Searching Activity Trajectories by Exemplar

ZHONG YANG, BOLONG ZHENG, and GUOHUI LI, Huazhong University of Science and Technology
NGUYEN QUOC VIET HUNG, Griffith University
GUANFENG LIU, Macquarie University
KAI ZHENG, University of Electronic Science and Technology of China

The rapid explosion of urban cities has modernized the residents' lives and generated a large amount of data (e.g., human mobility data, traffic data, and geographical data), especially the activity trajectory data that contains spatial and temporal as well as activity information. With these data, urban computing enables to provide better services such as location-based applications for smart cities. Recently, a novel exemplar query paradigm becomes popular that considers a user query as an example of the data of interest, which plays an important role in dealing with the information deluge. In this article, we propose a novel query, called searching activity trajectory by exemplar, where, given an exemplar trajectory τ_q, the goal is to find the top-k trajectories with the smallest distances to τ_q. We first introduce an inverted-index-based algorithm (ILA) using threshold ranking strategy. To further improve the efficiency, we propose a gridtree threshold approach (GTA) to quickly locate candidates and prune unnecessary trajectories. In addition, we extend GTA to support parallel processing. Finally, extensive experiments verify the high efficiency and scalability of the proposed algorithms.

CCS Concepts: • **Information systems → Spatial-temporal systems**; **Information retrieval query processing**; *Retrieval models and ranking*; *Similarity measures*;

Additional Key Words and Phrases: Spatio-temporal trajectory, activity trajectory, trajectorys similarity, exemplar query, query processing

ACM Reference format:
Zhong Yang, Bolong Zheng, Guohui Li, Nguyen Quoc Viet Hung, Guanfeng Liu, and Kai Zheng. 2020. Searching Activity Trajectories by Exemplar. *ACM/IMS Trans. Data Sci.* 1, 3, Article 19 (September 2020), 18 pages.
https://doi.org/10.1145/3379561

Z. Yang and B. Zheng contributed equally to the article.

This research is partially supported by NSFC (Grants No. 61902134, 61572215, 61972069, 61836007, 61832017, and 61532018) and the Fundamental Research Funds for the Central Universities (HUST: Grants No. 2019kfyXKJC021 and 2019kfyXJJS091).

Authors' addresses: Z. Yang, B. Zheng (corresponding author), G. Li, Huazhong University of Science and Technology, Wuhan, China; emails: {zhongyang90, bolongzheng, guohuili}@hust.edu.cn; N. Q. V. Hung, Griffith University, Gold coast, Australia; email: henry.nguyen@griffith.edu.au; G. Liu, Macquarie University, Sydney, Australia; email: guanfeng.liu@mq.edu.au; K. Zheng (corresponding author), University of Electronic Science and Technology of China, Chengdu, China; email: zhengkai@uestc.edu.cn.

2577-3224/2020/09-ART19 $15.00
https://doi.org/10.1145/3379561

1 INTRODUCTION

In recent years, the rapid explosion of urban cities has modernized residents' lives while increasingly generating a large amount of data such as human mobility data, traffic data, geographical data, and trajectory data. Besides, due to the prevalence of spatial web applications on the Internet and mobile techniques, a large amount of activity trajectory data is uploaded and shared by users in online services such as Foursquare and Gowalla, which consists of locations, timestamps, and activity keywords. With these rich information, the activity trajectory data enables urban computing that helps tackle the challenges engendered by the urbanization and provides better services for smart cities. For example, it provides better trip planning and trip recommendation services.

In the existing work, two kinds of trajectory search have been explored. One is the spatial-textual trajectory search [25, 29, 30] that concerns spatial and textual information, and the other is the spatial-temporal trajectory search [5, 7, 23, 24] that concerns spatial and temporal information. Nevertheless, only a few work considers all the three dimensions of spatial, temporal, and textual information at the same time. Although an activity trajectory is modeled as an ordered sequence of locations with activity information [12, 22, 28]; however, this ordered sequence model is not always reasonable. People commonly check-in their locations and activities at the landmarks, Internet celebrity restaurants, and places in their daily life. Different from the vehicle GPS trajectory, the correlation among points within an activity trajectory can be weak. For instance, the spatial and temporal distances between adjacent points can be too large, which leads the activity trajectory to be more like a "group of points" rather than a "sequence." Therefore, the distance measure between activity trajectories requires a different and careful design.

Recently, a novel exemplar query paradigm [18] emerges that considers a user query as an example of the data in which the user is interested. Traditional activity trajectory query assumes that users are aware of the characteristics of activity trajectories and are able to describe them in the query, but the users may not know how to describe the specifications of the activity trajectories in practical scenarios. With the help of exemplar query, the users who are not familiar with activity trajectories can have an exemplar query as starting point related to their real requirements. Having observed these issues, we propose a novel query of searching activity trajectories by exemplar (SATE), which incorporates three dimensions into a coherent definition and concerns more about points and activities.

