Answering Top-k Exemplar Trajectory Queries

Sheng Wang* Zhifeng Bao* J. Shane Culpepper* Timos Sellis† Mark Sanderson* Xiaolin Qin‡
*RMIT University, †Swinburne University of Technology, ‡Nanjing University of Aeronautics and Astronautics
*firstname.surname@rmit.edu.au †tsellis@swin.edu.au ‡qinxcs@nuaa.edu.cn

***Abstract*—We study a new type of spatial-textual trajectory search: the *Exemplar Trajectory Query (ETQ)*, which specifies one or more places to visit, and descriptions of activities at each place. Our goal is to efficiently find the top-k trajectories by computing spatial and textual similarity at each point. The computational cost for pointwise matching is significantly higher than previous approaches. Therefore, we introduce an incremental pruning baseline and explore how to adaptively tune our approach, introducing a gap-based optimization and a novel two-level threshold algorithm to improve efficiency. Our proposed methods support order-sensitive ETQ with a minor extension. Experiments on two datasets verify the efficiency and scalability of our proposed solution.**

I. Introduction

Large amounts of *trajectory* data, containing geographic information of moving objects, is being recorded with the proliferation of GPS-enabled devices. People commonly "check-in" with smart phones to record personal historical movement, see Figure 1. Composed of locations, timestamps, and keywords, the check-ins form a unique trajectory recorded, which can be later mined for applications such as trip recommendation [9].

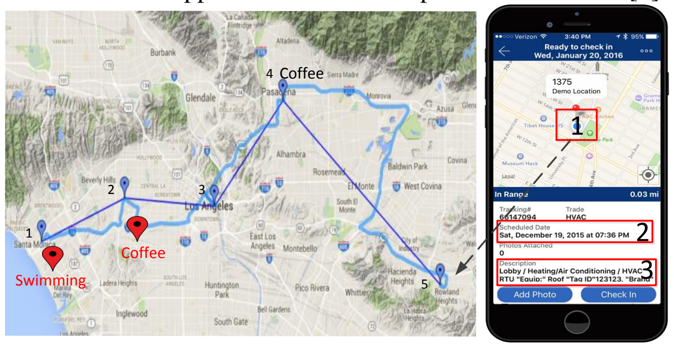

Fig. 1. Check-in points hold 1) a location, 2) a timestamp, and 3) keywords.

Example 1: Imagine that Jack is planning a trip to Los Angeles. He initially identifies two places to stay and visit: the hotel he booked and the beach. He specifies things to do around each place, for example "coffee" and "swimming" (red points in Figure 1) respectively. **Query:** Find past trajectories most related to Jack given his suggested locations and interests.

Commercial trip recommendation systems such as *Google Trips*[1] and *Triphobo*[2] are built for tourists to plan and share trips online. However, the input to such systems is usually a simple query such as "*Los Angeles*". If a user (such as Jack) has a more nuanced preference of locations and keywords, they cannot easily be expressed using existing systems.

An Exemplar Textual Query (ETQ) is defined as: *Given a database of trajectories, and a query Q that includes a tuple of one or more locations and associated keywords (with attached weights), retrieve the top-k trajectories ranked by spatial-textual similarity.* Consider the example shown in Figure 2, where the query consists of three points, each with an attached keyword, 2(c). Six potential result trajectories are listed, 2(a). Each trajectory also has a set of keywords with associated weights (only the keywords of T_1 are shown 2(b)). A Top-1 ETQ finds the most related trajectory from the six trajectories.

[1]https://www.google.com/trips/
[2]https://www.triphobo.com/

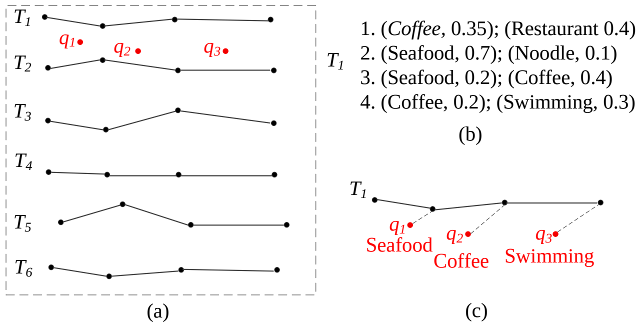
T_1
T_2
T_3
T_4
T_5
T_6
q_1
q_2
q_3
(a)
T_1
1. (Coffee, 0.35); (Restaurant 0.4)
2. (Seafood, 0.7); (Noodle, 0.1)
3. (Seafood, 0.2); (Coffee, 0.4)
4. (Coffee, 0.2); (Swimming, 0.3)
(b)
T_1
q_1 Seafood
q_2 Coffee
q_3 Swimming
(c)

Fig. 2. (a): An example of *ETQ* (6 trajectories and the exemplar query Q); (b): Keywords with a weight of T_1 in each point; (c) Point-wise matching between Q and T_1

At first glance, our problem can be scoped as a spatial-textual trajectory search [5, 7, 14, 19, 20]. However, our input query differs from past work. Shang et al. [14] modeled queries as distinct sets of locations and keywords. Consequently, retrieved trajectories were close to target locations, but contained no common keyword with each query point since the keywords were not necessarily constrained to any of the locations. In contrast, Zheng et al. [20] used Boolean queries, which only retrieved exact matches. Neither of these approaches satisfy the query we wish to ask. For example, in the work of Shang et al. [14], the trajectory in Figure 1 which contains "coffee" in the 4^{th} point that is faraway from the query point may be recognized as a quite similar trajectory as the 2^{nd} point is spatially close to the query point, and for Zheng et al. [20], a trajectory without the keyword "coffee" would not be a result even if all of the other keywords are matched.

We extend existing work on pointwise similarity [3, 11, 12, 15, 16, 17], so that every query point contains the best match with the highest score in the whole trajectory, and the aggregation of the scores for all query points, which reflects the closeness between trajectory *and* query, best represents the overall similarity. For instance, recall in Figure 2(c) where each point in Q finds the best-matching point in T_1, so the second point of T_1 has the highest similarity with q_1.

Efficient computation of such top-k trajectory queries is our key challenge. Trajectory pruning is more difficult when text descriptions are constrained to point locations. If traditional approaches are used to solve this problem, the pointwise similarity computation has a quadratic cost, since every point

TABLE I. Comparison between related work on spatial-textual trajectory search, where Eu stands for "Euclidean", Jac stands for "Jaccard" and ED stands for "Edit Distance".

Work	Input		Search Semantic	Search Metric		Output
	Query	Data Model		Spatial	Textual	
ETQ	Points with keywords	Trajectory	Disjunctive	Eu	TF·IDF	Top-k Trajectories
Zheng et al. [20]	Points with keywords	Trajectory	Conjunctive	Eu	N/A	Top-k Trajectory
Shang et al. [14]	Points, Keywords	Spatial-only Trajectory and Keywords	Disjunctive	Eu	Jac	Top-k Trajectories
Cong et al. [5]	Point, Keywords	Trajectory	Conjunctive	Eu	N/A	Top-k Sub-Trajectories
Zheng et al. [19]	Keywords	Trajectory	Conjunctive	Eu	ED	Top-k (Sub-)Trajectories
Han et al. [7]	Region, Keywords	Trajectory	Conjunctive	N/A	N/A	Trajectories

must be compared against all points in every other trajectory. Combining the two dimensions into a coherent definition of similarity while still allowing efficient pruning based on both spatial proximity and keyword similarity is the main problem addressed in this paper.

Contribution. In this paper, we define a top-k ETQ search, and introduce a fine-grained spatial-textual similarity function that starts with a point-level to trajectory-level match. We propose an incremental lookup framework to improve efficiency which works as follows: 1) Expand the search range λ to perform a top-λ spatial keyword query, and find the λ closest points and their affiliated trajectories, forming a candidate top-k set; 2) Expansion is halted once the upper bound for any unseen trajectory similarity is not greater than the lower bound of the currently processed trajectories in the candidate set, thus pruning away trajectories which cannot be in the top-k; 3) Sort the candidate set to find the top-k. The pruning methods in Step 2 form the baseline. Two additional directions to improve the efficiency are explored using this framework. Briefly, our contributions can be summarized as follows:

- We propose an adaptive gap-based optimization to terminate the search range expansion as early as possible. (Section 5)
- By observing that the same data points may be repeatedly probed until the candidate set is finalized during the incremental expansion, we propose a two-level threshold algorithm (TA) to avoid repetition. (Section 6)
- We show how to extend our framework to support order-sensitive exemplar search. (Section 7)
- We have performed extensive experiments using two datasets to verify the efficiency and scalability of our approach. (Section 8)

II. Related Work

In this section we review related work on spatial-textual trajectory search, spatial-only trajectory search, and spatial keyword search.

A. Spatial-Textual Trajectory Search

Several recent papers have explored the problem of spatial-textual trajectory search in a variety of different scenarios. Table I reports the differences and similarities between our work and prior work in terms of query and data models, search semantics, metrics and result output. From Table I, we can see that the only work which supports the same input and output as *ETQ* is the work of Zheng et al. [20]. However, their work only supports conjunctive matching, so all of the results must contain every query keyword, which simplifies the more general keyword search problem. Shang et al. [14] present a same search semantic, but keywords are attached to the whole trajectory, and not individual locations in the query or the data. Other work [5, 7, 19] is quite different than our work from the perspective of input and search semantics. Briefly, Cong et al. [5] proposed a sub-trajectory search method to find the minimum travel distance for a single query point using a Boolean keyword match. Zheng et al. [19] proposed an approximate query solution over trajectories with a set of keywords, and Han et al. [7] focused on range query search over trajectories. Our work focuses on ranked, disjunctive, pointwise search semantics which is distinct from all prior related work.

