Parallel Semantic Trajectory Similarity Join

Lisi Chen $^{\flat 1}$, Shuo Shang $^{\flat 2}$ *, Christian S. Jensen $^{\ddagger 3}$, Bin Yao $^{\sharp 4}$, Panos Kalnis $^{\dagger 5}$

$^{\flat}$ *UESTC, China* {[1]chenlisi.cs; [2]jedi.shang}@gmail.com

‡ *Aalborg University, Denmark* [3]csj@cs.aau.dk

$^{\sharp}$ *Shanghai Jiao Tong University, China* [4]yaobin@cs.sjtu.edu.cn

† *KAUST, Saudi Arabia* [5]panos.kalnis@kaust.edu.sa

***Abstract*—Matching similar pairs of trajectories, called trajectory similarity join, is a fundamental functionality in spatial data management. We consider the problem of semantic trajectory similarity join (STS-Join). Each semantic trajectory is a sequence of Points-of-interest (POIs) with both location and text information. Thus, given two sets of semantic trajectories and a threshold θ, the STS-Join returns all pairs of semantic trajectories from the two sets with spatio-textual similarity no less than θ. This join targets applications such as term-based trajectory near-duplicate detection, geo-text data cleaning, personalized ridesharing recommendation, keyword-aware route planning, and travel itinerary recommendation.**

With these applications in mind, we provide a purposeful definition of spatio-textual similarity. To enable efficient STS-Join processing on large sets of semantic trajectories, we develop trajectory pair filtering techniques and consider the parallel processing capabilities of modern processors. Specifically, we present a two-phase parallel search algorithm. We first group semantic trajectories based on their text information. The algorithm's per-group searches are independent of each other and thus can be performed in parallel. For each group, the trajectories are further partitioned based on the spatial domain. We generate spatial and textual summaries for each trajectory batch, based on which we develop batch filtering and trajectory-batch filtering techniques to prune unqualified trajectory pairs in a batch mode. Additionally, we propose an efficient divide-and-conquer algorithm to derive bounds of spatial similarity and textual similarity between two semantic trajectories, which enable us prune dissimilar trajectory pairs without the need of computing the exact value of spatio-textual similarity. Experimental study with large semantic trajectory data confirms that our algorithm of processing semantic trajectory join is capable of outperforming our well-designed baseline by a factor of 8–12.

I. Introduction

Trajectory similarity join is a fundamental functionality in trajectory data analytics (e.g., TS-Join [31], Strain-Join [38], DITA [36], DTJ [39]): Given sets T and Q of trajectories and a similarity threshold θ, finds all pairs of trajectories from T and Q with a similarity that no less than θ. Based on existing studies, the similarity between two trajectories is either measured by the spatial proximity [36], [38] or measured by spatio-temporal proximity [31], [39] between them, underlying that the join operation is conducted in spatial or spatio-temporal domain only.

However, recent years have witnessed the development of online map-based services (e.g., Bing Maps[1], Google Maps[2], Foursquare[3], and MapQuest[4]) and location-based social media (e.g., Twitter[5], Facebook[6], Instagram[7], and Flickr[8]). These applications are redefining and enriching the traditional trajectory data by associating locations with semantic meanings [48]. For example, Foursquare users can check in the Point-of-Interests (POI, which contains a spatial location and a semantic description) they are visiting and leave descriptions and comments for other users. Flickr allows tourists to upload their geo-text photos taken by smartphones during the travel, so that their trips can be outlined both by location and by semantic information embedded in the photos. Trajectories generated in these applications are defined as *semanic trajectories* [27]. They can be modeled either by a finite sequence of text-embedded geographical points (i.e., geo-textual objects, POIs) [17] or by a traditional trajectory enriched by a text document [34]. From a semantic trajectory, we can know not only where a user has been, as from the traditional trajectory, but also what he/she has done by retrieving the text documents associated with the locations (e.g., reviews, descriptions, text annotations), or by extracting the text information from the multimedia contents attached to the locations (e.g., images, videos).

To enable effective analytics of big semantic trajectory data, it is insufficient to evaluate the similarity between semantic trajectories solely based on spatial proximity. Consider semantic trajectories τ_1–τ_4 in Figure 1. The yellow diamonds represent the POIs covered by these trajectories. We see that trajectory τ_1 covers a couple of tolled road segments, τ_2 covers nearby free road segments with lake sightseeing, and τ_3 covers some off-road segments. Although these three routes are spatially close to each other, they are preferred by diverging groups. In particular, τ_1 is likely to be chosen by business travelers and commuters in a rush, τ_2 is favorable to some budget-sensitive tourists, and τ_3 may be taken by adventurous travelers who have off-road vehicles. Further, we see that τ_4 is more similar to τ_1 and τ_3 than to τ_2 in terms of spatial proximity. However, taking both trajectory spatial information and semantic (textual) information of POIs into account, we may find that travelers who choose τ_2 are more likely to favor τ_4 over τ_1 and τ_3. This example underlines that it is important to consider both spatial proximity and semantic similarity when determining whether two trajectories are similar with each other.

* Shuo Shang is the corresponding author.

[1]https://www.bing.com/maps/

[2]https://maps.google.com/

[3]https://foursquare.com

[4]https://www.mapquest.com

[5]https://twitter.com

[6]https://www.facebook.com

[7]https://instagram.com

[8]https://flickr.com

2375-026X/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDE48307.2020.00091

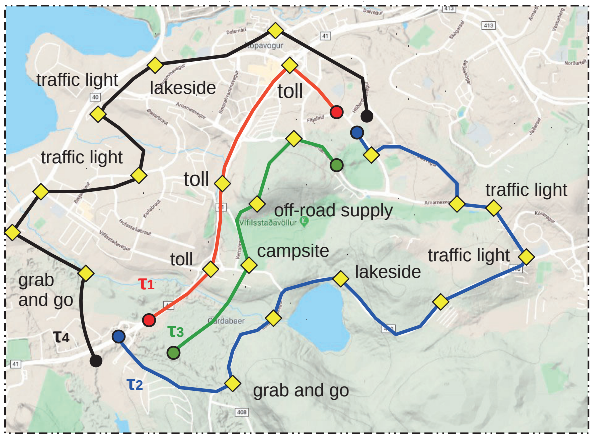


Fig. 1. An example of semantic trajectories

In this paper, we focus on a fundamental functionality in semantic trajectory data analytics, *Semantic Trajectory Similarity Join* (STS-Join): Given sets T and Q of semantic trajectories (or a single set T and its mirror Q in the case of self-join) and a similarity threshold θ, the STS-Join returns all pairs of semantic trajectories from T and Q with a spatio-textual similarity that no less than θ. Specifically, both spatial and textual domains are considered in measuring the similarity between two semantic trajectories. A linear combination method (e.g., [34], [47]) is adopted to combine the spatial and textual similarity into a spatio-textual similarity metric.

To process the STS-Join efficiently, we develop a two-phase parallel matching framework that consists of the following techniques:

(1) *Trajectory grouping and batch filtering:* Given sets T and Q of trajectories, we first group trajectories by a purposeful inverted index based on their text information. The inverted index takes into account the parallel processing capabilities of modern processors. We prove that join processes of trajectories in different groups (lists) are independent of each other. Next, trajectories in each group are further partitioned according to the spatial domain. We generate spatial and textual summaries for each trajectory batch, based on which we propose batch filtering and trajectory-batch filtering techniques to prune unqualified trajectory pairs in a batch mode. (Sections IV-D)

(2) *Semantic trajectory pair pruning:* It is computationally expensive to calculate the exact spatio-textual similarity between two semantic trajectories (see Section III-C for complexity analysis). To address the challenge, we propose an efficient divide-and-conquer algorithm (see Section IV-C for detailed complexity analysis) to derive upper bounds of spatial similarity and textual similarity between two semantic trajectories, which enable us prune dissimilar trajectory pairs without the need of computing the exact value of spatio-textual similarity. (Sections IV-B and IV-C)

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

- We present the first study on semantic trajectory similarity join, named STS-Join, that takes into account both spatial similarity and text similarity between trajectories.
- The STS-Join uses new metrics to evaluate semantic trajectory similarity in the spatial and textual domains.
- We develop a two-phase parallel matching framework consisting of semantic trajectory grouping, batch filtering, and trajectory pair pruning techniques that enable efficient parallel STS-Join processing.
- We conduct extensive experiments on large semantic trajectory data sets to study the performance of the algorithms.

The rest of the paper is organized as follows. Section II introduces two definitions of semantic trajectories and the similarity metrics used in the paper, and it defines the STS-Join problem. Section III presents the baseline algorithm for processing the STS-Join. Our two-phase parallel matching algorithm and corresponding filtering and pruning techniques are presented in Section IV. Next, we present our experimental study in Section V. Related work is covered in Section VI, and conclusions are presented in Section VII.

II. Preliminaries and Problem Statement

A. Semantic trajectories

We present two popular definitions of semantic trajectories: POI-based semantic trajectory (e.g., [13], [17], [47], [48]) and document-based semantic trajectory (e.g., [34]).