To this end, a top-k SATE query is defined as follow: *Given a set of activity trajectories, an exemplar trajectory, and an integer k, we aim to retrieve the top-k trajectories ranked by the match distances w.r.t the exemplar trajectory.* In the top-k SATE query, we consider the match of location, time, and activity at the same time. Besides the applications of trip planning and recommendation, our query results can also be applied to some scenarios in social network, such as community detection [16] and partner recommendation. Consider a user query, the top-k SATE returns trajectories of the other users that have nearby locations, close timestamps and the same activities to the query user. These users in the results may be potential trip partners of the user.

To answer a top-k SATE query, we are faced with two main challenges. First, it is difficult to evaluate the activity trajectory distance. Although various distance measures have been proposed in the existing work, only few of them takes spatial, temporal, and activity into consideration at same time. Therefore, a novel and suitable distance measure is required. Second, the brute-force approach that employs a pointwise similarity computation has a quadratic time cost. It is challenging to design an efficient index structure and algorithm to answer the top-k SATE query.

Contribution. The principal contributions of this article can be summarized as follows.

(1) We introduce a novel query called SATE and define a proper designed distance measure to evaluate the distance between activity trajectories.

(2) We first develop an inverted-index-based approach (ILA) by using a heap-based method. Then we propose a GridTree Threshold algorithm (GTA) to answer the top-k SATE query more efficiently. In GTA, we develop an AGI structure to quickly locate the candidate points and utilize a priority queue to sequentially process them. A threshold is used to ensure the process terminates as soon as possible, hence improving the search efficiency. In addition, we propose a parallel version of GTA (PGTA) that utilizes a global threshold to terminate all threads, further improving the performance.
(3) Our experimental evaluation demonstrates the efficiency of proposed algorithms and index structures for processing the top-k SATE query on three datasets. The results show the superiority of GTA in answering the top-k SATE query when compared with ILA.

The remainder of this article is organized as follows. We position our work with respect to the related literature in Section 2 and formulate the problem of searching activity trajectory query by exemplar in Section 3. We first provide an inverted-list index-based approach ILA as baseline in Section 4, and then we propose a gridtree-based approach GTA to answer the top-k SATE query more efficiently in Section 5 and a parallel version PGTA to further improve the performance in Section 6. We report on our experimental observations in Section 7. Finally, we draw a conclusion in Section 8.

2 RELATED WORK

In the past decades, the trajectory search has received significant attentions. There are a large number of trajectory distance measures, such as Euclidean distance, DTW [27], LCSS [24], EDR [5], and ERP [4]. DTW is originally introduced for signal processing, which allows time-shifting in comparison and now is widely applied in time sequence retrieval and trajectory retrieval. LCSS is proposed based on the concept of the longest common subsequence, which is robust to noise by allowing skip of some sample points. EDR is based on the edit distance using a threshold parameter to determine whether two points are matched while considering penalties to gaps, which is also robust against noise and addresses some deficiencies in LCSS. ERP aims to combine the merits of DTW and EDR, by using a constant reference point for computing distance.

The problem of spatial-temporal trajectory search that concerns only spatial and temporal information has been addressed in the existing work [7, 8, 23]. Chen et al. [7] investigate the problem of discovering the most popular route between two locations, they introduce an absorbing Markov chain model to analysis the traveling behaviors of historical traveling. Chen et al. [8] study a new k-BCT trajectory searching problem, in which the query is only a small set of locations while the target is to find the k Best-Connected Trajectories, such that the results best connect the designated locations geographically. Sherkat et al. [23] study the problem of efficiently evaluating similarity queries on histories, where a history is a d-dimensional time series. There are some solutions for time-series and spatial-temporal trajectories where $d \leq 3$, while they examine the problem for larger values of d.

Several recent papers have explored the problem of spatial-textual trajectory search using various measures in the textual domain. Liu et al. [14] infer the future locations of a moving object from its similar trajectories in which the semantic similarity adopted LCSS approach. Zheng et al. [29] explore the problem of approximate keyword search in the context of semantic trajectories. Wang et al. [25] search semantic trajectories in the textual domain with a TF-IDF model while Liu et al. [15] search semantic trajectories in the textual domain with a probabilistic topic model. Meanwhile, a lot of studies appear in spatial-textual search using novel spatial-textual indexes. Felipe et al. [11] propose IR^2-tree, which combines an R-Tree with superimposed text signatures to answer Boolean keyword queries. Cong et al. [9] and Li et al. [13] integrate R-tree and inverted files

(IR-tree) for text retrieval and spatial proximity querying. However, an IR-tree can be inefficient as the retrieve strategy scans a few leaf nodes that attached with many unrelated points that do not match any query keywords [17]. Zheng et al. [30] propose a grid index called GAT, which utilizes both spatial information and query keywords to reduce the search space. Rocha et al. [19] propose a mapping of pairs of data and feature objects to a distance-score space to speed up the performance of top-k spatial preference queries. Chen et al. [3] provide a survey of 12 state-of-the-art spatial-textual indexes of the spatial keyword query performance.