B. Spatial-only Trajectory Search by Points

As described in Section 1, there has been various work targeting search on spatial-only trajectory data using a set of points as the input query [3, 11, 12, 15, 16]. Chen et al. [3] initially formulated the problem of querying over a set of points with spatial-only trajectories. They proposed an incremental expansion method which used an R-tree for pruning, and referred to the search as *Incremental K Nearest Neighbors (IKNN)*. Tang et al. [15] proposed a qualifier expectation measure that ranks partial-matching candidate trajectories to accelerate query processing for non-uniform trajectory distributions and/or outlier query locations. Qi et al. [11, 12] went on to combine *IKNN* and the qualifier expectation model to improve the efficiency, and presented a range-based method to improve the efficiency even further. By extending a pointwise similarity model, Yan et al. [16] assigned an *importance value* to each point based on a time stamp to distinguish between points where a user spends more time within a trajectory.

C. Top-k Spatial Keyword Search

A top-k Spatial Keyword (*TkSK*) search retrieves the k objects with the highest ranking scores, measured as a combination of distance to the query location, and relevance of the text description to the query keywords. The solution can be broadly divided as two categories: spatial-first [4, 8] and text-first [13, 17]. A spatial-first strategy augments the nodes of a spatial data structure like an R-tree with a pseudo-document vector to store the textual information in each node. For example, the IR-tree [4, 8] was proposed to combine an R-tree with an inverted index. Spatial relevance is calculated from the MBR and an upper bound of textual relevance is derived from the pseudo-document for pruning. However, a spatial-first strategy will scan a few leaf nodes which contain many unrelated points that do not share any common keywords at all, so an IR-tree can be inefficient, especially when k is large [10]. For text-first methods such as [13], the search space is primarily organized by keywords, so only the points which share common keywords with the query will be checked, and the processing time does not increase as in the IR-tree, even though k is large. Recently, Zhang et al. [18] proposed an novel solution for top-k spatial keyword query which recasts the problem to the well-known top-k aggregation problem.

III. Problem Definition

In this section, we will formally define the problem of exemplar trajectory query. Table II summarizes the notation used throughout this paper.

TABLE II. SUMMARY OF NOTATION

Notation	Definition
T	Trajectory
Q	Exemplar query
$\hat{S}(Q,T)$	Trajectory similarity between Q and T (Definition 6)
p_i, p_j	Points p_i and p_j
$\hat{S}_T(p_i,p_j)$	Textual similarity between p_i and p_j
$\hat{S}_S(p_i,p_j)$	Spatial similarity between p_i and p_j
$\hat{S}(p_i,p_j)$	Spatial-textual similarity between p_i and p_j
unseen_UB	The upper bound across all unseen trajectories' similarity
$LB_{seen}(Q,T)$	A function returning the lower bound of the trajectory similarity between Q and T
$UB_{seen}(Q,T)$	A function returning the upper bound of trajectory similarity between Q and T
$\hat{S}_c(q_i)$	The minimum similarity of candidate points for q_i in the c-th round of expansion
$R_c(q_i)$	The ranked list for q_i in the c-th round of expansion
C_{tra}	The set of all checked trajectories
$BG(c)$	The gap between the lower and upper bound in the c-th round of expansion (Definition 8)
$Spa_c(q_i)$	The similarity sparsity of q_i in c-th expansion
it_{max}	The number of iterations to scan all points in candidate points

Definition 1: **(Point)** A point $p = (loc, act)$ is a pair consisting of a location *loc* and a set of associated keywords $act = (t_1, t_2, \ldots, t_i)$ describing the *loc* and/or the activities at *loc*.

Definition 2: **(Trajectory)** A trajectory T of length n is in the form of $p_1, p_2, \ldots, p_n$, where each p_i is a point.

Definition 3: **(Query)** A query Q (of size m) is a set of points in the form of $\{q_1,q_2,\ldots,q_m\}$.

The similarity between T and Q is computed between points which share at least one common keyword. While a query point may have multiple textwise matching points, recalling the related work on spatial-only trajectory search, similarity is computed from one point to another point. Therefore, we only choose the point with the maximum spatial-textual similarity, and add all point-to-point similarities to get the spatial-textual similarity between query and trajectories.

Definition 4: **(Point-to-Point Similarity)** We define the similarity between two points p_i, p_j as:

$$\hat{S}(p_i,p_j) = \begin{cases} 0, \ p_i.act \cap p_j.act = \varnothing \\ \alpha \cdot \hat{S}_S + (1-\alpha) \cdot \hat{S}_T, \text{otherwise} \end{cases} \quad (1)$$

where $\hat{S}_T(p_i,p_j)$ is the text similarity, $\hat{S}_S(p_i,p_j)$ is the spatial similarity between two points, and $\alpha \in (0,1)$ is used to adjust the relative importance of the spatial and textual similarity.

We use the sum of the textual relevance of each term [1, 18] to measure the textual similarity, and the Euclidean distance to measure the spatial similarity. The choice of similarity metric is orthogonal to our query processing method (in Sec. IV).

$$\hat{S}_T(p_i,p_j) = \sum_{t \in p_i.act \cap p_j.act} \gamma(t) \quad (2)$$

$$\hat{S}_S(p_i,p_j) = \frac{D_{max} - Euclidean(p_i,p_j)}{D_{max}} \quad (3)$$

Here, $\gamma(t)$ is the weight of keyword t in p_j calculated by a simple TF·IDF model [1]. The variable D_{max} is the maximum distance between any two unique points in geographical space, and used to normalize the spatial scoring between 0 and 1.

Definition 5: **(Point-to-Trajectory Similarity)** The similarity between a query point q_i and a trajectory T is defined as:

$$\hat{S}(q_i,T) = \max_{p_j \in T} \left\{ \hat{S}(q_i,p_j) \right\} \quad (4)$$

TABLE III. SIMILARITY TABLE BETWEEN Q AND ALL TRAJECTORIES T_1 TO T_6 SHOWN IN FIGURE 2 BASED ON DEFINITIONS 1-6 WHERE $\alpha = 0.5$. HERE, "ID" SHOWS THE POINT POSITION IN TRAJECTORY.

	q_1			q_2			q_3			Q
	ID	$\hat{S}_S$	$\hat{S}_T$	ID	$\hat{S}_S$	$\hat{S}_T$	ID	$\hat{S}_S$	$\hat{S}_T$	$\hat{S}$
T_1	2	0.7	0.7	3	0.4	0.5	4	0.3	0.5	0.516
T_2	2	0.6	0.5				4	0.5	0.5	0.35
T_3	1	0.9	0.5							0.233
T_4	1	0.2	0.3	2	0.7	0.5				0.283
T_5				2	0.7	0.3	3	0.5	0.3	0.3
T_6				3	0.4	0.2	4	0.2	0.2	0.166

Definition 6: **(Pointwise Similarity)** The pointwise similarity between T and Q is a sum of the point-trajectory similarities between T and each point in Q, normalized by $|Q|$:

$$\hat{S}(Q,T) = \sum_{q_i \in Q} \hat{S}(q_i,T) / |Q|. \quad (5)$$

In trajectory T, $|Q|$ points are chosen to compute the final similarity between T and Q. These $|Q|$ points form a sub-trajectory which can be taken as a representative result, and denoted as T_Q.

Definition 7: **(Top-k Exemplar Trajectory Query)** Given a trajectory database $D = \{T_1, \ldots, T_{|D|}\}$ and query Q, a trajectory search retrieves a set $R \subseteq D$ with k trajectories such that: $\forall r \in R, \forall r' \in D - R, \hat{S}(Q,r) > \hat{S}(Q,r')$.

Example 2: Figure 2 is an illustrative example of a query and trajectories, showing: (a) The spatial shapes of the query and trajectories; (b) The keywords attached to each point in T_1; and (c) The best match for each query point with T_1 based on our pointwise similarity model. Further, Table III presents an example of the similarity computations between a query Q and the six trajectories (shown in Figure 2). For each query point, we list the maximum similarity for every trajectory, and a blank space means that they share no common keywords. We can compute the similarity between Q and T_1 using $\hat{S}(Q,T_1) = \frac{\frac{0.7+0.7}{2} + \frac{0.4+0.5}{2} + \frac{0.3+0.5}{2}}{3} = 0.516$ and the similarities of other trajectories are listed in the right column of the table. As we can see, T_1, T_2, T_5 are the top-3 results.

IV. INCREMENTAL QUERY PROCESSING

The similarity (Definition 6) is an aggregation of spatial-textual similarities from all query points, and is inspired by spatial-only trajectory search [3, 11, 12, 15]. The threshold algorithm of Fagin et al. [6] can be used directly as a filtering framework for ranked lists. While in principle a similar idea can be modified to suit our purposes, using the algorithm of Fagin et al. directly does not work since the top-k list for every point in the query is not known a priori. However another solution, the incremental k nearest neighbor search algorithm *IKNN* [3, 11, 12] can be used to fill partially ranked lists with exactly λ nearest points for every query point. The ranked lists can be expanded by increasing λ until all unseen trajectories can not beat the current results. In this section, we show how to extend *IKNN* from spatial-only to spatial-textual search, and propose several bounds to terminate the expansion, which form a baseline processing framework for *ETQ*.

A. Incremental Lookup Algorithm

The Incremental Lookup Algorithm (**ILA**) can be divided into three steps, as shown in Algorithm 1.