Definition 1: (POI-based semantic trajectory) A POI-based semantic trajectory τ is a directed finite sequence of POIs $\langle o_1, o_2, ..., o_n\rangle$, where $o_i = \{\rho, \psi\}$ is a geo-textual object that contains a geographical location ρ and a text description ψ. □

Definition 2: (Document-based semantic trajectory) A document-based semantic trajectory $\tau = \{\mathbf{p}, \psi\}$ is defined by a directed finite location sequence $\mathbf{p} = \langle p_1, p_2, ..., p_n\rangle$ and a text document ψ. □

For document-based semantic trajectory, the spatial information (location sequence) and text information are independent of each other. In contrast, the text information of POI-based semantic trajectory is associated with a particular POI, making the trajectory pair matching process be more challenging. As a result, in the remaining parts of this paper we focus on the similarity join of POI-based semantic trajectory. Our proposal can be easily extended to support document-based semantic trajectory similarity join.

B. Similarity measures

In this section, we first present the definition of relevance between a geo-textual object and a semantic trajectory, based on which we present how to measure the similarity between two semantic trajectories.

1) Object-trajectory relevance: Given a geo-textual object o and a semantic trajectory τ, the spatio-textual relevance $\mathsf{R}(o, \tau)$ between o and τ are defined as follow. [9]

$$\mathsf{R}(o,\tau) = \max_{o_i \in \tau}\{\mathrm{Sim}(o, o_i)\} \tag{1}$$

[9] Unless otherwise stated, semantic trajectory denotes POI-based semantic trajectory.

where $\text{Sim}(o, o_i)$ denotes the spatio-textual similarity between o and o_i, which is computed by a linear combination of spatial proximity and text similarity [5], [18], [43], [44]:

$$\text{Sim}(o_1, o_2) = \alpha \cdot \mathsf{S}(o_1.\rho, o_2.\rho) + (1 - \alpha) \cdot \mathsf{T}(o_1.\psi, o_2.\psi) \quad (2)$$

where $\mathsf{S}(o_1.\rho, o_2.\rho)$ denotes the normalized spatial score between $o_1.\rho$ and $o_2.\rho$, which is inversely proportional to the distance, and $\mathsf{T}(o_1.\psi, o_2.\psi)$ denotes the normalized text similarity (e.g., cosine similarity, Jaccard similarity) between $o_1.\psi$ and $o_2.\psi$. Without loss of generality, we adopt Jaccard similarity (cf. Equation 3), which is a widely-used model to measure the similarity between geo-textual objects [10], [15], [18], [22]. It is important to note that our proposal is not specific to the Jaccard similarity measurement. It is straightforward to extend our method to other similarity measure such as cosine similarity [40], [44]. Parameter α balances the weight between spatial proximity and text similarity.

$$\mathsf{T}(o_1.\psi, o_2.\psi) = \frac{|o_1.\psi \cap o_2.\psi|}{|o_1.\psi \cup o_2.\psi|} \quad (3)$$

2) Trajectory-trajectory similarity: Given semantic trajectories τ_1 and τ_2, we define the spatio-textual similarity $\mathsf{ST}(\tau_1, \tau_2)$ between them based on an existing study [12].

$$\mathsf{ST}(\tau_1, \tau_2) = \frac{\sum_{o_i \in \tau_1} \mathsf{R}(o_i, \tau_2)}{|\tau_1|} + \frac{\sum_{o_j \in \tau_2} \mathsf{R}(o_j, \tau_1)}{|\tau_2|} \quad (4)$$

Here, τ denotes the number of geo-textual objects in a semantic trajectory. We ensure that the similarity measures are symmetrical, such that $\mathsf{ST}(\tau_1, \tau_2) = \mathsf{ST}(\tau_2, \tau_1)$.

C. Problem Definition

We formally define the STS-Join problem in Definition 3.

Definition 3: (STS-Join) Given sets T and Q of semantic trajectories and a similarity threshold θ, the semantic trajectory similarity join (STS-Join) finds a set A of all semantic trajectory pairs (τ_i, τ_j) from the two sets that satisfy the following conditions:

(1) τ_i and τ_j share at least one common term, and

(2) The spatio-textual similarity between τ_i and τ_j is no less than θ. □

Condition (1) in Definition 3 is to guarantee that the trajectory pairs in each join result have some minimum textual relevancy to each other. Specifically, if two semantic trajectories τ_1 and τ_2 are to be considered as similar, a very natural necessary condition is that τ_1 and τ_2 have some minimum overlap in both the spatial and textual domains. This setting has been applied widely in existing studies that measure the similarity between two geo-textual items (e.g., [3], [16], [21]).

Complexity analysis. We assume that the number of objects in each trajectory is $|\tau_{avg}|$ and that the number of terms per object is $|\psi_{avg}|$. According to Equations 1 and 4, to compute $\mathsf{ST}(\tau, \tau_j)$ we need to calculate the spatio-textual similarity between each pair of objects in τ and τ_j. So the time complexity of computing $\mathsf{ST}(\tau, \tau_j)$ is $O(|\psi_{avg}| \cdot |\tau_{avg}|^2)$. In the worst case, we need to evaluate all pairs of trajectories. Thus, the time complexity of the STS-Join is $O(|\psi_{avg}| \cdot |\tau_{avg}|^2 \cdot |T|^2)$.

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

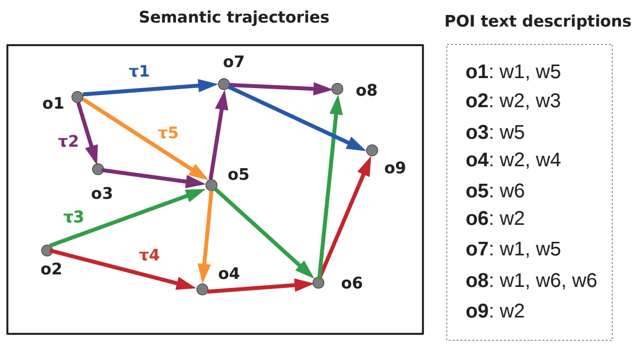
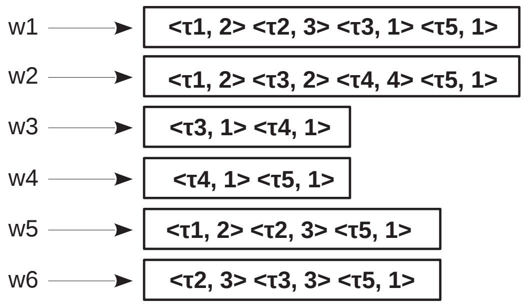


Fig. 2. An example of semantic trajectory collection



w1	<τ1, 2> <τ2, 3> <τ3, 1> <τ5, 1>
w2	<τ1, 2> <τ3, 2> <τ4, 4> <τ5, 1>
w3	<τ3, 1> <τ4, 1>
w4	<τ4, 1> <τ5, 1>
w5	<τ1, 2> <τ2, 3> <τ5, 1>
w6	<τ2, 3> <τ3, 3> <τ5, 1>

Fig. 3. Trajectory inverted file

III. Baseline Approach

A. Basic idea

Search with Trajectory Inverted File (STIF) is a straightforward baseline approach to computing the STS-Join. Based on Definition 3, trajectory pair τ_i and τ_j can be a join result only if they share at least one common term. In order to filter out trajectory pairs that do not share any term, we index the trajectory terms using inverted index. In particular, each inverted list, denoted by $IL(w)$, corresponds to a term w and it stores trajectories whose text descriptions contain w. Then we evaluate each trajectory pair that shares at least one common term. Specifically, for each trajectory τ we evaluate potential result pairs (τ, τ') by traversing all inverted lists $IL(w_i)$ where $w_i \in W$ and $W = \cup_{o_j \in \tau} o_j.\psi$. We compute $\mathsf{ST}(\tau, \tau')$ and compare it with θ.

B. Trajectory Inverted File

Figure 2 presents a collection of semantic trajectories. Gray circles o_1–o_9 are POIs and τ_1–τ_5 are semantic trajectories. Each trajectory is illustrated by a unique color. The text description of each POI is presented on right-hand side.Figure 3 presents the trajectory inverted file built for indexing the trajectories in Figure 2. Each posting $\langle \tau_i, n \rangle$ in $IL(w)$ consists of a trajectory id, τ_i, and the frequency of w in τ_i, n. Note that the frequency may be used for computing the spatio-textual similarity between two trajectories, depending on the text similarity metric we use.

C. Algorithm

This section presents our STIF algorithm (Algorithm 1). The inputs are a collection of trajectories T and a similarity threshold θ. The output is a set of trajectory pairs that are similar with each other. For each trajectory τ, we first generate a union of terms in all of its POIs (i.e., W) (Line 2). Next, for each term $w_i \in W$ we traverse its inverted list $IL(w_i)$ and evaluate each posting $\langle \tau_j, n \rangle$ in $IL(w_i)$ (Line 3). In particular, we compute the similarity between τ and τ_j. If the similarity is no less than θ, we add pair (τ, τ_j) to the result set A (Lines 4–7). At the end, we return set A as the final result. In this algorithm, the evaluation of each trajectory τ can be performed in parallel.