In addition, various novel trajectory queries have been proposed in some work. Cao et al. [1] propose a new type of query, the Location-aware top-k Prestige-based Text retrieval (LkPT) query, which retrieves the top-k spatial web objects ranked according to both prestige-based relevance and location proximity. Shang et al. [20] investigate a novel query type named trajectory search by regions of interest (TSR), which takes a set of regions of interest as a parameter and returns the trajectory with the highest spatial-density correlation to the query regions. The personalized trajectory matching query [21] takes a trajectory with user-specified weights for each sample point in the trajectory as its argument. Cao et al. [2] address the problem of a collective spatial keyword query, in which the group's keywords cover the query's keywords and the results have the lowest inter-object distances. Chen et al. [6] introduce a query of searching trajectories with activities and corresponding ranking information. Wen et al. [26] consider arbitrary text descriptions as keywords of user query, and design a keyword extraction module to classify the POI-related tags for effective matching with query keywords. Zheng et al. [28] propose a personalized route query that includes some clues describing the spatial context between PoIs along the route.

Different from aforementioned studies, only a few of them take all the three dimensions of spatial, temporal and activity information into consideration at the same time. Moreover, between prior work and our work in terms of query and data models, the query keywords, the distance measures and result output are not exactly the same. The hybrid indexes and the corresponding algorithms are also not suitable to our query. So, we propose novel algorithms with customized indexes.

3 PRELIMINARIES

This section formalizes the setting and defines the problem of top-k SATE query. Frequently used notations are summarized in Table 1.

3.1 Settings

Definition 3.1 (Activity Trajectory). An activity trajectory τ is a sequence of activities, i.e., $\tau = \{p_1, \ldots, p_n\}$, where each activity is denoted as $p_i = (l, t, \varphi), i \in [1, n]$, with l being a location, t being a timestamp, and φ being a set of keywords describing the activity.

To avoid clutter, we follow common practice and simply use *trajectory* and *point* to respectively represent activity trajectory and activity when this does not cause ambiguity.

Definition 3.2 (Match). Given two trajectories τ_1 and τ_2, for a point $p_i \in \tau_1$ and a point $q_j \in \tau_2$, we say that p_i is a match of q_j, i.e., $p_i \Leftrightarrow q_j$, if we have $p_i.\varphi_i \cap q_j.\varphi_j \neq \emptyset$. In addition, the point-to-point matching distance between p_i and q_j is computed as follows:

$$\text{Dist}(p_i, q_j) = \alpha \cdot d_S(p_i.v_i, q_j.v_j) + (1 - \alpha) \cdot d_T(p_i.t_i, q_j.t_j), \tag{1}$$

where $d_S(p_i.v_i, q_j.v_j)$ is the Euclidean distance between $p_i.v_i$ and $q_j.v_j$, $d_T(p_i.t_i, q_j.t_j) = |p_i.t_i - q_j.t_j|$ is the temporal distance, and $\alpha \in [0, 1]$ is a user-specified parameter that used to adjust the relative importance of the spatial and temporal distance. It is worth noting that if $p_i.\varphi_i \cap q_j.\varphi_j = \emptyset$, we consider that p_i and q_j are unmatched and non-relevant.

Table 1. Summary of Notations

Notation	Definition
$\tau = \{p_1, \ldots, p_n\}$	An activity trajectory τ
$p_i = (l_i, t_i, \varphi_i)$	A point p_i in an activity trajectory
$p_i \Leftrightarrow q_j$	The point p_i matches the point q_j
$d_S(p_i, q_j)$	The spatial distance between p_i and q_j
$d_T(p_i, q_j)$	The temporal distance between p_i and q_j
$\mathrm{Dist}(\cdot, \cdot)$	The point-to-point, point-to-trajectory, and trajectory-to-trajectory distance
$D_{i,j}$	An abbreviation of the distance $\mathrm{Dist}(p_i, q_j)$
$LBD(\tau_1, \tau_2)$	A lower bound for the distance between τ_1 and τ_2
$LBD_{tmp}(\tau_1, \tau_2)$	A temporary lower bound for the distance between τ_1 and τ_2
$H(\cdot)$	A binary encoding of activities contained in a node or a point.
$N_i(q_n)$	A search region node N_i of a query point q_n
$p_m(q_n)$	A candidate points-pair in the priority queue for a query point q_n
C	The set of candidate trajectories.
R	The set of top-k results.
$R.max$	The maximum value in the result set R.