Step 1: For each query point $q_i \in Q$, we first conduct a top-λ Spatial Keyword Query (*TkSK*) to find λ points with the

Algorithm 1: Incremental Lookup Algorithm

```
Input: Trajectory database $D$, query $Q$, $MT$
Output: Top-$k$ result set $RS$
1  $\lambda \leftarrow k$; $c \leftarrow 0$; $RS \leftarrow \varnothing$;
2  while $\lambda < \lambda^{max}$ do
3      foreach $q_i \in Q$ do
4      if $\lambda < \lambda_i^{max}$ then
5          $R_c(q_i) \leftarrow TkSK(q_i, \lambda, D.P)$;
6      else
7          $R_c(q_i) \leftarrow R_{c-1}(q_i)$;
8      $C_{tra} \leftarrow \bigcup_{i=1}^{|Q|} Covered(R_c(q_i), MT)$;
9      if $|C_{tra}| > k$ then
10         $unseen\_UB = UB_{unseen}(D - C_{tra})$ (Equation 8);
11         $seen\_LB[] = \bigcup_{T \in C_{tra}} LB_{seen}(T)$ (Equation 9);
12         Sort $seen\_LB[]$ in decreasing order;
13         if $seen\_LB[k] > unseen\_UB$ then
14             $seen\_UB[] = \bigcup_{T \in C_{tra}} UB_{seen}(T)$ ( Equation
               10);
15             Sort $C_{tra}$ by $seen\_UB[]$ in decreasing order;
16             foreach $T_i \in C_{\text{tra}}$ do
17                 Compute $\hat{S}(Q, T_i)$;
18                 if $|RS| < k$ then
19                     Insert $T_i$ in $RS$;
20                 else
21                     if $\hat{S}(Q, T_i) > RS.min$ then
22                         Replace $RS.min$ with $T_i$;
23                     if $RS.min > seen\_UB[i+1]$ then
24                         break;
25     $\lambda \leftarrow \lambda + \Delta$; $c \leftarrow c + 1$;
26 return $RS$;
```

highest pointwise similarity (Equation 1) from $D.P$ which are all points in D. Note that we have introduced several solutions for *TkSK* in Section II, and settle on the state-of-the-art method *RCA* [18] based on our experiments, which uses the index structure described in Section VI. Initially, λ is set to k. As a result, we have $|Q|$ ranked lists $R_c(q_i)$ (Lines 3-7). Then for each point in $R_c(q_i)$, we use *MT* which is a point-to-trajectory mapping table to find the affiliated trajectory, resulting in a set C_{tra} of trajectories (Line 8). Now each trajectory in the database D is in one of two groups: (i) Those that have already been seen (contained in C_{tra}); and (ii) Those that have not been seen yet (contained in $D - C_{tra}$).

Step 2: Since C_{tra} stores the candidates for the top-k results, before we compute their real similarities w.r.t. Q (in Step 3), we need to check whether other potential candidates still exist among the unseen trajectories ($D - C_{tra}$).

Trajectories that can not be top-k candidates (in Lines 10-11) are pruned by comparing against the upper bound of the similarity of all unseen trajectories (*unseen_UB*) in $D - C_{\text{tra}}$ using Equation 8, and the k-th lower bound of the seen trajectories C_{tra} (LB_{seen} by Equation 9). Further details on the bounding computation are described in Section IV-B. When the lower bound is not less than the upper bound (Line 14), the candidate set expansion stops, and Step 3 (Lines 15-25) is triggered. Otherwise, the search region is expanded (by incrementally increasing λ by Δ for the query points) to locate more candidate points (in Step 1), and the corresponding trajectories can be subsequently probed. Every iteration of Step 2 is referred to as an *expansion* in this paper, and c is the number of expansions.

Step 3: The top-k results are chosen from the candidate set C_{tra} by computing the upper bound for each candidate trajectory in C_{tra} (Equation 10), and ranking the results in descending order (Lines 14-15). Then the real similarity is computed (Equation 5) between each candidate trajectory T_i and Q (Line 17). If the similarity of the k-th result (*RS.min*) is larger than the upper bound of the next-to-compute trajectory ($UB_{seen}[i+1]$), the algorithm can terminate and return the top-k results (Line 26).

Search Range. Observe that only a point containing at least one query keyword can be a candidate point in $R_c(q_i)$. Hence, the maximum threshold can be set to λ_i^{max} for every query point in Line 4 in Algorithm 1. The maximum length λ_i^{max} of the ranked list $R_c(q_i)$ can be computed as follows:

$$\lambda_i^{max} = \sum_{j=1}^{|q_i.act|} df(q_i.act[j]) \tag{6}$$

where $df()$ records the number of points in $D.P$ that contain the keyword $q_i.act[j]$. When $\lambda_i > \lambda_i^{max}$, the ranked list $R_c(q_i)$ can be maintained during the next round of the expansion, and return the result directly using *TkSK* for q_i in Line 7 because all possible candidate points for q_i have already been probed. As the search range is increased, Δ is bounded by λ_i^{max} for each query point ($\Delta < \lambda_i^{max}$).

In addition to maintaining λ_i^{max}, a global λ^{max} is also set in Line 2 to support early termination. Early termination is defined as:

$$\lambda^{max} = \max_{q_i \in Q} \lambda_i^{max} \tag{7}$$

The ranked list includes all possible trajectories, and there is no need for further expansion. This is because candidate points can only be found after all ranked lists have been probed.

B. Bounding the Computation

The ranked list $R_c(q_i)$ for a query point q_i in the c-th round of expansion is created dynamically as a list of points sorted by their similarity w.r.t. query point q_i. Let $R_c(q_i)[\lambda]$ denote the λ-th point of the sorted list, and $\hat{S}(q_i, R_c(q_i)[\lambda])$ denote the similarity between q_i and the λ-th point, and $R_c(q_i)[\lambda].Tra$ denote the trajectory containing the λ-th point. For each unseen trajectory, no point can occur in any intermediate ranked list. So for each query point q_i, the spatial-textual similarity between q_i and a matching point (if any) in one of the unseen trajectories must be less than $\hat{S}(q_i, R_c(q_i)[\lambda])$. As a result, for any unseen trajectory, the trajectory similarity between the query Q and the trajectory is less than the sum of the minimum similarity $\hat{S}(q_i, R_c(q_i)[\lambda])$. Hence, the upper bound for unseen trajectories from each *TkSK* call is computed as:

$$UB_{unseen}(D - C_{tra}) = \frac{\sum_{i=1}^{|Q|} \hat{S}(q_i, R_c(q_i)[\lambda])}{|Q|} \tag{8}$$

For each trajectory T which has been checked, the existing maximum similarities in all $R_c(q_i)$ can be summed and used as the lower bound of T's similarity, which is less than or equal to the real similarity because points may exist which are not in $R_c(q_i)$:

q_1	p_3^1	0.7	T_3
	p_1^2	0.7	T_1
	p_2^2	0.55	T_2
	p_4^1	0.25	T_4
q_2	p_4^1	0.6	T_4
	p_5^2	0.5	T_5
	p_1^3	0.45	T_1
	p_6^3	0.3	T_6
q_3	p_2^4	0.5	T_2
	p_1^4	0.4	T_1
	p_5^3	0.4	T_5
	p_6^4	0.2	T_6

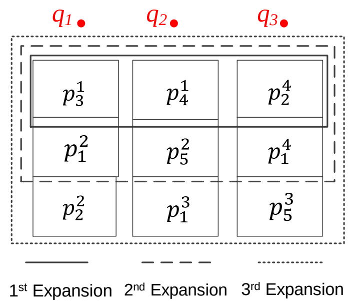

Fig. 3. The Table on the left shows the ranked lists for q_1, q_2, q_3, and the Figure on the right shows the first three rounds of the expansion for Q with $\Delta = 1$.

$$LB_{seen}\left(T\right) = \frac{\sum_{i=1}^{|Q|} \max_{j\in[1,\lambda]\wedge R_c(q_i)[j]\in T} \hat{S}(q_i, R_c\left(q_i\right)[j])}{|Q|} \tag{9}$$

For points in T, but not appearing in $R_c\left(q_i\right)$, their respective similarities can not be greater than the similarity of the λ-th point, $\hat{S}(q_i, R_c\left(q_i\right)[\lambda])$. Thus the upper bound for T's similarity w.r.t. Q is a summation of LB_{seen} and λ-th point's similarity:

$$UB_{seen}\left(T\right) = LB_{seen}\left(T\right) + \frac{\sum_{i=1\wedge T\cap R_c(q_i)=\emptyset}^{|Q|} \hat{S}(q_i, R_c\left(q_i\right)[\lambda])}{|Q|} \tag{10}$$

Note that when $|R_c\left(q_i\right)| < \lambda$, which occurs when the ranked list is not full, $\forall j \in (|R_c\left(q_i\right)|, \lambda]$, $\hat{S}(q_i, R_c\left(q_i\right)[j])$ is set to 0 when computing the bound in Equations 8-10.

We now illustrate how these three bounds are computed using the example shown in Figure 2 and Figure 3. For each query point q_1, q_2, q_3, all candidate points are retrieved and ranked by similarity relative to the associated query point in Figure 3. Here p_i^j in the second column is the j-th point for trajectory T_i, and the third column shows the similarity for the query point, and the last column shows the trajectory containing the point. Next the top-1 trajectory is found using the ranked lists, and Δ is set to 1. In each of the three rounds of expansion, $\Delta = 1$ new points for each query point of Q are searched. The three red points represent the exemplar query, resulting in 3 new points in each round.

Example 3: In the first round of expansion, a top-1 spatial keyword search is performed for every point, and the maximum similarity for each query point is 0.7, 0.6 and 0.5 respectively. Then by Equation 8-9, *unseen_UB* $= \frac{0.7+0.6+0.5}{3} = 0.6$, *seen_LB* $[1] = \frac{0.7}{3} = 0.233$, so the search continues to the top-2 points for each query point since *unseen_UB* $>$ *seen_LB* $[1]$. Then *unseen_UB* $= \frac{0.7+0.5+0.4}{3} = 0.566$, and *seen_LB* $[1] = \frac{0.7+0.4}{3} = 0.366 < 0.566$, so the search continues to the top-3 points. Now, *unseen_UB* $= \frac{0.55+0.45+0.4}{3} = 0.466$, *seen_LB* $[1] = \frac{0.7+0.45+0.4}{3} = 0.51 > 0.466$, so the expansion stops at $c = 3$-rd round. The scanned trajectories T_1, T_2, T_3, T_4, T_5 are checked, and T_6 remains unchecked since it can be safely pruned based on the current upper and lower bounds.