Algorithm 1: STIF

```
Data: Semantic trajectory set $T$, Similarity threshold $\theta$
Result: A set of result trajctory pairs $A$
1 for each $\tau$ in $T$ do
2     $W \leftarrow \cup_{o \in \tau} o.\psi$;
3     for each $w_i \in W$ do
4         for each posting $\langle \tau_j, n \rangle \in IL(w_i)$ do
5             $st \leftarrow$ ComputeST($\tau$,$\langle \tau_j, n \rangle$);
6             if $st \geq \theta$ then
7                 $A$.add($\langle \tau, \tau_j \rangle$);
8 return $A$;
```

IV. Two-Phase Parallel Matching

A. Basic Idea

We aim to address the following limitations exist in STIF algorithm:

(1) *High complexity of computing similarity between two trajectories*: When computing $\mathsf{ST}(\tau_i, \tau_j)$ we need to calculate the similarity between object pair $\langle o_i, o_j \rangle$ s.t. $o_i \in \tau_i$ and $o_j \in \tau_j$. It is computationally expensive especially when the average number of objects in a trajectory is large.

(2) *Individual evaluation*: In STIF, each trajectory pair is evaluated separately, which is time consuming given a large number of semantic trajectories.

To address the above two limitations, we develop a Two-phase Parallel Matching (2-PM) algorithm for processing STS-Join. Specifically, to address limitation (1) we propose an trajectory pair pruning strategy to prune "unqualified" trajectory pairs without the need of calculating the similarity between each object pair in τ_i and τ_j. To address limitation (2), we propose a parallelized batch processing algorithm that is able to evaluate a group of trajectory pairs simultaneously.

Figure 4 presents the framework of our 2-PM algorithm. Given a collection of semantic trajectories T, we generate summary for each trajectory to represent its spatial and textual information (cf. Section IV-B). Based on the summary information we group similar trajectories together and conduct batch filtering (cf. Section IV-D). Specifically, if a pair of batches (groups) cannot be pruned, we proceed to evaluate each trajectory pair in the two batches by using our trajectory pair filtering technique (cf. Section IV-C). Note that both batch filtering and trajectory pair filtering can be processed in parallel.

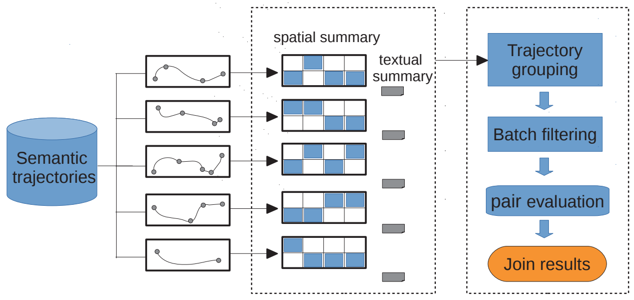


Fig. 4. Framework of 2-PM

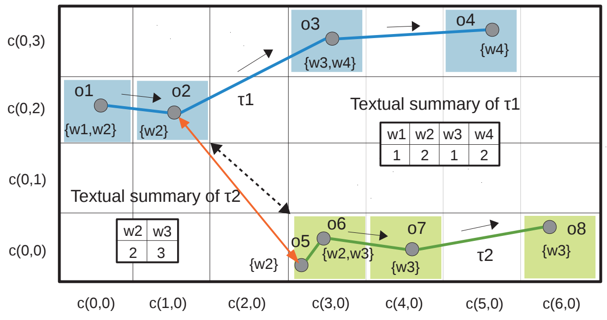


Fig. 5. Summary of trajectories

B. Semantic Trajectory Summarization

This section presents how to generate a summarization of a semantic trajectory. Recall that the time complexity of computing the spatio-textual similarity between two trajectories is $O(|\psi_{avg}| \cdot |\tau_{avg}|^2)$. With the summarizations of trajectories τ_i and τ_j, we can derive an upper bound of spatio-textual similarity between τ_i amd τ_j with $O(|\tau_{avg}| \cdot \log |\tau_{avg}| + |\tau_{avg}| \cdot |\psi_{avg}|)$.[10]

Figure 5 presents an example of summarizing trajectories $\tau_1 = \langle o_1, o_2, o_3, o_4 \rangle$ and $\tau_2 = \langle o_5, o_6, o_7, o_8 \rangle$. We summarize the spatial information and textual information of trajectories separately. For summarizing spatial information, we partition the underlying space into $m \times n$ (i.e., 7×4 in Figure 5) cells. Each object is associated with a cell $c(x, y)$. As a result, trajectory τ_1 is associated with $c(0, 2)$, $c(1, 2)$, $c(3, 3)$, and $c(5, 3)$, and trajectory τ_2 is associated with $c(3, 0)$, $c(4, 0)$, and $c(6, 0)$. As for summarizing textual information, we aggregate the term frequencies of all objects in each trajectory (see textual summaries of τ_1 and τ_2 in Figure 5).

C. Trajectory Pair Pruning

We proceed to introduce how to derive upper bounds of spatial similarity and textual similarity between two trajectories, respectively, which enable us prune "dissimilar" trajectory pairs without the need of computing the exact value of spatio-textual similarity.

1) Spatial similarity upper bound: Computing the spatial similarity upper bound between trajectories τ_i and τ_j, denoted by $\mathsf{ST}_S(\tau_i, \tau_j).ub$, is equivalent to computing the lower bound

[10]Assume that the trajectory is not summarized by grid cells.

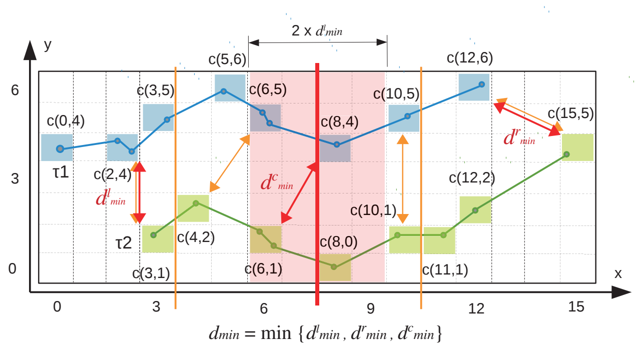


Fig. 6. Computation of spatial similarity upper bound

of minimum distance between τ_i and τ_j, which is denoted by d_{min}.

Figure 6 illustrates the high-level idea of our divide-and-conquer based algorithm of computing d_{min}. We first index τ_1 and τ_2 by grid cells. Next, we recursively calculate the local minimum distances between cells of τ_1 and τ_2 in both subareas divided by the vertical line (i.e., both left and right subareas contain the same number of cells). Specifically, in the first recursion, the red boundary line parallel to y-axis in Figure 6 partitions the underlying space into two subspaces. d^l_{min} and d^r_{min}, represented by double-headed red arrow lines, denotes the local minimum distances regarding the left subspace and right subspace, respectively. After calculating the local minimum distances regarding both subspaces, we compute d^c_{min}, the local minimum distance between cells of τ_1 and τ_2 in the area located at most $\min\{d^l_{min}, d^r_{min}\}$ distance away from the middle line dividing the two subspaces (i.e., the red area in Figure 6).

Algorithm 2 presents our divide-and-conquer algorithm of computing the lower bound of minimum distance given trajectories τ_1 and τ_2. At first, we index τ_1 and τ_2 by grid cells (Lines 1–2). In particular, C_1 and C_2 to denote the sets of grid cells associated with τ_1 and τ_2, respectively, and array A (Line 3) is a list of cells that merges C_1 and C_2. Next, we sort A based on x-coordinate and y-coordinate and generate arrays A_x and A_y, respectively. Finally, we call recursive function MinDistUtil by taking C_1, C_2, A_x, A_y, and $|A|$ as input.

Algorithm 2: MinDist

```
Data: Semantic trajectories $\tau_1$ and $\tau_2$
Result: Upper bound of minimum distance between $\tau_1$ and $\tau_2$
1 $C_1 \leftarrow$ grid cells associated with $\tau_1$;
2 $C_2 \leftarrow$ grid cells associated with $\tau_2$;
3 $A \leftarrow C_1 + C_2$;
4 $A_x \leftarrow C$ sorted by x axis; $A_y \leftarrow C$ sorted by y axis;
5 return MinDistUtil$(C_1, C_2, A_x, A_y, |A|)$;
```

We proceed to present the pseudo code of MinDistUtil in Algorithm 3. Here, integer m represents the total number of cells associated with τ_1 and τ_2 (i.e., $|C_1| + |C_2|$). At first, we check if m is large enough to use divide-and-conquer framework. If $m \leq 3$, we directly compute d_{min} in a brute force manner by enumerating all pairs of cells (Lines 31–32). If $m > 3$, we go with our divide-and-conquer method. Let A^l_x and A^r_x be the first and second half of A_x, respectively (Lines 6–7). We split A_y into A^l_y and A^r_y according to x-coordinate (Lines 8–12). Next, we recursively calculate the local d_{min} regarding A^l_y and A^r_y. In particular, if A^l_y (resp. A^r_y) contains cells associated with both τ_1 and τ_2, we compute the local minimum distance d^l_{min} (resp. d^r_{min}) by calling the next recursion of MinDistUtil; otherwise, we assign the initial value (max) to d^l_{min} (resp. d^r_{min}) as the local minimum distance does not exist (Lines 13–20). Having d^l_{min} and d^r_{min}, we choose the smaller one as the minimum distance regarding the left subspace and right subspace (Line 21). Following that, we compute the local minimum distance between cells close to the boundary line in the current recursion (Lines 22–30). Specifically, we generate array A_m, sorted based on y-coordinate, that contains cells whose distances to the boundary line are closer than d_{min} (Lines 23–25). Then we evaluate all pairs of cells from different trajectories and update the value of d_{min} (Lines 26–30). It is worth noting that the complexity of *cross-boundary computation* (i.e., Lines 26–30) is not $O((|C_1|+|C_2|)^2)$. Instead, it is $O((|C_1|+|C_2|))$ because the inner loop runs at most 7 times. We regard each cell as a point in order to prove the above proposition in a more understandable way (cf. Theorem 1).