Without loss of generality, we assume that both $d_S(p_i.v_i, q_j.v_j)$ and $d_T(p_i.t_i, q_j.t_j)$ are normalized by a monotonic increasing function, then we have $\mathrm{Dist}(p_i, q_j) \in [0, 1]$.

Definition 3.3 (Point-to-Trajectory Distance). Given two trajectories τ_1 and τ_2, for a point $p_i \in \tau_1$ and τ_2, the point-to-trajectory distance is computed as follows:

$$\mathrm{Dist}(p_i, \tau_2) = \min_{q_j \in \tau_2, p_i \Leftrightarrow q_j} \mathrm{Dist}(p_i, q_j). \tag{2}$$

The intuition of Equation (2) is that the distance from a point p_i to a trajectory τ_2 is the distance from p_i to its best matched point in τ_2.

Definition 3.4 (Activity Trajectory Distance). Given two trajectories τ_1 and τ_2, the activity trajectory distance is computed as follows:

$$\mathrm{Dist}(\tau_1, \tau_2) = \max\{\max_{p_i \in \tau_1} \mathrm{Dist}(p_i, \tau_2), \max_{q_j \in \tau_2} \mathrm{Dist}(q_j, \tau_1)\}. \tag{3}$$

From Equation (3), we have the following observations:

(i) This function captures the largest point-to-trajectory distance, which implies that the activity trajectory distance bounds the worst case point-to-trajectory distance.
(ii) We assume that the distance function between τ_1 and τ_2 is symmetrical, such that $\mathrm{Dist}(\tau_1, \tau_2) = \mathrm{Dist}(\tau_2, \tau_1)$.
(iii) Let $\tau.\varphi$ be the set of keywords that τ covers. If $\tau_1.\varphi \cap \tau_2.\varphi = \emptyset$, then we consider that τ_1 and τ_2 are unmatched, and $\mathrm{Dist}(\tau_1, \tau_2)$ is not applicable.

3.2 Problem Definition

Given a distance function $\mathrm{Dist}(\tau_1, \tau_2)$ that measures the dissimilarity/similarity between two trajectories τ_1 and τ_2, we study the problem of SATE.

Definition 3.5 (Top-k SATE). Given a trajectory set D, an exemplar trajectory τ_q, and an integer k. The top-k SATE query finds a set R of k trajectories with smallest distance w.r.t. τ_q, i.e., $\forall \tau \in R$, and $\forall \tau' \in D - R$, $\mathrm{Dist}(\tau_q, \tau) < \mathrm{Dist}(\tau_q, \tau')$.

3.3 Computing Activity Trajectory Distance

Matrix Representation (MR). Given τ_1 and τ_2, we employ a matrix representation to interpret the distance computation as shown in Equation (4). Suppose $|\tau_1| = m$ and $|\tau_2| = n$, we obtain a $m \times n$ matrix $\mathbf{D}$, where each element $D_{i,j}$ denotes the point-to-point matching distance between $p_i \in \tau_1$ and $q_j \in \tau_2$. We denote $D_{i,:}$ as the minimum value of all elements in row i, and $D_{:,j}$ as the minimum value in column j, respectively. In other words, $D_{i,:}$ is the point-to-trajectory distance between p_i and τ_2, and $D_{:,j}$ is the distance between q_j and τ_1,

$$\left.\begin{bmatrix} D_{1,1} & D_{1,2} & \cdots & D_{1,n} \\ D_{2,1} & D_{2,2} & \cdots & D_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ D_{m,1} & D_{m,2} & \cdots & D_{m,n} \end{bmatrix} \begin{matrix} \xrightarrow{\min} D_{1,:} \\ \xrightarrow{\min} D_{2,:} \\ \vdots \\ \xrightarrow{\min} D_{m,:} \end{matrix} \right\} \max D_{i,:}, \qquad \begin{matrix} \downarrow \min \downarrow \min \cdots \downarrow \min \\ \underbrace{D_{:,1} \quad D_{:,2} \quad \cdots \quad D_{:,2}}_{\max D_{:,j}} \end{matrix} \tag{4}$$

Therefore, Equation (3) can be reformalized as follows:

$$\mathrm{Dist}(\tau_1, \tau_2) = \max\{\max D_{i,:}, \max D_{:,j}\}. \tag{5}$$

To compute $\mathrm{Dist}(\tau_1, \tau_2)$, a brute-force method is to compute all the elements in $\mathbf{D}$, which costs $O(mn)$ time. To reduce the time cost, we consider the following observation: Suppose we already obtain $D_{i,:}$ of row i, and an element $D_{i',j}$ in row i'. If $D_{i',j}$ is no larger than $D_{i,:}$, then the enumeration on row i' can be skipped. This is because we can easily determine $D_{i',:} \leq D_{i,:}$, since $D_{i',:} \leq D_{i',j}$. That is, $D_{i',:}$ has no chance to contribute the distance between τ_1 and τ_2.

