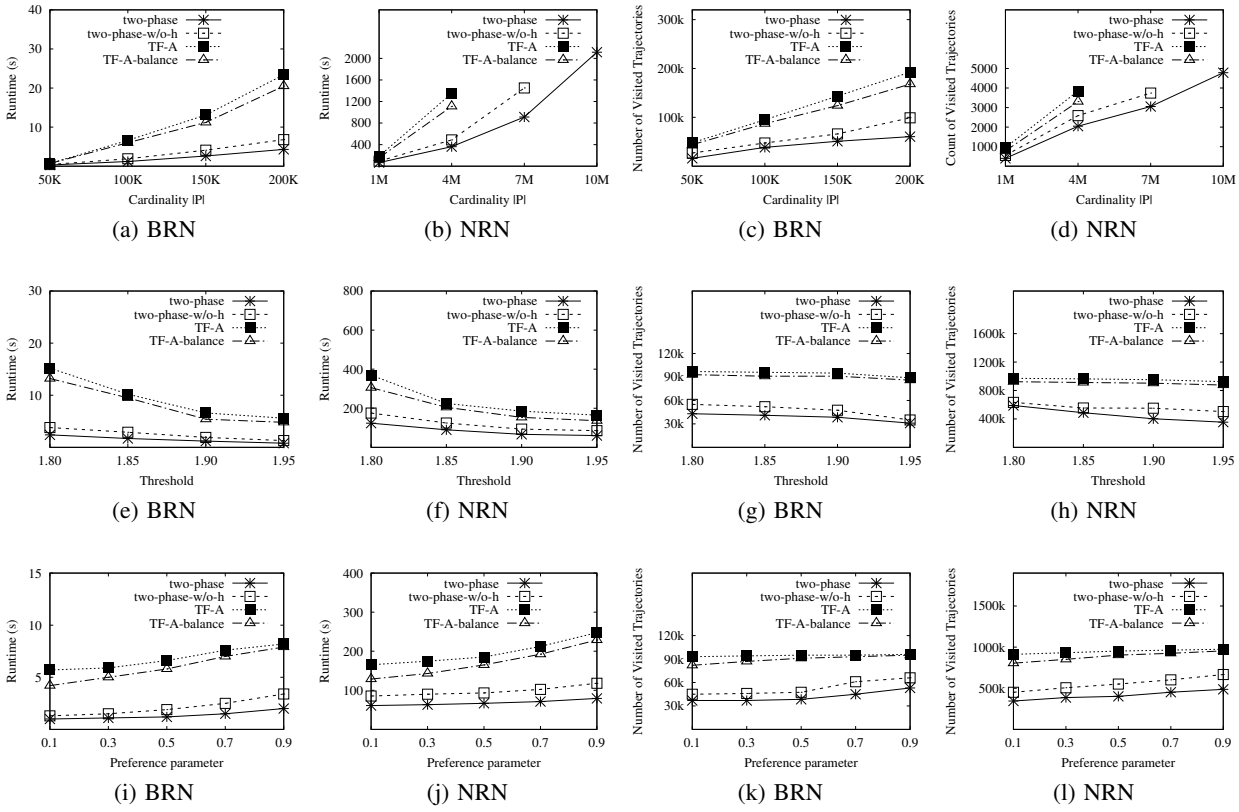


Figure 9: Performance for self join

times 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 in 300 seconds, see Figure 5(b)). Notice that the number of visited trajectories is independent of where the data is stored.

6.4 Effect of Threshold θ

Next, we vary the threshold θ in Figure 6. For the two-phase algorithm, a larger θ leads to higher pruning effectiveness (refer to Equation 4.4). Thus, the larger θ becomes, the smaller the search space becomes, and the required CPU time and the number of visited trajectories are expected to decrease correspondingly. When $\theta = 1.95$, the two-phase algorithm is able to process 1M trajectories ($|P|$ = 1 M and $|Q|$ = 0.5 M) in 28 seconds. In TF-Matching, a larger θ does not help pruning the search space (refer to Equation 3.5), and only slightly fewer trajectories are visited when θ increases. In contrast, a larger θ is useful in reducing the similarity computation (see Equation 7). Thus, the CPU times of TF-A, and TF-A-balance decrease when θ increases.

6.5 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 Equation 5). When λ = 1, the TS-Join is in the spatial domain only, and when λ = 0, only temporal similarity is considered. Figure 7 shows that the spatial domain needs more search effort than the temporal domain.

6.6 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 join and $|P|$ = 10 M for self join).

From the results shown in Figure 8, we see that the two-phase algorithm outperforms TF-Matching by almost an order of magnitude in term of CPU time. For the non-self join, the two-phase algorithm is able to process 10 M $\times$ 2 M trajectories with 120 threads in 255 seconds, while for the self join, the two-phase algorithm is able to process 10 M $\times$ 10 M trajectories with 120 threads in 540 seconds.

In Figure 8, we increase the thread count m from 48 to 120 (2.5 times). This improves the CPU time of the two-phase algorithm by a factor of 1.9–2.2, while the CPU time of TF-Matching (TF-A and TF-A-balance) is improved by a factor of around 1.6. The main reason for the smaller improvement is that more threads (more leaf nodes) leads to a higher merging cost (cf. Section 3.6).

6.7 Performance for Self Join

Figure 9 shows the runtime and number of visited trajectories for the self joins 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 join. The two-phase algorithm outperforms TF-Matching (TF-A and TF-A-balance) by almost an order of magnitude in terms of both CPU time and the number of visited trajectories, and the heuristic search strategy improves the efficiency by almost a factor of 1.5.