Theorem 1: The number of inner loops (Lines 27–30 in Algorithm 3) in the cross-boundary computation is at most 7.

Proof: Given a point p_0 that is visited by a particular outer loop (Lines 26–30 in Algorithm 3). The points visited by the corresponding inner loop (Lines 27–30) must lie either in the left or in the right $d_{min} \times d_{min}$ square. Because that for any two points p_i and p_j, the distance between p_i and p_j is larger than d_{min}, we can pack at most 4 points into one square. Thus, we have 7 points in total expect p_0. ■

Complexity Analysis. Given two sets of cells C_1 and C_2 associated with trajectories τ_1 and τ_2, respectively, the time complexity of MinDist algorithm is $O((|C_1|+|C_2|)\cdot\log(|C_1|+|C_2|))$.

2) Textual similarity upper bound: We proceed to compute the upper bound of text similarity between trajectories τ_1 and τ_2 based on Equation 5.

$$\mathsf{ST}_T(\tau_1,\tau_2).ub = \frac{\sum_{o_i\in\tau_1}\mathsf{R}_T(o_i,\tau_2).ub}{|\tau_1|} + \frac{\sum_{o_j\in\tau_2}\mathsf{R}_T(o_j,\tau_1).ub}{|\tau_2|}, \tag{5}$$

where $\mathsf{R}_T(o_i,\tau).ub$ denotes the upper bound of text relevance between object o_i and trajectory τ. Equation 5 calculates $\mathsf{ST}_T(\tau_1,\tau_2).ub$ by aggregating the upper bound of text relevances between each $o_i \in \tau_1$ and τ_2, and between each $o_i \in \tau_2$ and τ_1.

Algorithm 4 presents the pseudo code of computing the upper bound of text similarity between τ_1 and τ_2. We use $ts(\tau)$ to denote the set of terms in trajectory τ, which can be directly retrieved from the textual summary of τ (cf. Figure 5). We take the text summaries of τ_1 and τ_2 as input. At first, we initialize $TSim$, which maintains the current text similarity (Line 1). Then we aggregate $TSim$ with the upper bound of text relevances between each $o_i \in \tau_1$ and τ_2 (i.e., $\mathsf{R}_T(o_i,\tau_2).ub$) (Lines 2–4). Next, we do the same with the upper bound of text relevances between each $o_i \in \tau_2$ and τ_1 (i.e., $\mathsf{R}_T(o_i,\tau_1).ub$) (Lines 5–7). We can see that the time

Algorithm 3: MinDistUtil

```
   Data: Sets of grid cells $C_1$ and $C_2$, Sorted array of cells $A_x$
         and $A_y$, Integer $m$
   Result: Minimum distance between $C_1$ and $C_2$
1  $d_{min} \leftarrow +\infty$;
2  if $m > 3$ then
3      $n \leftarrow \lfloor \frac{m}{2} \rfloor$;
4      Initialize cell arrays $A_x^l$, $A_x^r$, $A_y^l$, $A_y^r$;
5      $i_l, i_r \leftarrow 0$;
6      $A_x^l \leftarrow A_x$.subarray$(0, n)$;
7      $A_x^r \leftarrow A_x$.subarray$(n+1, m-1)$;
8      for $i$ from $0$ to $m-1$ do
9          if $A_y[i].x \leq A_x[n].x$ then
10             $A_y^l[i] \leftarrow A_y[i]$;
11         else
12             $A_y^r[i] \leftarrow A_y[i]$;
13     if $A_x^l$ contains cells from both $C_1$ and $C_2$ then
14         $d_{min}^l \leftarrow$ MinDistUtil$(C_1, C_2, A_x^l, A_y^l, n)$;
15     else
16         $d_{min}^l \leftarrow +\infty$;
17     if $A_x^l$ contains cells from both $C_1$ and $C_2$ then
18         $d_{min}^r \leftarrow$ MinDistUtil$(C_1, C_2, A_x^r, A_y^r, m-n)$;
19     else
20         $d_{min}^r \leftarrow +\infty$;
21     $d_{min} \leftarrow$ Min$(d_{min}^l, d_{min}^r)$;
22     Initialize cell array $A_m$;
23     for $i$ from $0$ to $m-1$ do
24         if $A_y[i].x - A_x[n].x < d_{min}$ then
25             $A_m$.add$(A_y[i])$;
26     for $i$ from $0$ to $|A_m|-1$ do
27         for $j$ from $i+1$ to $|A_m|-1$ do
28             if $(A_m[i] \in C_1$ and $A_m[j] \in C_2)$ or
               $(A_m[i] \in C_2$ and $A_m[j] \in C_1)$ then
29                 if $dist(A_m[i], A_m[j]) < d_{min}$ then
30                     $d_{min} \leftarrow dist(A_m[i], A_m[j])$;
31 else
32     $d_{min} \leftarrow$ BruteForce$(C_1, C_2, A_x, m)$;
33 return $d_{min}$;
```

Algorithm 4: MaxTSim

```
   Data: Semantic trajectories $\tau_1$,$\tau_2$, Text summaries of $\tau_1$ and
         $\tau_2$
   Result: Text similarity upper bound between $\tau_1$ and $\tau_2$
1  $TSim \leftarrow 0$;
2  for each $o_i \in \tau_1$ do
3      if $o_i.\psi \cap ts(\tau_2) \neq \emptyset$ then
4          $TSim \leftarrow TSim + \frac{1}{|\tau_1|}$;
5  for each $o_i \in \tau_2$ do
6      if $o_i.\psi \cap ts(\tau_1) \neq \emptyset$ then
7          $TSim \leftarrow TSim + \frac{1}{|\tau_2|}$;
8  return $TSim$;
```

complexity of computing $\mathsf{R}_T(o_i, \tau_1).ub$ and $\mathsf{R}_T(o_i, \tau_2).ub$ is $O(|o_i.\psi|)$ and $O(|o_i.\psi|)$, respectively.[11]

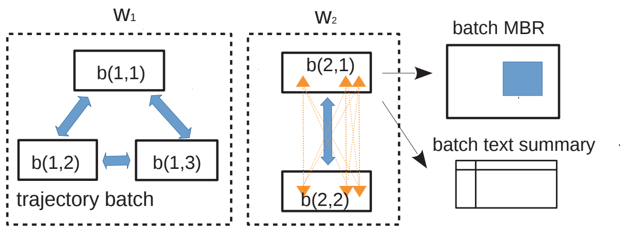


Fig. 7. Trajectory grouping

D. Trajectory Grouping and Batch Filtering

In our semantic trajectory join setting, the number of semantic trajectories can be very large, making it very time consuming to evaluate each trajectory pair individually. So we propose to evaluate groups of spatially and textually similar trajectories simultaneously.

Figure 7 presents our framework of trajectory grouping scheme. Similar to the baseline approach introduced in Section III, semantic trajectories are firstly indexed by an inverted file. Each inverted list maintains trajectories that contain a particular term. For example, in Figure 7, trajectories maintained under w_i must contain term w_i in at least one of their object. Trajectories in each inverted list are grouped into batches based on their spatial information (see Section IV-D4 for grouping algorithm). Here, we use $b(i,j)$ to denote the j-th batch of trajectories in inverted list of term w_i (i.e., $IL(w_i)$). Each batch is augmented with *batch MBR* and *batch text summary*. In particular, the batch MBR is a minimum bounding rectangle that enclose the objects of all trajectories in the batch and the batch text summary summarize the terms of trajectories in the batch.

Based on our trajectory grouping scheme, we develop the join algorithm consisting of the following three phases.

(1) *Intra-list batch filtering*: Given $IL(w_i)$, we first evaluate each pair of batches in $IL(w_i)$ (e.g., $b(i,j)$ and $b(i,k)$) and check if there may exist $\tau \in b(i,j)$ and $\tau' \in b(i,k)$ s.t. $\mathsf{ST}(\tau, \tau') \geq \theta$. If not, batch pair $b(i,j)$ and $b(i,k)$ can be filtered out. Note that it is unnecessary to evaluate batch pair $b(x,j)$ and $b(y,k)$ if $x \neq y$ (cf. Theorem 2). The upper bound of similarity between two batches is introduced in Section IV-D1

(2) *Trajectory-batch filtering*: If batch pair $b(i,j)$ and $b(i,k)$ cannot be filtered, we proceed to evaluate each trajectory $\tau \in b(i,j)$. Specifically, we check if there may exist $\tau' \in b(i,k)$ s.t. $\mathsf{ST}(\tau, \tau') \geq \theta$. If not, trajectory-batch pair τ and $b(i,k)$ can be filtered out. The upper bound of similarity between a trajectory and a batch is introduced in Section IV-D2

(3) *Individual evaluation*: If trajectory-batch pair τ and $b(i,k)$ cannot be filtered, we need to evaluate each trajectory pair of τ and $\forall \tau' \in b(i,k)$. Next, we evaluate each pair in $b(i,j)$. Trajectory pair pruning technique (cf. Section IV-C) is applied in individual evaluation.