Lemma 3.6. *Given $D_{i,:}$ or $D_{:,j}$, if there exists an element $D_{i',j'}$ with $D_{i',j'} \leq D_{i,:}$ or $D_{i',j'} \leq D_{:,j}$, then we need not to enumerate the elements in row i' or in column j'.*

Computing Exact Distance. Instead of exhaustively enumerating all elements, we propose an ExactDist method to compute the exact distance by skipping unnecessary rows and columns. We first enumerate the elements row by row, and then column by column. From Equation (5), we know that any $D_{i,:}$ or $D_{:,j}$ is a lower bound for the distance between two trajectories. Therefore, we can take an arbitrary $D_{i,:}$ or $D_{:,j}$ as a lower bound, denoted by $LBD(\tau_1, \tau_2)$. Once we complete the scan on a row or column, we update the lower bound if a larger value is obtained. If an element $D_{i',j'}$ is no larger than the current lower bound, then both row i' and column j' can be safely dismissed.

4 INVERTED LIST-BASED ALGORITHM

The inverted list-based algorithm (ILA), as shown in Algorithm 1, builds an inverted index $\mathcal{I}$ for the keywords in the trajectories. Let $\mathcal{I}(w)$ be the inverted list for keyword w, which includes all the trajectories that contain w, and $|\mathcal{I}(w)|$ be the length of $\mathcal{I}(w)$. For simplicity, we assume that the trajectories in each $\mathcal{I}(w)$ are already ordered in ascending order of their IDs. Given an exemplar τ_q, to obtain the candidate trajectories C, the first step is to merge the inverted lists of all the keywords in τ_q. This step can be accomplished by using an existing heap-based method [10], which costs $O(\sum |\mathcal{I}(w)|)$ time, where $w \in \bigcup p.\varphi$ and $p \in \tau_q$. The second step is to estimate the distance between τ_q and each candidate by computing a lower bound distance. The third step simply applies a threshold algorithm (TA), which processes each candidate in ascending order on the lower bound. For the next unprocessed candidate, we exactly compute its distance to τ_q by ExactDist and add it to R if its distance is smaller than the current largest distance in R. The

algorithm terminates when the current largest distance in R is smaller than the lower bound of the next unprocessed candidate.

ALGORITHM 1: InvertedListAlgorithm (ILA)

Input: D: all the trajectories, τ_q: a query exemplar
Output: R: a set of top-k trajectories

```
Initialize R ← ∅;
W ← w ∈ ⋃ p.φ and p ∈ τ_q;
C ← merge I(w) of all w ∈ W;
foreach τ ∈ C do
    Compute LBD(τ_q, τ);
Sort τ ∈ C in ASC by LBD(τ_q, τ);
foreach τ ∈ C do
    if Dist(R_k) ≤ LBD(τ_q, τ) then
        break;
    Dist(τ_q, τ) ← ExactDist();
    Update R if necessary;
return R;
```

Computation Analysis. Compared with the brute-force enumeration method, ILA is more efficient. Although ILA traverses all candidate trajectories and computes *LBDs*, it avoids computing the exact distances between query trajectory and candidate trajectories. In the best case, only k exact distances of the candidate trajectories are required, thus a large amount of computation can be saved; even in the worst case, the method ends after all the exact distances of candidate trajectories are computed, and the time cost equals to that of the brute-force method.

5 GRIDTREE THRESHOLD APPROACH

In ILA, an inverted index is applied to find the candidate trajectories whose activities intersect with that of the query trajectory. However, the candidate trajectories may be retrieved multiple times in different activity posting lists, especially when the number of activities contained in the candidate trajectories is large. Moreover, it suffers from retrieving all candidate trajectories and computing a *LBD* for each of them. Therefore, we propose a GridTree Threshold Approach (GTA) to prune the trajectories that are far away from the query trajectory such that avoids duplicate retrieval. In GTA, we develop an Activity GridTree Index to quickly locate the candidate points that match with the query trajectory points. Meanwhile, we utilize a priority queue to process the candidate points sequentially. To determine when to calculate the match distance between the candidate trajectory and the query trajectory, each candidate trajectory is annotated by using a Scan Flag that records whether it is scanned partially or fully by the query trajectory. The thresholds are used to prune points and determine the early stop of the query processing.

Activity GridTree Index (AGI). A basic GridTree is a tree-based data structure in which each internal node has four children decomposing parent node planar space region into four equal quadrants, and each leaf node contains data corresponding to a specific subregion. In our problem, given a query trajectory τ_q, we aim to find trajectories containing activities that match the query trajectory activities while the distance between τ_q and τ_j is the smallest. To enable the pruning on activities, we construct the Active GridTree as follows.