V. Dynamic Expansion

As shown in Step 2 of Algorithm 1, the ranked list is expanded with the parameter Δ until the final candidate set that contains the true top-k results is found. In this section, we will first explain the impact that the choice of Δ has on performance, and then propose a gap-bounded optimization method called **ILA-GAP** to avoid the disadvantages of using a fixed Δ.

A. *Choosing a Δ Increment*

The choice of Δ affects the number of times that *TkSK* is called for each point in Q. Suppose that Algorithm 1 stops at $\lambda_f \in [k, \lambda^{max}]$ in Line 2. Then, *TkSK* is called *Fre* $\left(q_i\right)$ times for each query point q_i:

$$Fre\left(q_i\right) = \begin{cases} \left\lceil \frac{\lambda_f - k}{\Delta} \right\rceil + 1, & \lambda_f < \lambda_i^{max} \\ \left\lceil \frac{\lambda_i^{max} - k}{\Delta} \right\rceil + 1, & \text{otherwise} \end{cases} \tag{11}$$

where *TkSK* is no longer called since all candidate points have been found when $\lambda_f \geq \lambda_i^{max}$.

We can see that *TkSK* in the current round of expansion will repeatedly find results which have been probed in the former round(s) of expansion, so minimizing *Fre* $\left(q_i\right)$ can improve efficiency. For example, if *Fre* $\left(q_i\right) = 1$, the minimum number of expansions is performed. However, it is difficult to achieve the desired goal when using a fixed Δ.

Worst case. For a certain query Q, every trajectory only occurs in one ranked list of Q no matter how large Δ is, and the lower bound of the seen trajectory similarities will never be updated as the search range is expanded. In this case, $\lambda_f = \lambda^{max}$. Moreover, the lower bound can be computed directly based on the top-k points in each ranked list, and the final top-k results can be found using only these points. The upper bound must be decreased to the lower bound based on the maximum difference between the two bounds.

Best case. The lower bound represents the true top-k results when all ranked lists have been fully computed, which implies that the largest lower bound is equal to the upper bound. In this case, $\lambda_f = k$. After the top-k computation is done for every query point, all top-k trajectories have been found, and the whole search process can safely terminate.

However, predicting λ_f or setting a proper Δ in advance that minimizes *Fre* $\left(q_i\right)$ is difficult in practice. In order to reduce *Fre* $\left(q_i\right)$, Δ should be increased. However, this may result in too many trajectories being probed. Conversely, repetitive lookups occur when Δ is also too small. So the efficiency heavily depends on the choice of Δ, which in turn depends on the collection and the query.

Instead of trying to tune Δ for each dataset, our aim is to dynamically compute the difference between the upper and lower bound, and use it as the increment in the next expansion round in order to achieve a faster convergence. Consider the following definition.

Definition 8: (**Bound Gap**) A Bounded Gap *BG* (c) is defined as the difference between the lower bound and the upper bound in the c-th round of expansion, namely:

$$\begin{aligned} BG\left(c\right) &= |unseen\_UB - seen\_LB[k]| \\ &\leq \frac{\sum_{i=1}^{|Q|} \hat{S}(q_i, R_c\left(q_i\right)[\lambda])}{|Q|} - \sum_{q_i\in Q} TkSK\left(q_i, k, D.P\right)[k] \end{aligned} \tag{12}$$

As λ is increased by Δ in a new expansion, the upper bound will decrease and the lower bound will increase until the lower bound is not less than the upper bound. So *BG* (c) is reduced monotonically. Notice that the biggest challenge is how to make this happen efficiently while scanning fewer trajectories.

B. Adaptive Gap-bounded Expansion

To expand the ranked lists for a query after knowing the gap between the two bounds, the simplest solution is to assign an equal decreasing similarity by dividing the gap by $|Q|$ for each query point. However, we can further improve the lower bound *seen_LB*$[k]$ and terminate the algorithm earlier by assigning a different decreasing similarity for each query point. This diversification of the expansion has been used successfully by Chen et al. [3] in previous work, who dynamically assigned a different Δ based on the sparsity of the area around a scanned query point. Chen et al. proved that higher priority given to the query points which scan sparse areas containing fewer candidates can help increase the lower bound, and improve efficiency by reducing the number of candidate trajectories. Similarly, we assign a higher decreasing similarity to query points in a sparse area. This can be achieved by checking how many new trajectories are probed by each query point in each round of the expansion. Using a decreasing similarity for the ranked list of two adjacent expansions, we propose the concept of *similarity sparsity*.

Definition 9: (**Similarity Sparsity**) Given a decrease of the minimum similarity in ranking list from $\hat{S}_c(q_i)$ to $\hat{S}_{c-1}(q_i)$, and the number of newly scanned trajectories in the c-th expansion, the similarity sparsity is defined as:

$$Spa_c(q_i) = \frac{\hat{S}_{c-1}(q_i) - \hat{S}_c(q_i)}{|R_c(q_i).T_m| - |R_{c-1}(q_i).T_m|} \tag{13}$$

where $\hat{S}_c(q_i)$ is the minimum similarity in the ranking list of q_i in the c-th expansion, and $|R_c(q_i).T_m|$ is the number of trajectories covered by $R_c(q_i)$, where $\hat{S}_0(q_i)$ is set to the maximum similarity in the ranked list of q_i initially.

A larger $Spa_c(q_i)$ means fewer trajectories are found, so in the next round, a higher similarity should be assigned to the query points in order to find fewer new trajectories, and therefore increase the lower bound faster. Hence, the new minimum similarity for the query point q_i of the next expansion is computed as:

$$\hat{S}_{c+1}(q_i) = \hat{S}_c(q_i) - \frac{BG(c)}{2} \cdot |Q| \cdot \frac{Spa_c(q_i)}{\sum\limits_{q_j \in Q} Spa_c(q_j)} \tag{14}$$

where $\frac{Spa_c(q_i)}{\sum_{q_j \in Q} Spa_c(q_j)}$ is the proportion that the upper bound is decreased based on the similarity sparsity, and the new upper bound expected is reduced by $\frac{BG(c)}{2}$ in the $(c+1)$-th expansion.

After computing the minimum similarity in the next expansion, it will be used to find new candidate points with a higher similarity than it instead of finding Δ new points for every query point in **ILA**. Instead, *TkSK* is expanded to search for the points whose similarities are greater than $\hat{S}_{c+1}(q_i)$. This search is referred to as a *Top Similarity Spatial Keyword Query* (*TsSK*) henceforth. In contrast to *TkSK*, *TsSK* does not maintain a result heap or update the k-th similarity, so pruning is more efficient. For any TkSK which employs filtering unseen objects based on upper bound, we can easily extend and set the upper bound as the similarity and compute the number of iterations we need to get this position directly, so it will jump to this iteration directly, which make scanning significantly more efficient. Based on the idea of gap bounding, and *TsSK*, the refined Algorithm 2 is significantly more efficient and robust when compared with the **ILA** approach described in Section IV.

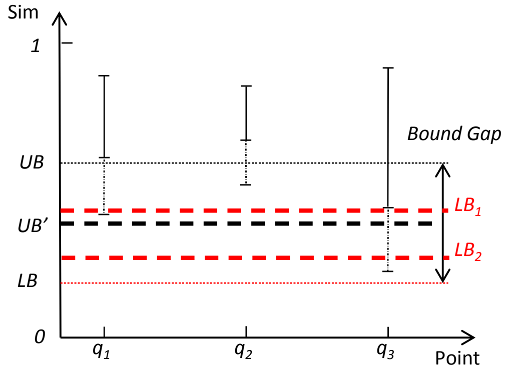

Fig. 4. Gap-bounded Expansion

Algorithm 2: Gap-bounded Algorithm

```
Input: Trajectory database $D$, query Q
Output: Top $k$ result set $RS$
1  $RS \leftarrow \varnothing$; $c \leftarrow 0$;
2  foreach $q_i \in Q$ do
3      $R_c(q_i) \leftarrow TkSK(q_i, k, D.P)$;
4  while TRUE do
5      $C_{tra} \leftarrow \bigcup_{i=1}^{|Q|} Covered(R_c(q_i))$;
6      if $|C_{tra}| > k$ then
7          Same as line 10-24 in Algorithm 1;
8          foreach $q_i \in Q$ do
9              if $|R_c(q_i)| < \lambda_i^{max}$ then
10                 Update $\hat{S}_{c+1}(q_i)$ using Equation (16);
11                 $R_{c+1}(q_i) \leftarrow TsSK(q_i, \hat{S}_{c+1}(q_i), D.P)$;
12             else
13                 $R_{c+1}(q_i) \leftarrow R_c(q_i)$;
14     $c \leftarrow c + 1$;
15 return $RS$;
```

The biggest difference between Algorithm 2 and Algorithm 1 is that *TkSK* in no longer used in Line 3 after the first expansion. Instead, if the lower bound is still smaller than the upper bound, the minimum similarity $\hat{S}_{c+1}(q_i)$ is updated for every query point q_i used in the $(c+1)$-th expansion in Line 10. Then, *TsSK* is called to find all the points with a similarity greater than $\hat{S}_{c+1}(q_i)$ in Line 11. This is illustrated in the example below.