[11]The complexity result is based on the fact that the textual summary of each trajectory is maintained by hashmap.

Theorem 2: Given $b(x, j)$ and $b(y, k)$, batch pair $b(x, j)$ and $b(y, k)$ can be filtered out if $x \neq y$.

Proof: Let $\tau_a \in b(x, j)$ and $\tau_b \in b(y, k)$. We consider the following two scenarios: (1) There exist $o_m \in \tau_a$ and $o_n \in \tau_b$ s.t. o_m and o_n share common term w_q; (2) For any $o_m \in \tau_a$ and $o_n \in \tau_b$, o_m and o_n do not share common term. For scenario (1), we can deduce that both τ_a and τ_b are indexed by $IL(w_q)$. Thus, they must be evaluated in intra-list batch filtering phase. As for scenario (2), trajectory pair τ_a and τ_b can be safely pruned based on Definition 3. ■

1) Spatio-textual similarity upper bound between two batches: Given batches b_i and b_j where $b_i, b_j \in IL(w)$, the spatio-textual similarity upper bound of trajectory pair (τ_a, τ_b), where $\tau_a \in b_i$ and $\tau_b \in b_j$, is computed by Equation 6.

$$\mathsf{ST}_B(b_i, b_j).ub = \alpha \cdot \mathsf{S}_{max}(b_i.mbr, b_j.mbr) \cdot 2 + (1-\alpha) \cdot 2, \quad (6)$$

where $\mathsf{S}_{max}(b_i.mbr, b_j.mbr)$ denotes the maximum spatial similarity between the minimum bounding rectangle (MBR) of trajectories in b_i and the MBR of trajectories in b_j. If $\mathsf{ST}_B(b_i, b_j).ub < \theta$, we filter out all trajectory pairs in b_i and b_j.

2) Spatio-textual similarity upper bound between a trajectory and a batch: Given trajectory τ and batch b, the spatio-textual similarity upper bound between τ and b is computed by Equation 7.

$$\mathsf{ST}_{TB}(\tau, b).ub = \frac{\sum_{o_i \in \tau} \mathsf{R}_{max}(o_i, b)}{|\tau|} + \alpha \cdot \mathsf{S}_{max}(\tau, b.mbr) + 1 - \alpha \quad (7)$$

$\mathsf{S}_{max}(\tau, b_j.mbr)$ denotes the maximum spatial score between τ and the MBR of b, $\mathsf{R}_{max}(o_i, b)$ denotes the maximum relevance between o_i and trajectories in b, which is computed by Equation 8.

$$\mathsf{R}_{max}(o_i, b) = \alpha \cdot \mathsf{S}_{max}(o_i.\rho, b.mbr) + (1-\alpha) \cdot \mathsf{T}_{max}(o_i.\psi, b), \quad (8)$$

where $\mathsf{T}_{max}(o_i.\psi, b)$ can be easily acquired by traversing the text summary of b. If $\mathsf{ST}_{TB}(\tau, b).ub < \theta$, we filter out all pairs between τ and trajectories in b.

3) BFJoin Algorithm: Our join algorithm, BFJoin, can be performed in parallel. Based on Theorem 2, we only need to run trajectory join algorithm within each inverted list. Consequently, the join algorithms regarding different inverted lists are independent of each other.

Algorithm 5 presents the pseudo code of out join algorithm regarding inverted list $IL(w_x)$. The inputs are $IL(w_x)$ (including its MBR and text summary) and the similarity threshold θ. The output is a set of similarity trajectory pairs within $IL(w_x)$, which is denoted by $Result$. We start our intra-list batch filtering phase (Lines 4–16). Specifically, we compute the spatio-textual similarity upper bound between batches $b(x, i)$ and $b(y, i)$ based on Equation 6. If the upper bound is lower than θ, we filter out batch pair $b(x, i)$ and $b(y, i)$. If the pair cannot be filtered, we proceed with the trajectory-batch filtering phase (Lines 6–16). To improve the pruning power, we compute the spatio-textual similarity upper bound between the batch that has smaller MBR (i.e., b_2) and each τ in the batch that has larger MBR (i.e., b_1) based on Equation 7. If the upper bound is not lower than θ, we proceed

Algorithm 5: BFJoin

```
   Data: Inverted list of term $w_x$, Similarity threshold $\theta$
   Result: Join result of trajectories in $IL(w_x)$
1  $Result \leftarrow \emptyset$;
2  Initialize $b_1$, $b_2$;
3  for each batch $b(x, i)$ do
4      for each batch $b(y, i)$ do
5          if batch pair $b(x, i)$ and $b(y, i)$ cannot be filtered
           then
6              if $b(x, i).mbr$ is larger than $b(y, i).mbr$ then
7                  $b_1 \leftarrow b(x, i)$; $b_2 \leftarrow b(y, i)$;
8              else
9                  $b_1 \leftarrow b(y, i)$; $b_2 \leftarrow b(x, i)$;
10             for each $\tau \in b_1$ do
11                 if trajectory-batch pair $\tau$ and $b_2$ cannot be
                   filtered then
12                     for each $\tau_j \in b_2$ do
13                         if $\mathsf{ST}(\tau, \tau_j).ub \geq \theta$ then
14                             Compute exact value of
                               $\mathsf{ST}(\tau, \tau_j)$;
15                             if $\mathsf{ST}(\tau, \tau_j) \geq \theta$ then
16                                 $Result$.add($\tau, \tau_j$);
17 return $Result$;
```

with our individual evaluation phase, which evaluates each pair of trajectories in b_1 and b_2 (Lines 12–16). In particular, we compute the similarity upper bound of a trajectory pair by using the trajectory pruning technique (cf. Section IV-C). If the upper bound is not lower than θ, we need to calculate the exact value of the trajectory pair similarity (Line 14). If the similarity is not lower than θ, we add the pair to $Result$ (Lines 15–16).

4) Trajectory grouping algorithm: To improve the effectiveness of intra-list batch filtering and trajectory-batch filtering, we need to compute a more accurate value of maximum spatial score. Note that the maximum spatial score is equivalent to minimum distance. For this purpose, it is useful to maintain the "quality" of each batch b in order to reduce the discrepancy between the minimum distance from object o to $b.mbr$ (denoted by $minDist(o.\rho, b.mbr)$) and the exact distance from o to o' (denoted by $dist(o.\rho, o'.\rho)$) where $o'.rho \in b.mbr$. According to triangle inequity theorem, it is easy to deduce that the maximum possible discrepancy between $minDist(o.\rho, b.mbr)$ and $dist(o.\rho, o'.\rho)$ is the diagonal length of $b.mbr$, which is denoted by $\mathsf{dg}(b.mbr)$.

To increase the accuracy of $minDist(o.\rho, b.mbr)$, we constrain the value of $\mathsf{dg}(b.mbr)$ for each batch. We develop a Diagonal-Constrained Online Grouping (DCOG) algorithm for grouping trajectories in an inverted list such that the MBR of trajectories in each batch b satisfies $\mathsf{dg}(b.mbr) \leq \theta_{diag}$, where θ_{diag} is the threshold of MBR diagonal length.

Algorithm 6 presents the pseudo code of the DCOG algorithm. It takes the original inverted list $IL(w)$ and the threshold of MBR diagonal length θ_{diag} as input. For each trajectory $\tau \in IL(w)$, we perform online grouping. Specifically, we assign τ to the batch with MBR that has the shortest diagonal length. At first, we initialize b_{min} and $diag_{min}$, which

Algorithm 6: DCOG

```
Input: Original inverted list $IL(w)$, Threshold of MBR
       diagonal length $\theta_{diag}$
Output: Batch set $B$
1  $B \leftarrow \emptyset$;
2  for each $\tau \in IL(w)$ do
3      Initialize $b_{min}$;
4      $diag_{min} \leftarrow$ dist_max;
5      for each batch $b$ in $B$ do
6          $b' \leftarrow b$.add($\tau$);
7          if $\mathsf{dg}(b'.mbr) < diag_{min}$ then
8              $diag_{min} \leftarrow \mathsf{dg}(b'.mbr)$;
9              $b_{min} \leftarrow b$;
10     if $diag_{min} \leq \theta_{diag}$ then
11         $b_{min}$.add($\tau$);
12         Update $b_{min}$ in $B$;
13     else
14         $b_n \leftarrow \{\tau\}$;
15         $B$.add($b_n$);
16 return $B$;
```

denote the current batch with the shortest diagonal length and the corresponding diagonal length, respectively (Lines 3–4). Next, for each batch b in batch set B we add τ to b, which is denoted by b' (Line 6). We find the batch b_{min} that has the minimum MBR diagonal length (Lines 7–9). If the corresponding diagonal length does not exceed the threshold θ_{diag}, we assign τ to b_{min} and update B (Lines 10–112; otherwise, we create a new batch b_n with the single trajectory τ and add b_n to B (Lines 13–15). Finally, we return B as the grouping result (Line 16).

E. Extension of Non-self Join

We proceed to explain how to extend our proposal to support non-self join, where we have two semantic trajectory sets T and Q ($T \neq Q$). We can extend the BFJoin algorithm to support the case where $T \neq Q$ in a straightforward manner by maintaining inverted file for trajectories in T and Q separately.