- **Points Storage.** First, we construct a GridTree T with a depth of d, each node N_i in the tree represents a space region R_{N_i}. Then we insert points of all trajectories into leaf nodes of the

tree: We compare the location (x_p, y_p) of each point p with each child node N_i of the root node, if $(x_p, y_p) \in R_{N_i}$, and then we continue to compare (x_p, y_p) with each child node $N_i{}'$ of the node N_i, iterating this process until node $N_i{}'$ is a leaf node and we store the point in the leaf node.

- **Activity Encoding.** Then, we map activities to a binary code with h bits utilizing an encoding function H. In binary code $H(w)$, one bit is set to 1 to represent activity keyword w. Hence, for a point p the binary code $H(p)$ is the superposition of $H(w)$ of all activity keywords it contains, formally written as $H(p) = \vee_{w \in p.\varphi} H(w)$. Similarly, the binary code of a trajectory $H(\tau)$ is the superposition of $H(p)$ of all $p \in \tau$; and for a node N, if N is a leaf node, then $H(N)$ is the superposition of $H(p)$ of all points p it contains; otherwise, $H(N)$ is the superposition of $H(N')$ of its child nodes N'.

Therefore, a non-zero value of $H(p) \wedge H(q)$ indicates that $p.\varphi \cap q.\varphi \neq \emptyset$, and $H(p) \wedge H(q) = 0$ means points p and q have no activity intersection. Likewise a non-zero value of $H(N) \wedge H(q)$ indicates that there exists points p_i having $p_i.\varphi \cap q.\varphi \neq \emptyset$ stored in the descendant leaf nodes of node N, and $H(N) \wedge H(q) = 0$ means there is no match points for query point q stored in the descendants of node N. Each node contains the information of its space region coordinates and activity binary code $H(N)$, which provides strong support to find match points for a query point q in its nearby regions.

Priority Queue (PQ). In our algorithm, a priority queue is used to process the nodes and points sequentially, in which only one node or one point is accessed at each iteration. Based on AGI, the PQ processing can be divided into two levels:

(1) In the first level, each candidate search region node for each query point q_n, written as $N_i(q_n)$, is inserted into PQ according to the distance from q_n to the region of nodes (as the node region has no temporal information, the distance between a point and the node region is defined as $\text{Dist}(q_n, N_i) = \alpha \cdot \min d_s(q_n, N_i)$). That is, in each iteration of the processing, if the top element of PQ is a non-leaf node, after popping the node, then its childnodes are inserted into PQ according to $\text{Dist}(q_n, N_i)$, if $H(N_i) \wedge H(q_n) \neq 0$.
(2) In the second level, every candidate point for each query point q_n, written as $p_m(q_n)$, is inserted into PQ according to the distance between q_n and p_m. That is, in each iteration of the processing, if the top element of PQ is a leaf node, then the points p_m contained in it are inserted into PQ according to $\text{Dist}(q_n, p_m)$ if $H(p_m) \wedge H(q_n) \neq 0$.

An example of a simplified priority queue is shown in Figure 1 to help understanding. We consider the distance between a point and a node is zero when the point locates in the node, for example, q_1 and $N_5(q_1)$. First in level 1 of Figure 1, assume N_2, N_3, N_4, N_5 are childnodes of N_0, after $N_0(q_1)$ is popped from PQ, these nodes are inserted into PQ in order of $\{N_5(q_1)$, $N_3(q_1)$, $N_4(q_1)$, $N_2(q_1)\}$ as $\text{Dist}(q_1, N_5) < \text{Dist}(q_1, N_3) < \text{Dist}(q_1, N_4) < \text{Dist}(q_1, N_2)$. Whereafter, assume nodes N_6, N_7, N_8, N_9 are childnodes of N_1, after $N_1(q_2)$ and $N_1(q_3)$ are popped from PQ, nodes around q_2 and q_3 will be inserted into PQ. Assume that $\text{Dist}(q_1, N_5) \leq \text{Dist}(q_2, N_8) \leq \text{Dist}(q_3, N_6) \leq \text{Dist}(q_1, N_3) \leq \text{Dist}(q_2, N_9) \leq \text{Dist}(q_1, N_4) \leq \cdots \leq \text{Dist}(q_3, N_9)$, then the order of nodes in PQ of level 1 is as shown in Figure 1. Next in level 2, assume the top few nodes $N_5(q_1)$, $N_8(q_2)$, $N_6(q_3)$ in PQ are leafnodes, points are stored in these nodes. After $N_5(q_1)$ is popped from PQ, points in $N_5(q_1)$ are inserted into PQ in order of their distances to q_1, and after $N_8(q_2)$ is popped from PQ, points in $N_8(q_2)$ are inserted into PQ in order of their distances to q_2. Then the order of PQ in level 2 is as shown in Figure 1 if we assume $\text{Dist}(q_1, p_2) \leq \text{Dist}(q_2, p_6) \leq \text{Dist}(q_2, p_4) \leq \text{Dist}(q_1, p_1) \leq \text{Dist}(q_1, p_3) \leq \text{Dist}(q_2, p_5) \leq \cdots \leq \text{Dist}(q_3, N_9)$. The algorithm continues accessing