7. RELATED WORK

7.1 Trajectory Similarity Search

Trajectory similarity search [7, 16, 17, 22] 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 $UOTS$ [16] and $ATSQ$ [22] extend these to spatial and textual domains, while PTM [17] extends them into spatial and temporal domains. Next, several similarity functions exist for trajectory or time-series data, including Dynamic Time Warping [21], 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, 16, 17] are asymmetrical; thus, they cannot be used directly in the TS-Join.

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-Join uses spatiotemporal similarity, taking into account both spatial and temporal proximities in a continuous manner. Thus, the existing time-interval based solutions are not suitable for the TS-Join (e.g., the temporal-first matching, cf. Section 3). Moreover, in contrast to most existing trajectory join studies (e.g., [2, 3, 6, 10, 19]), the TS-Join is 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, 19]) 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-Join efficiently on very large trajectory data sets.

8. CONCLUSION AND FUTURE WORK

We studied a novel trajectory similarity join (TS-Join) in spatial networks, which targets many applications such as trajectory duplicate detection, data cleaning, ridesharing/carpooling recommendation, and traffic congestion prediction. To process the TS-Join efficiently, a two-phase algorithm was developed based on a divide-and-conquer strategy. The algorithm enables parallel processing. We also proposed an upper bound and a heuristic scheduling strategy to prune the search space effectively. The performance of the TS-Join was investigated through extensive experiments on very large trajectory data.

Three interesting directions for future research exist. First, it is of interest to extend the existing algorithms to support a top-k TS-Join without a threshold θ. Challenges include how to design the parallelism and how to communicate between different threads. Second, it is of interest to take the visiting sequence of sample points into account when matching trajectories. When doing this, the upper bound and heuristic scheduling strategy need to be reworked. Third, it is of interest to study more systems-level optimizations in the TS-Join.

9. ACKNOWLEDGEMENT

This work is partially supported by KAUST, the National Natural Science Foundation of China (61402532, 61532018), Beijing Nova Program (xx2016078), and by the DiCyPS center, funded by Innovation Fund Denmark.

10. REFERENCES

[1] R. Agrawal, K. Lin, H. S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *VLDB*, pages 490–501, 1995.

[2] P. Bakalov, M. Hadjieleftheriou, E. J. Keogh, and V. J. Tsotras. Efficient trajectory joins using symbolic representations. In *MDM*, pages 86–93, 2005.

[3] P. Bakalov and V. J. Tsotras. Continuous spatiotemporal trajectory joins. In *GSN*, pages 109–128, 2006.

[4] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. In *VLDB*, pages 853–864, 2005.

[5] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, pages 491–502, 2005.

[6] Y. Chen and J. M. Patel. Design and evaluation of trajectory join algorithms. In *ACM-GIS*, pages 266–275, 2009.

[7] Z. Chen, H. T. Shen, X. Zhou, Y. Zheng, and X. Xie. Searching trajectories by locations: an efficiency study. In *SIGMOD*, pages 255–266, 2010.

[8] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 2008.

[9] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Math*, 1:269–271, 1959.

[10] H. Ding, G. Trajcevski, and P. Scheuermann. Efficient similarity join of large sets of moving object trajectories. In *TIME*, pages 79–87, 2008.

[11] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[12] Y. Jiang, G. Li, J. Feng, and W. Li. String similarity joins: An experimental evaluation. *PVLDB*, 7(8):625–636, 2014.

[13] W. Luo, H. Tan, L. Chen, and L. M. Ni. Finding time period-based most frequent path in big trajectory data. In *SIGMOD*, pages 713–724, 2013.

[14] S. Shang, L. Chen, C. S. Jensen, J.-R. Wen, and P. Kalnis. Searching trajectories by regions of interest. *IEEE Trans. Knowl. Data Eng.*, online first:1–14, 2017.

[15] S. Shang, L. Chen, Z. Wei, C. S. Jensen, J. Wen, and P. Kalnis. Collective travel planning in spatial networks. *IEEE Trans. Knowl. Data Eng.*, 28(5):1132–1146, 2016.

[16] S. Shang, R. Ding, B. Yuan, K. Xie, K. Zheng, and P. Kalnis. User oriented trajectory search for trip recommendation. In *EDBT*, pages 156–167, 2012.

[17] S. Shang, R. Ding, K. Zheng, C. S. Jensen, P. Kalnis, and X. Zhou. Personalized trajectory matching in spatial networks. *VLDB J.*, 23(3):449–468, 2014.

[18] S. Shang, K. Zheng, C. S. Jensen, B. Yang, P. Kalnis, G. Li, and J. Wen. Discovery of path nearby clusters in spatial networks. *IEEE Trans. Knowl. Data Eng.*, 27(6):1505–1518, 2015.

[19] N. Ta, G. Li, and J. Feng. Signature-based trajectory similarity join. *IEEE Trans. Knowl. Data Eng.*, online first:1–14, 2017.

[20] C. Wenk, R. Salas, and D. Pfoser. Addressing the need for map-matching speed: Localizing globalb curve-matching algorithms. In *SSDBM*, pages 379–388, 2006.

[21] B. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *ICDE*, pages 201–208, 1998.

[22] K. Zheng, S. Shang, N. J. Yuan, and Y. Yang. Towards efficient search for activity trajectories. In *ICDE*, pages 230–241, 2013.