Let $IL_T(w)$ and $IL_Q(w)$ be inverted lists of term w for T and Q, respectively. We present our extension of BFJoin algorithm that supports non-self join (Algorithm 7).

Algorithm 7: BFJoin (non-self)

```
Data: Inverted lists $IL_T(w)$ and $IL_Q(w)$, Similarity
      threshold $\theta$
Result: Join result of trajectories in $IL_T(w)$ and $IL_Q(w)$
1  $Result \leftarrow \emptyset$;
2  for each batch $b_T \in IL_T(w)$ do
3      for each batch $b_Q \in IL_Q(w)$ do
4          Run Lines 5–16 in Algorithm 5;
5  return $Result$;
```

V. Experimental Study

A. Experiment Settings

1) Datasets: We use two datasets: New York trajectory data (NTD)[12] and Beijing trajectory data (BTD). The NTD consists of a road network and 10M taxi trips. Each taxi trip is a source-destination pair. We regard the shortest path from the source to the destination as the trajectory of a trip. In addition, we use a real POI data set that contains 19,969 POIs in New York City[13]. Each POI has a spatial coordinate with latitude and longitude and a text description. Because the POIs may not match the trajectory points, we map each POI to its nearest node in road network and regard the POI as an object in a semantic trajectory. In BTD, we use a real taxi trajectory data set collected by the T-drive project [42]. The original trajectories in BTD are very long since each of them contains all trips during a particular time period, which can be a couple of days. We divide these trajectories into half-an-hour sub-trajectories. The intent is to create trips with a realistic length and duration. To augment each trajectory point with text description, we randomly select a tweet from a collection of real-life tweets that comprises 2 million tweets and associate the text description of the tweet with the trajectory point.

2) Implementations: In the experiments, semantic trajectories and indexing structures are memory resident. 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. We use CPU time and the number of visited trajectories are our metrics of performance evaluation. 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., $T \neq Q$, and self joins, i.e., $T = Q$, in Sections V-B and V-C, respectively. Semantic trajectories in T and Q are selected randomly from the datasets.

We evaluate the following three methods:

- STIF: Search with trajectory inverted file (Section III);
- STM: Semantic trajectory summarization with trajectory pair pruning (Sections IV-B and IV-C);
- BFJoin: BFJoin algorithm with trajectory grouping and batch filtering (Sections IV-D).

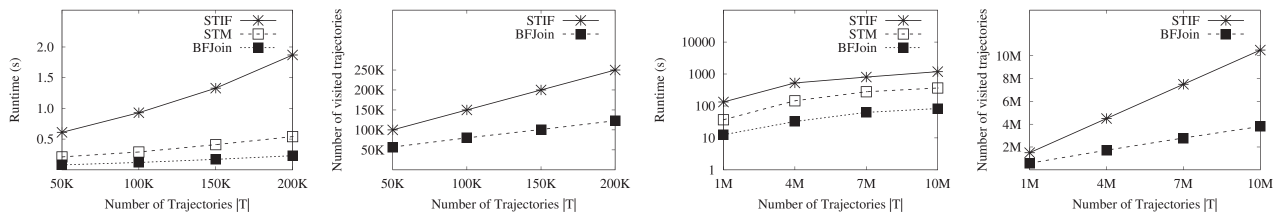
When evaluating the number of trajectory visits, we do not report the performance of STM because STM and STIF incur the same numbers of trajectory visits. The parameter settings are listed in Table I.

B. Experiment Results of Non-self Join

1) Effect of the number of trajectories: Figures 8 and 9 show the effect of semantic trajectory cardinalities $|T|$ and $|Q|$, respectively, on the performance of the three algorithms. Intuitively, a larger $|T|$ (or $|Q|$) causes more trajectory pairs to be evaluated, which denotes that both CPU time and the number of visited trajectories are expected to be higher for

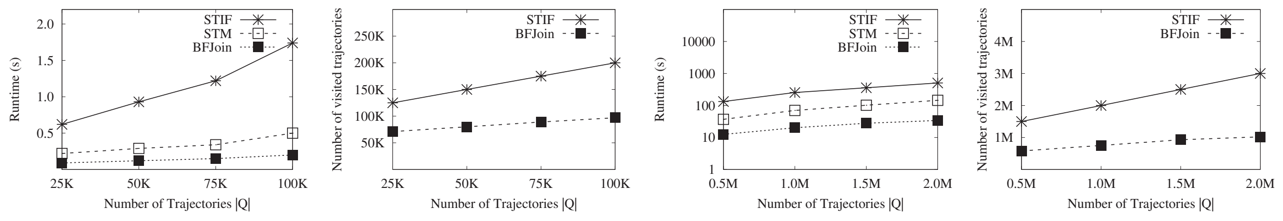
[12]https://publish.illinois.edu/dbwork/open-data/

[13]https://catalog.data.gov/dataset/points-of-interest-4aea0



(a) BTD (b) BTD (c) NTD (d) NTD

Fig. 8. Effect of the number of trajectories $|T|$



(a) BTD (b) BTD (c) NTD (d) NTD

Fig. 9. Effect of the number of trajectories $|Q|$

TABLE I: PARAMETER SETTINGS

	NTD	**BTD**
Cardinality of trajectories $|T|$	1,000,000–10,000,000 / default 1,000,000	50,000–200,000 / default 100,000
Cardinality of trajectories $|Q|$	500,000–2,000,000 / default 500,000	25,000–100,000 / default 50,000
Similarity threshold θ	1.8–1.95 / default 1.9	1.8–1.95 / default 1.9
Preference parameter α	0.3–0.9 / default 0.5	0.3–0.9 / default 0.5
MBR diagonal threshold θ_{diag}	5–20 / default 10	5–20 / default 10
Side length of grid cell	0.5–2 / default: 1	0.5–2 / default: 1
Thread count	24–120 / default 24	24–120 / default 24

all algorithms. We see that STM performs substantially better than STIF, which underscores the efficacy of our trajectory pair pruning scheme (cf. Sections IV-B and IV-C). In addition, our BFJoin algorithm further improves the CPU time by a factor of 1.8–2.2 (cf. Figures 8(a), 8(c), 9(a), and 9(c)), showing that a substantial proportion of trajectory pairs are filtered out during the intra-list batch filtering and trajectory-batch filtering phases. From Figure 8(c), we see that the BFJoin is able to process 1 million semantic trajectories ($|T|$ = 1 M and $|Q|$ = 0.5 M) in 12.5 seconds and 10 million semantic trajectories ($|T|$ = 10 M and $|Q|$ = 0.5 M) in 85 seconds on default 24 threads. In contrast, the STIF algorithm takes 133 and 1,190 seconds in processing 1 million and 10 million trajectories, respectively.

2) Effect of similarity threshold θ: This set of experiments investigates the effect of similarity threshold θ. Figure 10 shows the results when we vary θ. We see that STM and BFJoin perform better when we increase the value of θ. In contrast, the performance of STIF remains constant as we vary θ. The reasons can be explained as follow. For STM and BFJoin, a higher value of θ means that more trajectory pairs may be pruned by our trajectory pair pruning scheme. As a consequence, both CPU time and the number of visited trajectories may decrease. In particular, we find that BFJoin is more sensitive to θ in comparison to STM (Figure 10(c)). This can be explained by the fact that a higher value of θ may increase the pruning power of both intra-list batch filtering and trajectory-batch filtering as more trajectory batches may be filtered out when we increase θ.

3) Effect of preference parameter α: Figure 11 shows the effect of varying the preference parameter α, which balances the importance of spatial similarity versus text similarity. The importance of the spatial similarity increases as we vary α from 0.3 to 1.0. In particular, when $\alpha = 1.0$, the similarity between two trajectories is solely measured by the spatial proximity. We see that the value of α has little influence on the performances of STIF and STM. STIF has no pruning scheme, and we need to compute the exact similarity of each pair of semantic trajectories in an inverted list. Thus, its performance is independent of α. Next STM has a trajectory pair pruning scheme, which considers both the spatial and textual aspects. The observed performance of STM suggests that our trajectory pair pruning scheme is robust to imbalanced weights between spatial and text similarity. In contrast, BFJoin is moderately sensitive to α. The reason is that the intra-batch filtering technique only considers the spatial aspect. Putting less weight on spatial similarity (i.e., decreasing α) may lower the power of intra-batch filtering. Nevertheless, our trajectory-batch filtering technique, followed with intra-batch filtering, takes both spatial and textual aspects into account. As a result, the performance of BFJoin is still moderately better than STM even if we set α to 0.3.