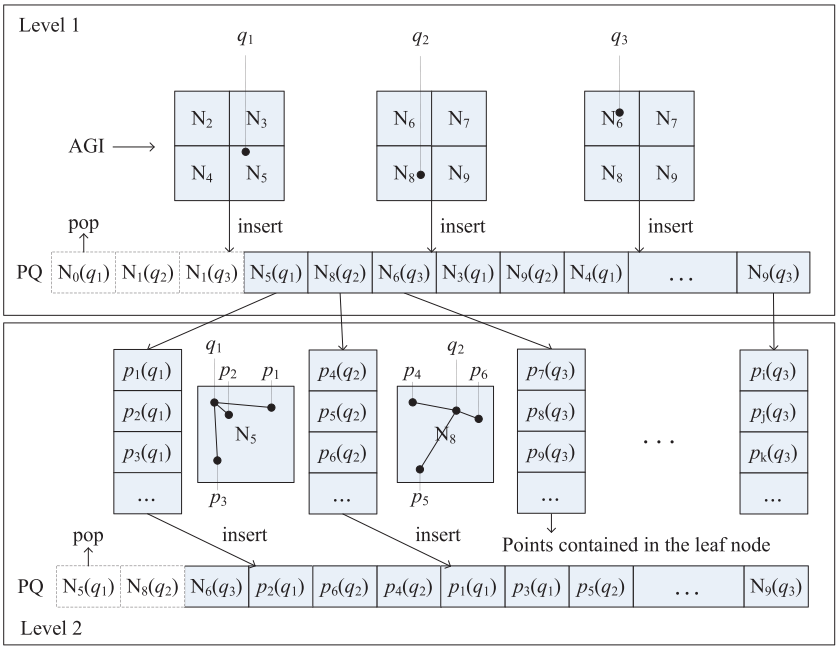


Fig. 1. An example of the priority queue.

points in PQ until there is no item in it or until a particular condition is encountered. The particular condition is introduced later after introducing Scan Flag in this section.

Scan Flag. Since the algorithm accesses one point at each iteration, the points in each trajectory are accessed discretely. To determine when the distance between the candidate trajectory and the query trajectory should be calculated, a notation is required to identify whether all the points in a candidate trajectory have been accessed by the query trajectory.

For a candidate trajectory τ:

- **None-Scanned:** If no points in τ has been accessed by any $q_i \in \tau_q$, then we call τ none-scanned;
- **Pre-Scanned:** If a portion of points in τ has been accessed by q_i, then we call τ pre-scanned by q_i;
- **Partial-Scanned:** If all points in τ has been accessed by q_i, then we call τ partial-scanned by q_i;
- **Full-Scanned:** If τ is partial-scanned by all $q_i \in \tau_q$, then we call τ full-scanned by τ_q.

It can be observed in the matrix representation of Equation (4) that, once τ is partial-scanned by q_i, we achieve a temporary lower bound $LBD_{tmp}(\tau_q, \tau) = D_{i,:} = \min D_{i,j}$, and once τ is full-scanned by τ_q, we achieve the lower bound $LBD(\tau_q, \tau) = \max D_{i,:}$. A Scan Flag is similar to a matrix representation, where each element in it, noted as SC_{ij} (similarly to the elements D_{ij} in matrix representation), is not a distance but a bit of one or zero to indicate whether the candidate point p_j is scanned by the query point q_i or not. All the elements are set to zero, which is the initial status of Scan Flag that represents none-scanned. All the elements in row i are one representing partial-scanned by q_i. All elements in the Scan Flag are one indicating full-scanned by τ_q.

Transformation of Scan Flag. In each iteration, we transform the element SC_{ij} in Scan Flag from zero to one if the corresponding candidate point p_j is accessed by the query point q_i. It seems

that we need to access all the points in a candidate trajectory to transform the Scan Flag elements to ones and we must spend a lot of space to store the Scan Flag. But in practice, the transformation of Scan Flag elements does not require all points to be actually accessed, and a strategy is introduced in the following to quickly transform the Scan Flag to partial-scanned and significantly reduce the space cost of the Scan Flag.