Figure 4 shows how to accelerate the lookup process based on gap bounding. We can see that when the three query points (x-axis) have different similarity ranges (y-axis), the upper bound *UB* (black dotted line) can be calculated by summing the minimum of each range, and also the lower bound *LB* (red dotted line) for the covered trajectories can be found. Then, the gap bound can be computed, and used to predict the upper bound UB' for the next round (the bold black dotted line). Next, the similarity bound is assigned to q_1, q_2, q_3 using

Equation 14, and a new ranked list and a new lower bound is computed. Here, there are two possible cases. Case 1: If the new lower bound is larger than the predicted upper bound $UB^{'}$, namely $LB_1 > UB^{'}$, then the algorithm can terminate. Case 2: Otherwise, $LB_2 < UB^{'}$ and iterations continue using the monotonically decreasing gap bound. Now consider Figure 4 and the refined Algorithm 2 which shows an example of how the pruning efficiency is improved using our new technique.

Example 4: After the first expansion, $unseen\_UB = 0.6$, and $LB_{seen}[1] = 0.233$, so $BG(1) = 0.6 - 0.233 = 0.367$. Instead of setting $\Delta = 1$ to do the expansion, we assign a customized value to each q_i based on Equation (16). So, $\hat{S}_2(q_1) = 0.7 - \frac{0.367}{2} \times 3 \times \frac{0.7}{0.7+0.6+0.5} = 0.485$, $\hat{S}_2(q_2) = 0.416$ and $\hat{S}_3(q_3) = 0.347$. Accordingly in Algorithm 2, *TsSK* is called to find the points whose similarity is not less than $\hat{S}_2(q_i)$. Then, the same ranked lists with Example 3 which needed three expansions are got by only two expansions using **ILA-GAP**. The Gap-bounded algorithm improves the likelihood of early termination.

VI. Two-level Threshold algorithm

Both **ILA** and **ILA-GAP** exploit the *TkSK* method during incremental expansion to filter out impossible results by comparing the k-th result and the upper bound of any unseen trajectories. However, recall from Line 5 of Algorithm 1, that the same data points already probed in previous round(s) of the expansion will be rescanned by *TkSK* in each iteration. We refer to this as repetitive lookup in this paper. Recall from Examples 4, 5, and Figure 3 that if we use **ILA** in Example 4, p_3^1, p_2^4, p_4^1 are scanned three times, and p_1^4, p_5^2, p_1^2 are scanned twice. In Example 5 where **ILA-GAP** is used, p_3^1, p_2^4, p_4^1 still need to be scanned twice.

Repetitive lookups also exist in spatial-only trajectory search problems [3]. Qi et al. [11] used an R-tree to perform spatial range queries for each query point to fill the ranked list, where the radius is gradually increased in order to avoid repetitive lookups. However, the R-tree based expansion described by Qi et al. [11] was designed for spatial-only expansion, and it is not clear how to extend their solution to the spatial-textual trajectory search problem we are exploring here. In this section, we will describe how to support non-repetitive expansion in both spatial and textual dimensions concurrently, which can be used to compute a new upper bound similarity for unseen trajectories, and achieve efficient rank-safe search.

A. Framework

The core component of our framework is the index used to fill the ranked list non-repetitively and further filter out unseen trajectories which cannot make it into the top-k. As shown in Figure 5, the indexes used here are an inverted list, and grid index, which are used as follows:

1) A rank-ordered inverted index for the terms. For every keyword t, the points which contain t are sorted by their TF·IDF weight in a descending order.
2) A grid-index labeled using a Z-curve that supports range queries by accessing the leaf node around each query point directly with an increasing radius.

Based on above index, the processing framework of **2TA** in Figure 5 is composed of two levels. In the first level, each query point is decomposed into keywords $t_1, \ldots, t_n$ and a location *loc*. The main objective is to scan candidates in

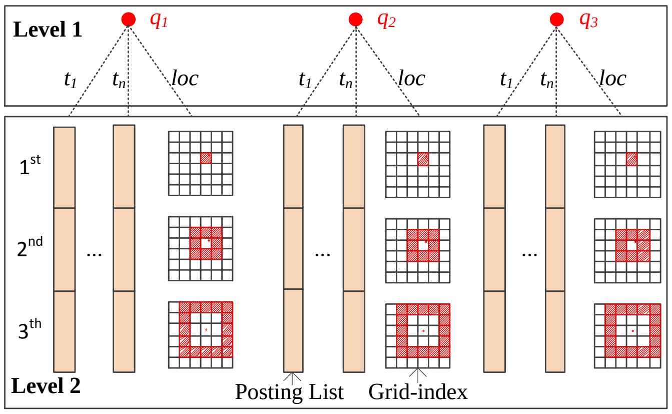

Fig. 5. Incremental expansion on textual (a) and spatial (b) similarity

the lower level using the keyword posting lists, and the grid-index for the location. In the second level, the posting lists of $t_1, \ldots, t_n$ are scanned blockwise which is denoted by a rectangle in the posting list, and stores a set of points that share the same range by weight of keywords. Meanwhile, the cells around the query point are scanned layer by layer, where points which are closer to the query point are scanned preferentially. The red cells are loaded in each iteration. When new points are scanned, they will be inserted into the ranked list of the query point for further computation and refinement.

For an unseen trajectory which has not been scanned, according to Definition 6 and Definition 4, the upper bound of the similarity for unseen trajectories can be computed by summing up the maximum score of the current block in each posting list, and in the grid index as:

$$UB_{unseen}(D - C_{tra}) = \alpha \cdot \sum_{t \in Q} UB_t(t, it)/|Q| + (1-\alpha) \cdot \sum_{loc \in Q} UB_s(loc, it)/|Q| \quad (15)$$

where *it* is the *it*-th iteration on each keyword and location, $UB_t(t, it)$ ($UB_s(loc, it)$) is the maximum textual (spatial) score fetched from the *it*-th iteration for keyword t (location *loc*). Based on above bound, we have the following lemma to filter unseen trajectories.

Lemma 1: For a trajectory T which is not scanned by any posting list or grid-index of Q, $\hat{S}(T, Q) \le UB_{unseen}(D - C_{tra})$.

Proof: Since T does exist in the ranked list, for any point p in T we have $\hat{S}(q_i, p) \le \alpha \cdot \sum_{t \in q_i} UB_t(t, it) + (1 - \alpha) \cdot UB_s(q_i.loc, it)$. Then $\hat{S}(T, Q) = \sum_{q_i \in Q} (\hat{S}(q_i, T))/|Q| \le \sum_{q_i \in Q} (\alpha \cdot \sum_{t \in q_i} UB_t(t, it) + (1 - \alpha) \cdot UB_s(q_i.loc, it))/|Q| = UB_{unseen}(D - C_{tra})$. ■

Our approach draws inspiration from the RCA algorithm of Zhang et al. [18], which processes a top-k spatial keyword search using posting lists of query keywords which are organized using a Z-curve. As the similarity is the sum of the spatial and textual sub-similarities, RCA processes TkSK as a top-k aggregation query [6]. When compared with RCA, the main difference is that our algorithm must search trajectories instead of points. After identifying the points, further bound refinements are necessary, and unrelated trajectories must be filtered out. RCA only scans the posting lists for a single query point, and adds candidate to the result heap until the similarity of the k-th item is not less than the upper bound of the unprocessed points.

To reduce the cost of random access, points are scanned in a batch for both spatial and textual data instead of accessing a single point for each similarity computation, as shown in Figure 5. Similar to RCA, **2TA** divides each posting list into $it^t_{\max}$ blocks, and the whole space of the Z-curve into $it^s_{\max}$ blocks for a given location *loc*. In each iteration, **2TA** loads one block from each list concurrently. Here we will give a brief description on how to divide the list scanning into a fixed number of iterations and upper bound in it-th iteration.

Textual Scanning. The scanning operation in an inverted term list is called *ExploreTextual*(t, it). As different keywords have different weight ranges, in order to make sure the scanned blocks are not empty, the maximum and minimum weights of each keyword are computed as $\gamma_{max}(t)$ and $\gamma_{min}(t)$. Then the range $[\gamma_{max}(t), \gamma_{min}(t)]$ is divided into $it^t_{\max}$ intervals and filled with points according to their weight. This ensures that none of the scanned blocks are empty. For the *it*-th block of keyword t, the upper bound can be computed as:

$$UB_t(t, it) = \frac{\gamma_{max}(t) - \gamma_{min}(t)}{it_{max}} \cdot (it_{max} - it) + \gamma_{min}(t) \quad (16)$$

For instance, Figure 6 shows the posting list of "coffee" which contains three blocks, and each block stores the points which are bounded by a weight range. The first block stores all of the points containing "coffee" with a weight between 0.53 and 0.7, such as p^4_2, p^3_2 and p^6_3, and *ExploreTextual*$(``coffee'', 1)$ will load these three points. After loading the first block in the first iteration, the upper bound similarity of "coffee" is 0.53.

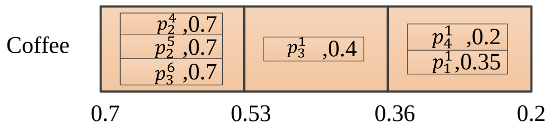

Fig. 6. Posting list of "coffee" with three blocks.

Spatial Scanning. In order to compute spatial similarity using a grid-index (labeled with a Z-curve), the function *ExploreSpatial*(loc, it) in the iteration *it* can be used. Given a query location *loc*, the cells around *loc* within a radius covered by a range of Z-curve labels $[L_{\min}, L_{\max}]$ which includes the label L_{loc} of cell the where *loc* is located ($L_{loc} \in [L_{\min}, L_{\max}]$). By expanding the range $[L_{\min}, L_{\max}]$, cells that are further away can be loaded gradually. More details can be found in the work of Zhang et al. [18]. Then the upper bound of the spatial similarity for loc in the it-th iteration can be computed as:

$$UB_s(loc, it) = \frac{D_{max} - it \cdot \frac{D_{max}}{it_{max}}}{D_{max}} \quad (17)$$

where D_{max} is the maximum distance between any two unique points in geographical space.