4) Effect of the side length of grid cell: We proceed to evaluate the effect of varying the grid granularity in trajectory

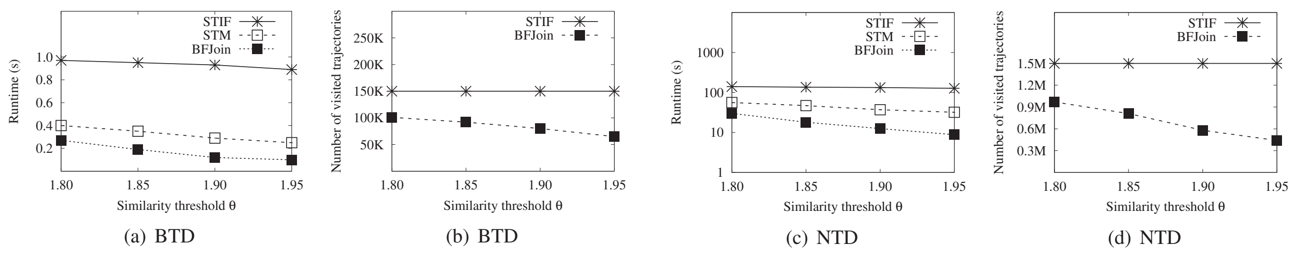


Fig. 10. Effect of spatio-textual similarity threshold θ

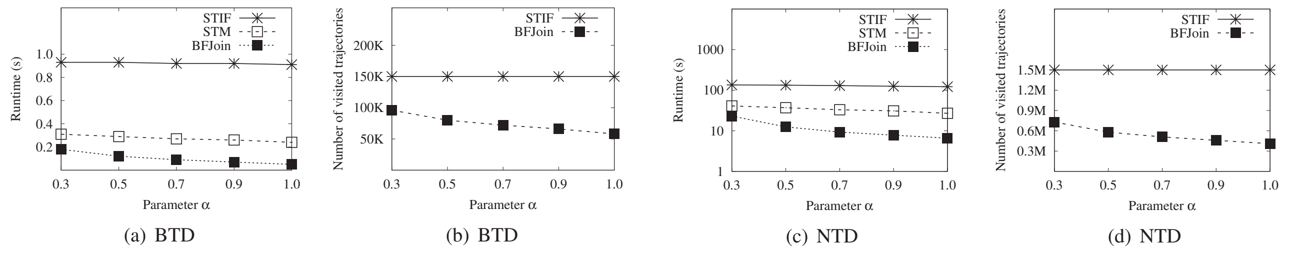


Fig. 11. Effect of preference parameter α

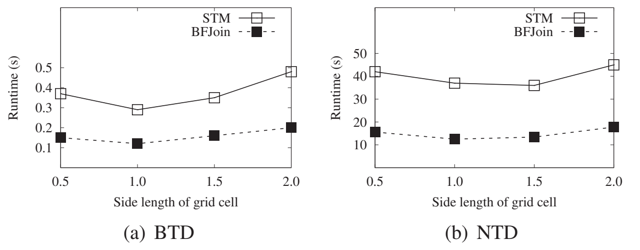


Fig. 12. Effect of the side length of grid cell

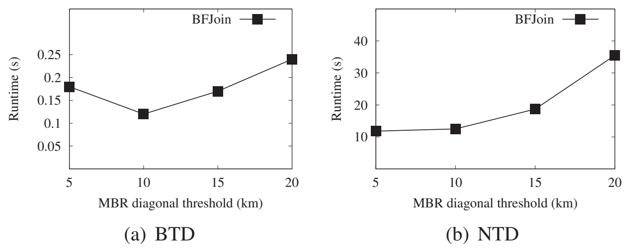


Fig. 13. Effect of MBR diagonal length threshold θ_{diag}

pair pruning technique. From Figure 12, we find that both STM and BFJoin exhibit the best performance when the side length of grid cell is set between 1 to 1.5. A finer granularity can improve the accuracy of the spatial similarity upper bound between two trajectories (i.e., $\mathsf{ST}_S(\tau_i, \tau_j).ub$, see Section IV-C). However, the time cost of computing the upper bound may increase accordingly. On the contrary, a coarser granularity may improve the efficiency of upper bound computation at the cost of lowering the accuracy.

5) Effect of MBR diagonal length threshold θ_{diag}:

TABLE IV: AVERAGE MBR SIZE IN PROPORTION TO THE TOTAL MAP SIZE

MBR diagonal length (km) / Dataset	5	10	15	20
BTD	0.46%	1.5%	3.8%	6.1%
NTD	0.22%	0.83%	1.7%	2.8%

Figure 13 shows the effect of varying the MBR diagonal length threshold, θ_{diag}, which is used in Algorithm 6. Table IV reports the average MBR size in proportion to the MBR size of the entire road network when we vary θ_{diag} from 5 km to 20 km. When θ_{diag} is low, we may have a large number of "small" batches associated with each inverted list. If we increase θ_{diag}, we will have fewer trajectory batches. However, the average MBR diagonal length of each group may increase, which lowers the accuracy of the similarity upper bound between two batches (Equation 6) and the similarity upper bound between a trajectory and a batch (Equation 7).

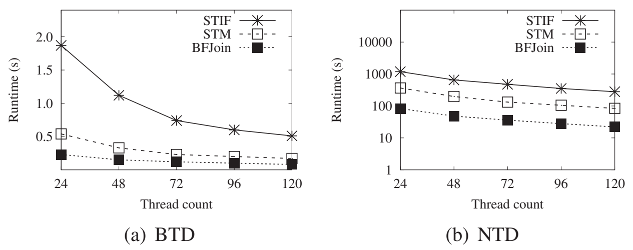


Fig. 14. Effect of thread counts

6) Effect of thread counts: We study the effect of thread count on the efficiency of the algorithms using large trajectory data sets ($|T|$ = 200 K and $|Q|$ = 50 K for BTD, and $|T|$ = 10 M and $|Q|$ = 500 K for NTD). Figure 14 show that BFJoin consistently outperforms STIF by a factor of 8–12 and outperforms STM by 50%–75% regarding CPU time. In BTD, when we set the thread counts to 120, BFJoin is able to solve the STS-Join problem over a collection of 200 K trajectories in 0.1 second, while in NTD, BFJoin is able to solve the STS-Join problem with 10M trajectories in 25 seconds.

C. Experiment Results of Self Join

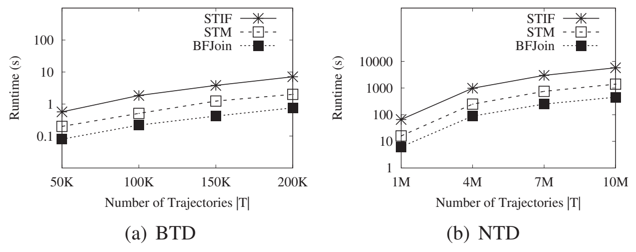


Fig. 15. Effect of the number of semantic trajectories (self-join)

Figure 15 shows the runtime for the self joins when varying the trajectory cardinality. Note that the thread counts are set to 24 and 120 on BTD and NTD, respectively. The trends are similar to those of the non-self join. The BFJoin outperforms STIF by a factor of $\sim$8 and by a factor of $\sim$11 on BTD and NTD, respectively.

VI. Related Work

Spatial keyword trajectory query processing: A spatial keyword trajectory query contains a set of query locations and a set of keywords. Given a collection of semantic trajectories, the query finds a subset of trajectories that are spatio-textually similar to the query. The problem of efficient processing of top-k spatial keyword trajectory query is studied by Cong et al. [13] and Zheng et al. [47]–[49] and Shang et al. [34]. Given a set T of semantic trajectories and a query q that includes a set of query locations and a set of keywords, a top-k spatial keyword trajectory query (TkSK) finds k semantic trajectories from T that have the shortest minimum match distances to q. In particular, the minimum match distance considers (1) the distance between query locations and a trajectory and (2) the number of POIs in the trajectory that cover the query keywords. Besides, Han et al. [17] focus on processing spatial keyword range trajectory query over semantic trajectory data. The query consists of query ranges and a set of keywords. It finds all trajectories that locate within the query ranges and collectively contain query keywords.

These proposals cannot process the STS-Join due to three reasons. (1) Different problems and matching schemes (query vs. join): The spatial keyword trajectory query contains a set of locations and a set of keywords, which means that the text information is not associated with the locations. In contrast, each object in a semantic trajectory contains both location text information, indicating that they are associated with each other. As a result, the query-trajectory matching scheme is totally different our trajectory-trajectory matching scheme. (2) Parallel processing: none of the above proposals are developed for parallel processing. Hence, they are incapable of handling very large semantic trajectory data sets.

Spatial keyword search and join: Given two collections of geo-textual objects T and Q, the problem of spatial keyword join is to find all pairs of objects respectively from R and S that are both spatially close and textually similar. Liu et al. [16], [20] and Hu et al. [19] develop a filter-and-refine framework by utilizing spatial and textual signatures. Bouros et al. [3] propose a batch processing mechanism that dynamically partitions the geo-textual objects into groups according to their spatial and textual information. Rao et al. [28] develop two spatial-first and two text-first indexing schemes for efficient spatial-textual joint search. Either grid or Quad-tree is used for spatial indexing part. Zhang et al. [45] develop solutions for spatio-textual join under MapReduce framework by reducing the duplicates of data and the workload of the reducers. However, these proposals focus on object-object matching, which is different from our problem of trajectory-trajectory matching. Additionally, spatial keyword search is extensively investigated by existing studies [4], [7], [23], [24]. Given a collection of static or a stream of dynamic geo-textual objects, the problem is to find a subset of geo-textual objects (e.g., [25], [26], [41]) or summarized results (e.g., [8], [9], [37], [46]) based on a query with both spatial and textual requirements.

Traditional trajectory search and join: Trajectory join without text information is widely studied in literature [1], [2], [6], [11], [14], [29]–[33], [35], [36], [38]. The applications of trajectory similarity joins include trajectory near-duplicate detection, data cleaning, ridesharing recommendation, and traffic congestion prediction. Developing such joins 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. Second, an efficient algorithm is developed to retrieve all trajectory pairs whose similarity satisfies a pre-defined threshold. However, these studies do not take text domain into consideration when calculating the similarity between two trajectories.

VII. Conclusions

We studied a novel semantic trajectory similarity join (STS-Join), which targets a number of useful applications such as term-based trajectory duplicate detection, geo-text data cleaning, personalized ridesharing/carpooling recommendation, and itinerary recommendation. To process the STS-Join efficiently, we developed a number of trajectory pair filtering techniques that took into account the parallel processing capabilities of modern processors. We also proposed an efficient divide-and-conquer algorithm for computing an upper bound of spatio-textual similarity between two semantic trajectories. The performance of the STS-Join was investigated through extensive experiments on large trajectory data. Experimental result has shown that the pruning power of our filtering techniques were strong enough and our join algorithm was capable of achieving high efficiency and scalability on massive-scale trajectory data.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (61932004, 61922054, 61872235, 61729202, 61832017, U1636210) and the National Key Research and Development Program of China (2018YFC1504504, 2016YFB0700502).

REFERENCES

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

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

[3] P. Bouros, S. Ge, and N. Mamoulis. Spatio-textual similarity joins. *PVLDB*, 6(1):1–12, 2012.

[4] X. Cao, L. Chen, G. Cong, C. S. Jensen, Q. Qu, A. Skovsgaard, D. Wu, and M. L. Yiu. Spatial keyword querying. In *ER*, volume 7532, pages 16–29. Springer, 2012.

[5] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: An experimental evaluation. *PVLDB*, 6(3):217–228, 2013.

[6] L. Chen, S. Shang, C. S. Jensen, B. Yao, Z. Zhang, and L. Shao. Effective and efficient reuse of past travel behavior for route recommendation. In *KDD*, pages 488–498.

[7] L. Chen, S. Shang, C. Yang, and J. Li. Spatial keyword search: a survey. *GeoInformatica*, 24(1):85–106, 2020.

[8] L. Chen, S. Shang, Z. Zhang, X. Cao, C. S. Jensen, and P. Kalnis. Location-aware top-k term publish/subscribe. In *ICDE*, pages 749–760.

[9] L. Chen, S. Shang, K. Zheng, and P. Kalnis. Cluster-based subscription matching for geo-textual data streams. In *ICDE*, pages 890–901.

[10] L. Chen, J. Xu, X. Lin, C. S. Jensen, and H. Hu. Answering why-not spatial keyword top-k queries via keyword adaption. In *ICDE*, pages 697–708, 2016.

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

[12] 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.

[13] G. Cong, H. Lu, B. C. Ooi, D. Zhang, and M. Zhang. Efficient spatial keyword search in trajectory databases. *arXiv:1205.2880*, pages 1–12, 2012.

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

[15] C. Doulkeridis, A. Vlachou, D. Mpestas, and N. Mamoulis. Parallel and distributed processing of spatial preference queries using keywords. In *EDBT*, pages 318–329, 2017.

[16] J. Fan, G. Li, L. Zhou, S. Chen, and J. Hu. SEAL: spatio-textual similarity search. *PVLDB*, 5(9):824–835, 2012.

[17] Y. Han, L. Wang, Y. Zhang, W. Zhang, and X. Lin. Spatial keyword range search on trajectories. In *DASFAA*, pages 223–240, 2015.

[18] H. Hu, Y. Liu, G. Li, J. Feng, and K. Tan. A location-aware publish/subscribe framework for parameterized spatio-textual subscriptions. In *ICDE*, pages 711–722, 2015.

[19] H. Huiqi, L. Guoliang, B. Zhifeng, F. Jianhua, W. Yongwei, G. Zhiguo, and X. Yaoqing. Top-k spatio-textual similarity join. *IEEE Trans. Knowl. Data Eng.*, 28(2):551–565, 2016.

[20] S. Liu, G. Li, and J. Feng. Star-join: spatio-textual similarity join. In *CIKM*, pages 2194–2198, 2012.

[21] S. Liu, G. Li, and J. Feng. A prefix-filter based method for spatio-textual similarity join. *IEEE Trans. Knowl. Data Eng.*, 26(10):2354–2367, 2014.

[22] Y. Lu, J. Lu, G. Cong, W. Wu, and C. Shahabi. Efficient algorithms and cost models for reverse spatial-keyword *k*-nearest neighbor search. *ACM Trans. Database Syst.*, 39(2):13:1–13:46, 2014.

[23] A. Magdy, L. Abdelhafeez, Y. Kang, E. Ong, and M. F. Mokbel. Microblogs data management: a survey. *VLDB J.*, 29(1):177–216, 2020.

[24] A. R. Mahmood and W. G. Aref. *Scalable Processing of Spatial-Keyword Queries*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2019.

[25] A. R. Mahmood, W. G. Aref, A. M. Aly, and M. Tang. Atlas: on the expression of spatial-keyword group queries using extended relational constructs. In *SIGSPATIAL*, pages 45:1–45:10.

[26] K. Mouratidis, J. Li, Y. Tang, and N. Mamoulis. Joint search by social and spatial proximity. *IEEE Trans. Knowl. Data Eng.*, 27(3):781–793, 2015.

[27] C. Parent, S. Spaccapietra, C. Renso, G. L. Andrienko, N. V. Andrienko, V. Bogorny, M. L. Damiani, A. Gkoulalas-Divanis, J. A. F. de Macêdo, N. Pelekis, Y. Theodoridis, and Z. Yan. Semantic trajectories modeling and analysis. *ACM Comput. Surv.*, 45(4):42:1–42:32, 2013.

[28] J. Rao, J. Lin, and H. Samet. Partitioning strategies for spatio-textual similarity join. In *BigSpatial@SIGSPATIAL*, pages 40–49, 2014.

[29] S. Ray, A. D. Brown, N. Koudas, R. Blanco, and A. K. Goel. Parallel in-memory trajectory-based spatiotemporal topological join. In *Big Data*, pages 361–370. IEEE, 2015.

[30] S. Shang, L. Chen, C. S. Jensen, J. Wen, and P. Kalnis. Searching trajectories by regions of interest. *IEEE Trans. Knowl. Data Eng.*, 29(7):1549–1562, 2017.

[31] S. Shang, L. Chen, Z. Wei, C. S. Jensen, K. Zheng, and P. Kalnis. Trajectory similarity join in spatial networks. *PVLDB*, 10(11):1178–1189, 2017.

[32] S. Shang, L. Chen, Z. Wei, C. S. Jensen, K. Zheng, and P. Kalnis. Parallel trajectory similarity joins in spatial networks. *VLDB J.*, 27(3):395–420, 2018.

[33] S. Shang, L. Chen, K. Zheng, C. S. Jensen, Z. Wei, and P. Kalnis. Parallel trajectory-to-location join. *IEEE Trans. Knowl. Data Eng.*, 31(6):1194–1207, 2019.

[34] 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.

[35] 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.

[36] Z. Shang, G. Li, and Z. Bao. DITA: distributed in-memory trajectory analytics. In *SIGMOD*, pages 725–740, 2018.

[37] A. Skovsgaard, D. Sidlauskas, and C. S. Jensen. Scalable top-k spatio-temporal term querying. In *ICDE*, pages 148–159.

[38] N. Ta, G. Li, Y. Xie, C. Li, S. Hao, and J. Feng. Signature-based trajectory similarity join. *IEEE Trans. Knowl. Data Eng.*, 29(4):870–883, 2017.

[39] P. Tampakis, C. Doulkeridis, N. Pelekis, and Y. Theodoridis. Distributed subtrajectory join on massive datasets. *CoRR*, abs/1903.07748, 2019.

[40] X. Wang, W. Zhang, Y. Zhang, X. Lin, and Z. Huang. Top-k spatial-keyword publish/subscribe over sliding window. *VLDB J.*, 26(3):301–326, 2017.

[41] C. Yang, L. Chen, S. Shang, F. Zhu, L. Liu, and L. Shao. Toward efficient navigation of massive-scale geo-textual streams. In *IJCAI*, pages 4838–4845.

[42] J. Yuan, Y. Zheng, X. Xie, and G. Sun. T-drive: Enhancing driving directions with taxi drivers' intelligence. *IEEE Trans. Knowl. Data Eng.*, 25(1):220–232, 2013.

[43] C. Zhang, Y. Zhang, W. Zhang, and X. Lin. Inverted linear quadtree: Efficient top k spatial keyword search. In *ICDE*, pages 901–912, 2013.

[44] D. Zhang, K. Tan, and A. K. H. Tung. Scalable top-k spatial keyword search. In *EDBT*, pages 359–370, 2013.

[45] Y. Zhang, Y. Ma, and X. Meng. Efficient spatio-textual similarity join using mapreduce. In *WI-IAT*, pages 52–59, 2014.

[46] K. Zhao, L. Chen, and G. Cong. Topic exploration in spatio-temporal document collections. In *SIGMOD*, pages 985–998.

[47] B. Zheng, N. J. Yuan, K. Zheng, X. Xie, S. W. Sadiq, and X. Zhou. Approximate keyword search in semantic trajectory database. In *ICDE*, pages 975–986, 2015.

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

[49] K. Zheng, B. Zheng, J. Xu, G. Liu, A. Liu, and Z. Li. Popularity-aware spatial keyword search on activity trajectories. *World Wide Web*, 20(4):749–773, 2017.