Note that in the case that the points in one posting list are all scanned but the grid index is not, we set $it^s_{\max} = it^t_{\max}$. When $it^s_{\max} < it^t_{\max}$, the worst case occurs when all spatial candidates have been scanned, but the search continues to access the term posting lists. However, there is no need to continue scanning as all candidates have been identified, so the additional work is unnecessary. Similarly, when $it^s_{\max} > it^t_{\max}$, the same problem occurs when all postings lists have been exhausted, but unscanned points remain in the grid index. Hence, we can conclude that when $it^t_{\max} = it^s_{\max}$, the worst case occurs, and points being scanned cannot affect the final top-k list. Here $it_{\max}$ is used to represent both cases, and is a key parameter that will affect the performance of queries, as the size of the blocks depends on this value. In our experiments, $it_{\max}$ is found using a parameter sweep as it is dataset specific.

B. Two-level Threshold Algorithm

Algorithm 3: Two-level TA algorithm

```
Input: Trajectory database $D$, query $Q$, $MT$
Output: Top-$k$ result set $RS$
1  $it \leftarrow 0$; $RS \leftarrow \varnothing$;
2  while $it < it_{max}$ do
3      foreach $q_i \in Q$ do
4          $R_{it}(q_i) \leftarrow ExploreTextual(q_i, it, D.P)$;
5          $R_{it}(q_i) \leftarrow ExploreSpatial(q_i, it, D.P)$;
6      $C_{tra} \leftarrow \bigcup_{i=1}^{|Q|} Covered(R_{it}(q_i), MT)$;
7      if $|C_{tra}| > k$ then
8          $unseen\_UB = UB_{unseen}(D - C_{\text{tra}})$ (by Equation 15);
9          $seen\_LB[] = \bigcup_{T \in C_{tra}} LB_{seen}(T)$ (by Equation 9);
10         Sort $seen\_LB[]$ in decreasing order;
11         if $seen\_LB[k] > unseen\_UB$ then
12             $C_{tra} \leftarrow \varnothing$;
13             for each $q_i \in Q$ do
14                 for $p_j \in R_{it}(q_i)$ do
15                     if $\hat{S}(q_i, p_j) \geq UB_{it}(q_i)$ then
16                         $C_{tra} \leftarrow Covered(p_j, MT)$;
17             Same as line 15-26 in Algorithm 1;
18             return $RS$;
19     $it$++;
```

Algorithm 3 presents the details of **2TA** when using an inverted list and a grid index. Similar to **ILA** and **ILA-GAP**, the new algorithm still conducts incremental iteration using a while loop (Line 2). A significant enhancement in **2TA** is that the expansion is performed at the keyword level and the location level concurrently rather than performing them at the point level first. In particular, Line 4 computes similarity for the inverted lists (based on the keyword t) to retrieve the points with the highest textual similarity, and Line 5 computes the similarity of the locations using the grid index (based on location *loc*) to retrieve the spatially closest points. Then the trajectories C_{tra} are found for the points in Line 6. Another significant difference in **2TA** is the second round of filtering (Line 12-16). Here, points in C_{tra} that have a lower bound less than the bound of single query point are filtered out, so the number of trajectories that need to be refined is reduced, before computing the spatial-textual similarity. Finally, the same refinement process as **ILA** and **ILA-GAP** (Line 17) is used to find the final top-k results. Next we will show why a second round of filtering is required.

Second Round Filtering. After all of the unseen trajectories whose similarities are less than $UB_{unseen}(D - C_{tra})$ are pruned, the second round of filtering begins. This is required since the current set of candidates may still contain trajectories whose similarities are smaller than the upper bound that was computed using Equation 15, and therefore cannot be in the top-k. To understand why, consider the following lemma:

Lemma 2: For $q_i \in Q$, $\exists p \in R_{it}(q_i)$, $\hat{S}(p, q_i) < UB_{it}(q_i)$. where $UB_{it}(q_i)$ is similarity upper bound of all unprocessed points for q_i in the *it*-th iteration, which is computed as:

$$UB_{it}(q_i) = \alpha \cdot \sum_{t \in q_i} UB_t(t, it) + (1 - \alpha) \cdot UB_s(q_i.loc, it) \quad (18)$$

Proof: Assume that a point p contains only one keyword t in q_i, and has an equal bounded weight for t, such that $\hat{S}(p, q_i) = UB_t(t, it)$. The point p can be scanned and inserted into $R_{it}(q_i)$, where according to Equation 18, $UB_{it}(q_i) > UB_t(t, it)$. So, $\hat{S}(p, q_i) < UB_{it}(q_i)$. ■

In the final expansion, points like p can be found in the ranked list of every query point, which means that for a scanned trajectory T, if the similarity of each query point q_i is less than $UB_{it}(q_i)$, then $\hat{S}(Q, T) < UB_{unseen}(D - C_{tra}) = \sum_{q_i \in Q} (UB_{it}(q_i))/|Q|$, which implies that T cannot be a top-k result. Hence, filtering out trajectories such as T before the final stage of refinement is desirable.

A second round of filtering is performed to filter the bottom part of $R_{it}(q_i)$, as shown in Lines 12-16 of Algorithm 3. For every query point q_i, the points with a similarity greater than $UB_{it}(q_i)$ are maintained, and the remaining points are dropped. Then the union of all of the points is taken to find the parent trajectories using the mapping table *MT*. The ranked lists of q_i are divided by $UB_{it}(q_i)$, and are used to maintain the top half of $R_{it}(q_i)$ as a new ranked list $R^{'}_{it}(q_i)$. Based on the new ranked lists, the upper bound for trajectories that are not covered can be computed using Equation 8 as $\sum_{q_i \in Q} (UB_{it}(q_i))/|Q| = UB_{unseen}(D - C_{tra})$, which is less than $seen\_LB[k]$ according to the condition in Line 11 of Algorithm 3, So, the uncovered trajectories can not be the top-k results.

Comparison with ILA. Note that after the second round of filtering, the new ranked lists $R^{'}_{it}(q_i)$ are the target of **ILA**. As the upper bound calculated from $R^{'}_{it}(q_i)$ using Equation 8 is less than $seen\_LB[k]$ derived from Equation 9, filtering in **ILA** terminates. **2TA** can achieve the same result with fewer expansions.

Another efficiency concern in **2TA** is that the algorithm has to scan additional points which are removed in the second round of filtering. It should be noted that **ILA** also needs to scan the same number of points in the final expansion when using TkSK. Consider the example in Figure 5 where RCA is used with TkSK and the query point q_1. For q_1, RCA will scan the posting list and grid index to find the top-λ points for the ranked list in **ILA** until the upper bound is smaller than the λ-th result. We denote the similarity of the λ-th item as $\hat{S}(q_i, p)$. The search stops at $UB_{it}(q_i) \leq \hat{S}(q_i, p)$. To determine the ranked list with λ points, **ILA** has the same upper bound, but scans more than λ candidates for the query point. The candidates outside the top-λ points are actually the same points dropped in second round of filtering by $UB_{it}(q_i)$ in **2TA**.

Generality. Our framework can also support spatial-only trajectory search [3, 11, 15]. As shown in Figure 5, the spatial expansion can be achieved using a range query in the grid-index. For spatial-only search, the inverted term lists are ignored during the expansion. When compared with an R-tree based expansion [11], the grid index can access the leaf node around the query point directly, while an R-tree has to traverse from root to the leaf node, which is less efficient in practice.

Para	**Value**
$\lvert Q\rvert$	1,2,3,4,$\underline{5}$,6,7,8,9,10
$\lvert q.act\rvert$	$\underline{3}$,4,5,6,7,8,9,10
k	10,$\underline{20}$,30,40,50,60
Δ	1000
α	0.5
$it_{\max}$	150

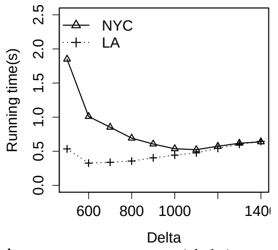

Fig. 7. Parameter choices (left), and a Δ parameter sweep (right).

VII. Order-Sensitive Search

In the previous three sections, an incremental expansion algorithm was proposed for top-k spatial keyword search over trajectories using a set of *unordered* query points.

In this section, we discuss how to process an *order-sensitive* query using a new similarity metric which considers the ordering of points in the query instead of treating it as a set of points. First, we define *order-sensitive similarity*, and show how to leverage our algorithms to efficiently process this new query type.

Definition 10: **(Order-sensitive Trajectory Similarity)** We define the order-sensitive trajectory similarity between $\mathcal{T}$ and Q as:

$$\hat{S}_o(Q, T) = \begin{cases} 0, & Q = \varnothing \quad or \quad T = \varnothing \\ \max(\hat{S}_o(q, t) + \hat{S}_o(Rest(Q), T), & \\ \hat{S}_o(Q, Rest(T))), & \text{otherwise} \end{cases} \quad (19)$$

where q and t are the first point in Q and T respectively, and $Rest(Q)$ and $Rest(T)$ indicate the remaining trajectories excluding the points q and t. This similarity computation yields to a dynamic programming solution [3] with a complexity of $O(n^2)$.

The Order-sensitive Exemplar Trajectory Query can be formulated by replacing the $\hat{S}(Q, T)$ with $\hat{S}_o(Q, T)$ in Definition 7 directly. For the order-sensitive search, we can extend the above algorithms for search. The upper bound will not change as the highest similarities are summed together as in the unordered case. However, the lower bound will change as the ordering constraint can produce smaller similarities. The computation can be executed using the recursion in Equation 19 and dynamic programming. The new bounds of trajectory T can be computed as follows:

$$LB^{o}_{seen}(T) = \hat{S}_o(Q, T^{'}) \quad (20)$$

where $T^{'}$ is sub-trajectory composed by all searched points in the top-k lists found during the incremental expansion.

$$UB^{o}_{seen}(T) = \hat{S}_o(Q, T^{'}) + \frac{\sum\limits_{i=1 \wedge T \cap R_c(q_i) = \varnothing}^{|Q|} \hat{S}(q_i, R_c(q_i)[\lambda])}{|Q|} \quad (21)$$

where the second part is the sum of the λ-th similarity in the top-k lists belonging to the uncovered points of T.

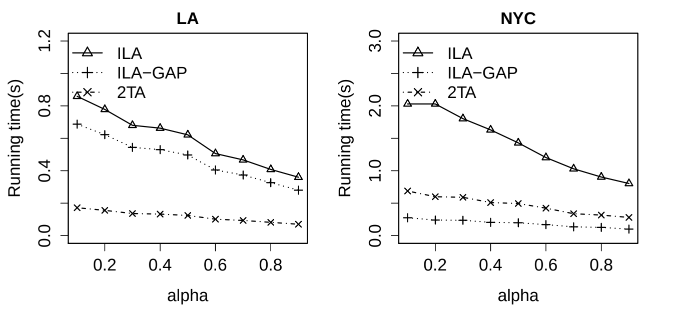

Fig. 8. Effect of α on the total running time for the LA (left) and NYC (right) collections.

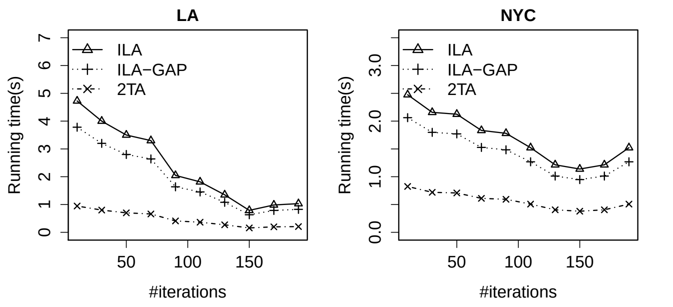

Fig. 9. Effect of iteration times on the total running time for the LA (left) and NYC (right) collections.

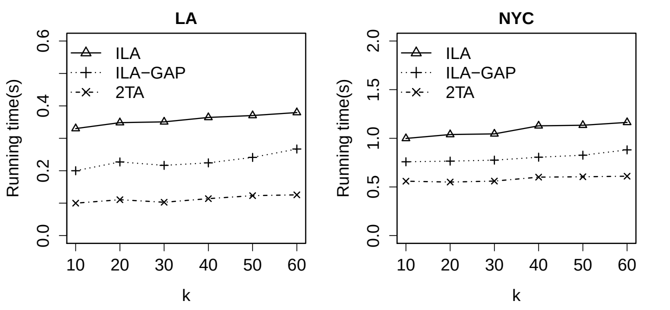

Fig. 10. Effect of k on the total running time for the LA (left) and NYC (right) collections.

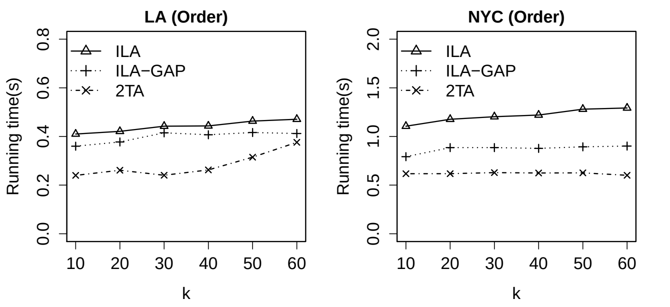

Fig. 11. Effect of k on the total running time for the LA (left) and NYC (right) collections with order constraint.

VIII. Experiments

A. *Experiment Setup*

Dataset. Two trajectory datasets from Foursquare are used: Los Angeles (LA) and New York (NY) [2, 7, 20]. Table IV summarizes the number of trajectories, average length of trajectory and the number of keywords in the trajectories.

Trajectory queries are generated as follows: First, we collect all unique keywords and their frequencies in each dataset, and randomly pick keywords based on the term's distribution to produce the text description of a point. Then, we extract sub-trajectories of various lengths from the dataset to form the spatial component of the points. Finally, we randomly pair keywords with points to form a query. A total of 100 queries are generated where the length per query varies from 3 to 10. All experimental results are averaged by running all 100 queries. Algorithms are implemented in Java, and ran on a PC with a 3.30GHz CPU and 8GB RAM using Ubuntu 14.04.

Parameters. Figure 7 summarizes all of the parameters used in this work, and the default values are underlined. For the choice of search range increment Δ, a parameter sweep was ran for both datasets, and the results are shown in Figure 7. The running time decreases as Δ increases since fewer expansions are needed before termination. However, using a very large Δ can result in more unnecessary similarity computations. In this work, we choose a default cutoff value of $\Delta = 1{,}000$ (for the baseline method **ILA**) as this is the earliest point that the parameter converges in both datasets. A local minima could be selected for each dataset, but doing so runs the risk of overtuning and biasing the experimental results.

Algorithms for evaluation. We compare the following methods to process **ETQ** over two datasets. (1) **ILA**: The baseline method proposed in section IV forming the framework of this paper. Note that for the incremental expansion on spatial similarity, we adopt the state-of-the-art IKNN method developed by Qi et al. [12]. (2) **ILA-GAP**: The gap-based dynamic expansion method built on top of the baseline proposed in Section V. (3) **2TA**: The proposed two-level threshold algorithm which avoids repetitive search of points in Section VI.

TABLE IV. FourSquare Datasets

Dataset	$\lvert D\rvert$	$\lvert D.P\rvert$	$\lvert T\rvert$	$\lvert q.act\rvert$
LA	31,557	215,614	6.83	14.67
NYC	49,027	206,416	4.21	9.96

Evaluation Metrics. We will explore the running time of the algorithms proposed in Section VIII-C. In Section VIII-D we explore the impact of several different variables on overall performance.

B. *Effect of parameters*

First, we explore the effect of two critical parameters: the textual-spatial weighting α, and the maximum number of iterations required for convergence, it_{max}, when scanning all of the posting lists in the **2TA** algorithm.

As shown in Figures 8(a) and 8(b), when more weight is put on spatial similarity, the running time for all methods declines, which suggests that the pruning effect of the spatial component is more significant than in the textual component.

Figure 9 shows the performance differences as it_{max} is increased. The running time declines until it_{max} converges to 150 in both datasets. This is because fewer iterations cover more non-top-k trajectories in the last iteration round. After 150, the running time begins to increase. This is because fewer iterations can reduce the number of grid cell accesses. Thus more cells and more candidates are covered in a single iteration, and more time is spent accessing the candidates when it_{max} is large.

This suggests that we can choose a single parameter for both datasets easily as there is a clear similarity in the performance characteristics. All three algorithms rely on the choice of it_{max}, and show similar trends. Only the **2TA** can efficiently perform the parameter sweep. For α, if users want to have more weight on textual similarity, efficiency is reduced,

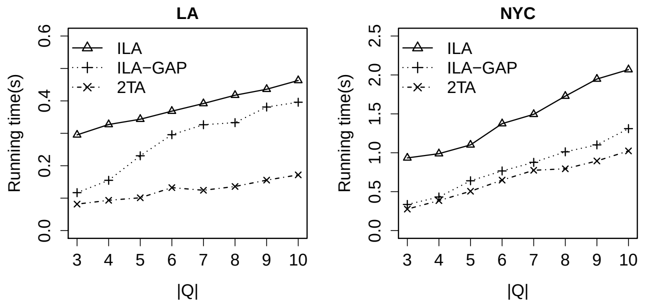

Fig. 12. Effect of $|Q|$ on the total running time for the LA (left) and NYC (right) collections.

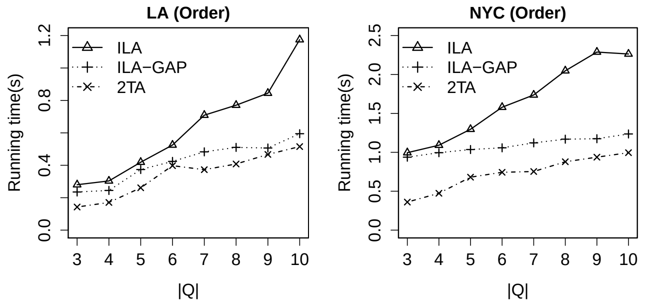

Fig. 13. Effect of $|Q|$ on the total running time for the LA (left) and NYC (right) collections with order constraint.

but overall performance is still acceptable. Henceforth, we set $it_{max} = 150$ and $\alpha = 0.5$.

C. Performance Evaluation

Before presenting further experimental details, a summary of the main observations is:

1) For three main query dependent parameters $|Q|$, k and $|q.act|$, **2TA** has the best performance: **2TA** is at least two times faster than the baseline method **ILA**, and can be five times faster in some cases.
2) The parameter k has the smallest effect to the performance.
3) The parameter $|q.act|$ affects the performance the most. More keywords in the query increases the number of candidate trajectories.

Effect of $|Q|$. Figure 12 shows that the running time for all of methods increases linearly w.r.t. $|Q|$ in both datasets. In particular, **ILA-GAP** and **2TA** have a performance between 0.07 and 0.3 second in both datasets, while **ILA** needs 1 - 2 seconds per query. The reasons for the performance difference are two-fold: (1) When compared to **ILA-GAP** which manages to compute a dynamic yet tighter increment in each round of expansion, **ILA** uses a fixed Δ value for the increment, so **ILA** usually needs 2 more rounds of expansion to stop, leading to scanning more points; (2) When comparing against **2TA**, **ILA** has to perform repetitive searches on points found in previous rounds of the expansion which are not required in **2TA**.

Effect of $|Q|$ on order-sensitive search. Figure 13 shows that it costs more time to conduct all queries compared with non-ordered search for all methods. There are two reasons: (1) The computation of similarity based on dynamic programming is time consuming (complexity of $O(n^2)$); (2) The gap between the lower bound and the upper bound is greater than that in non-ordered search.

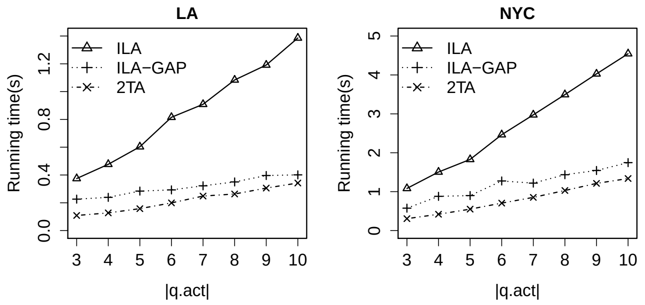

Fig. 14. Effect of $|q.act|$ on the total running time for the LA (left) and NYC (right) collections.

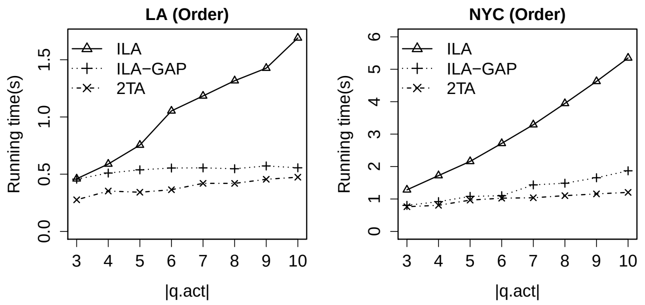

Fig. 15. Effect of $|q.act|$ on the total running time for the LA (left) and NYC (right) collections with order constraint.

Effect of k. Figure 10 presents the running time w.r.t. k. We find that the running time not heavily influenced by the choice of k, but the number of points in the query $|Q|$ clearly has an influence on performance, as shown in Figure 12. One possible reason is that the lower bound of the k-th result does not change too much as k increases, so it does not result in more expansion rounds, which is the main performance bottleneck in all of the algorithms. In contrast, when the number of query points increases, the bounds do change as more ranked lists are required to compute the upper bound for the unseen trajectories, and the lower bound of the seen trajectories, which translates into more computational costs.

Effect of k on order-sensitive search. When comparing Figure 11 and 10, we see that the ordering constraint does not incur an extra cost in query processing time, which is not sensitive to k.

Effect of $|q.act|$. The relationship between the number of keywords in each query point and the performance is presented in Figure 14. The number of query points is fixed at 5, and the number of keywords is increased for each query point (from 3 to 10). The running time of **ILA** and **ILA-GAP** increases much faster than **2TA** with respect to $|Q|$ and k. As more keywords are added, more posting lists must be scanned, and so the gap bound between the upper and lower bound is greater, which in turn results in more expansion rounds on average. In contrast, **2TA** can achieve is more robust as scanning repetition is minimized in each round of the expansion.

Effect of $|q.act|$ on order-sensitive search. When an ordering constraint is applied, Figure 15 shows a similar trend with the non-ordered search, but is less efficient in all three methods since the similarity computation and more data points are scanned which contain common keyword(s).

TABLE V. A COMPREHENSIVE ANALYSIS SHOWING THE AVERAGE NUMBER OF EXPANSIONS c, ITERATIONS it_{max}, THE LOWER BOUND *seen_LB* OF SEEN TRAJECTORIES, THE UPPER BOUND *unseen_UB* OF UNSEEN TRAJECTORIES, AND THE NUMBER OF RANDOM ACCESS OF THE POINTS *rp* AND TRAJECTORIES *rt*, *frp* IS NUMBER OF POINTS SHOWN IN ALL FINAL RANKING LISTS OF QUERY.

Type	Method	LA							NYC						
		rp	*frp*	*rt*	*unseen_UB*	*seen_LB*	it_{max}	*c*	*rp*	*frp*	*rt*	*unseen_UB*	*seen_LB*	it_{max}	*c*
Un-order	**ILA**	29729	8143	4980	1.8471	2.3021	NA	2.7	55790	19242	8732	3.4275	3.5115	NA	2.8
	ILA-GAP	26280	7012	3721	2.1740	2.2890	NA	2.2	52716	17324	8012	3.4319	3.5034	NA	2.3
	2TA	**10383**	**5383**	**2943**	**2.2341**	**2.2517**	117	NA	**32343**	**15771**	**7120**	**3.4632**	**3.4712**	121	NA
Order	**ILA**	33085	10434	5762	1.62025	2.1552	NA	2.9	63935	27095	10234	3.0033	3.0862	NA	2.9
	ILA-GAP	31688	9122	4211	1.9668	2.0687	NA	2.3	61488	25459	9790	3.0157	3.0936	NA	2.4
	2TA	**12374**	**6134**	**3020**	**2.0342**	**2.0510**	130	NA	**32896**	**23277**	**8143**	**3.0491**	**3.5897**	132	NA

D. In-depth Analysis behind Efficiency

To better illustrate the advantages of **ILA-GAP** and **2TA**, we now conduct an in-depth performance analysis on three more aspects: (1) the number of expansions, (2) the number of random accesses on points and trajectories, and (3) the final lower and upper bound when query processing terminates. Details for both non-ordered and order-sensitive search are shown in Table V, and the best result in each column is highlighted in bold. In summary, We make six main observations:

(1) **2TA** scans the minimum number of points and trajectories among all methods in both datasets.

(2) Regarding the number of expansions c which only applies to **ILA** and **ILA-GAP**, a bigger c means that more points will be scanned. **ILA-GAP** requires fewer expansions to find the top-k candidate set than **ILA**, and also eliminates the effect of the increment Δ, which is fixed in **ILA**.

(3) Regarding the final bound values when query processing terminates, the upper bound of all three methods are nearly identical (the difference is within 1 decimal point), and the gap between the lower and upper bound of **2TA** is smaller than **ILA-GAP**, which is in turn smaller than **ILA**. This is in line with our previous observations that **ILA-GAP** with a tighter increment is able to scan fewer points than **ILA**.

(4) By comparing **2TA** and **ILA-GAP** in Section VIII-C, we conclude that controlling the number of points scanned is the critical factor in the pruning step.

(5) For **2TA**, the non-ordered search needs fewer iterations when compared with the order-sensitive search, but is still more robust than the other approaches.

(6) The second round of filtering in **2TA** can filter extra points effectively. In the LA dataset, it scans $10{,}383$ points in the first iteration, an $5{,}383$ additional points remain in the top-k for further refinement.

IX. CONCLUSION

In this work, we introduced the top-k exemplar trajectory search problem, and proposed a framework using incremental expansion and pointwise similarity. We have identified that the most critical aspect in performance is minimizing the number of data points scanned, and shown how to efficiently compute the pointwise similarity between the data points and the query points. In order to achieve this goal, we first proposed a gap-based expansion method to reduce number of expansions needed before finalizing a top-k candidate set, leading to fewer scanned data points. However, this approach still suffers from redundant computations between expansion rounds. Therefore, we proposed a novel two-level Threshold Algorithm to separate the scanning of points between spatial and textual similarity in order to remove repetitive scans. We have conducted extensive experiments to verify the efficiency and scalability of the three proposed approaches. Finally, our framework was generalized to handle both spatial-only search, and spatial-textual search, and order-sensitive search over trajectories.

X. ACKNOWLEDGEMENT

This work was partially supported by ARC DP170102726, DP170102231 and National Natural Science Foundation of China (NSFC) 91646204, 61373015, 41301407.

REFERENCES

[1] R. Baeza-Yates, B. Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[2] J. Bao, Y. Zheng, and M. F. Mokbel. Location-based and preference-aware recommendation using sparse geo-social networking data. In *SIGSPATIAL*, pages 199–208, 2012.

[3] 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.

[4] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.

[5] G. Cong, H. Lu, B. C. Ooi, D. Zhang, and M. Zhang. Efficient spatial keyword search in trajectory databases. *ArXiv preprint:1205.2880*, 2012.

[6] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. of Comp. and Sys. Sci.*, 66(4): 614–656, 2003.

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

[8] Z. Li, K. C. Lee, B. Zheng, W.-C. Lee, D. Lee, and X. Wang. IR-Tree: An efficient index for geographic document search. *TKDE*, 23(4):585–599, 2011.

[9] X. Lu, C. Wang, J.-M. Yang, Y. Pang, and L. Zhang. Photo2trip: generating travel routes from geo-tagged photos for trip planning. In *ACMMM*, pages 143–152, 2010.

[10] J. Mackenzie, F. M. Choudhury, and J. S. Culpepper. Efficient location-aware web search. In *ADCS*, pages 1–8, 2015.

[11] S. Qi, P. Bouros, D. Sacharidis, and N. Mamoulis. Efficient point-based trajectory search. In *SSTD*, pages 179–196, 2015.

[12] S. Qi, D. Sacharidis, P. Bouros, and N. Mamoulis. Snapshot and continuous points-based trajectory search. *GeoInformatica*, pages 1–33, 2016.

[13] J. B. Rocha-Junior, A. Vlachou, C. Doulkeridis, and K. Nørvåg. Efficient processing of top-k spatial preference queries. *PVLDB*, 4(2):93–104, 2010.

[14] 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.

[15] L.-A. Tang, Y. Zheng, X. Xie, J. Yuan, X. Yu, and J. Han. Retrieving k-nearest neighboring trajectories by a set of point locations. In *SSTD*, pages 223–241, 2011.

[16] D. Yan, J. Cheng, Z. Zhao, and W. Ng. Efficient location-based search of trajectories with location importance. *Knowledge and Information Systems*, 45(1):215–245, 2015.

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

[18] D. Zhang, C.-Y. Chan, and K.-L. Tan. Processing spatial keyword query as a top-k aggregation query. In *SIGIR*, pages 355–364, 2014.

[19] 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.

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