Lemma 5.1. *For a query point q_i and a candidate trajectory τ_j, if we have not accessed any points-pairs $p_j(q_i)$, where $p_j \in \tau_j$ before, after we access the first points-pair $p_j(q_i)$ for q_i, the distance between these two points* $\text{Dist}(q_i, p_j)$ *is the point-to-trajectory distance between q_i and τ_j and it is a temporary lower bound $LBD_{tmp}(\tau_q, \tau_j)$ between τ_q and τ_j. Thus, τ_j can be labeled as partial-scanned by q_i. And other points-pairs $p_j(q_i)$ for q_i can be skipped in the priority queue.*

Proof. As the constraint of the priority queue, we have $\text{Dist}(q_i, p_j) \leq \text{Dist}(q_i, p'_j)$ if the points-pair $p_j(q_i)$ is in the front of the points-pair $p'_j(q_i)$ in the queue. Thus, the first accessed points-pair $p_j(q_i)$ for q_i has the minimal point-to-point distance between q_i and τ_j, i.e., $\text{Dist}(q_i, p_j) = min\{D_{i,j}\} = D_{i,:}$, the distance is also a temporary lower bound $LBD_{tmp}(\tau_q, \tau_j)$ between τ_q and τ_j. So after we access the first points-pair $p_j(q_i)$ for q_i, the distance $\text{Dist}(q_i, p_j)$ is the point-to-trajectory distance from q_i to τ_j, and τ_j can be labeled as partial-scanned by q_i. □

With the above lemma, the Scan Flag can be reduced to a matrix with only one column, which significantly saves the space consumption.

Termination Threshold. Along with the iteration, the trajectories' Scan Flag constantly updates. Whenever a candidate trajectory τ_j is partial-scanned by a $q_i \in \tau_q$, we obtain a $D_{i:}$, and we update the temporary lower bound $LBD_{tmp}(\tau_q, \tau_j)$ to the larger $D_{i:}$. After τ_j is full-scanned, we get the $LBD_{tmp}(\tau_q, \tau_j) = LBD(\tau_q, \tau_j) = \max D_{i:}$. Then the trajectory-to-trajectory match distance is computed and inserted into R. R always saves the minimum k results. The iteration continues processing until there is no item in PQ or until the following condition is encountered.

Lemma 5.2. *After R reserves k temporary results, the current $R.max$ is a upper bound of the final results, if the current $R.max < \text{Dist}(q_c, p_c)$, where $p_c(q_c)$ is current points-pair in the queue, then the query processing terminates, and all the none-scanned trajectories can be pruned.*

Proof. It is easy to learn that the final $R.max \leq$ any approximate $R.max$, so the current $R.max$ is a upper bound of the final results. For every none-scanned trajectory τ_{ns}, its points-pairs are all behind the current top points-pair in PQ, i.e., $\forall q_i \in \tau_q$ and $\forall p_j \in \tau_{ns}$, $\exists \text{Dist}(q_c, p_c) < \text{Dist}(q_i, p_j)$. If the current $R.max < \text{Dist}(q_c, p_c)$, then $R.max < \text{Dist}(q_c, p_c) < \text{Dist}(q_i, p_j)$. As we have $\text{Dist}(q_i, p_j) \leq \text{Dist}(\tau_q, \tau_{ns})$. Thus, the final $R.max \leq$ current $R.max < \text{Dist}(\tau_q, \tau_{ns})$, which means τ_{ns} cannot be one of the final results and can be pruned. □

After the iteration terminates, all the non-scanned trajectories are pruned, the remaining pre-scanned and partial-scanned trajectories and the current R require a further verification to determine the final top-k results. Fortunately, for each partial-scanned trajectories we have achieved a lower bound $LBD_{tmp}(\tau_q, \tau)$, and for all pre-scanned trajectories, we can transform them into partial-scanned trajectories according to Lemma 5.1. Therefore, with all these LBDs, we utilize the ILA method in Algorithm 1 to achieve the final results.

GridTree Threshold Approach (GTA). Combining AGI, PQ, Scan Flag, and the termination strategy, we propose our GTA, shown in Algorithm 2. After initializing GridTree and PQ, the root node of GridTree is inserted into PQ (lines 1–5). The query processing can be divided into two levels. In the first level, nodes are processed (lines 8–13), child nodes or points contained in them are inserted into PQ. In the second level, points are processed (lines 14–31), the encoding function $H()$ is employed to distinguish whether points-pair has activity intersection, then strategies are

ALGORITHM 2: GridTree Threshold Algorithm

Input: τ_q: a query exemplar
Output: R: a set of top-k trajectories

```
1  Initialize GridTree T;
2  Initialize Priority Queue PQ;
3  Initialize C ← ∅;
4  Initialize R ← ∅;
5  foreach q_n ∈ τ_q do
6      Insert T.root(q_n) into PQ
7  while PQ.size > 0 do
8      if PQ.top is a node then
9          N(q_n) ← PQ.top;
10         if N(q_n) is non-leaf node then
11             Insert all childnodes cN_i(q_n) into PQ if H(cN_i) ∧ H(q_n) ≠ 0;
12         else
13             Insert all p_m(q_n) ∈ N(q_n) into PQ if H(p_m) ∧ H(q_n) ≠ 0;
14     if PQ.top is a point then
15         p_m(q_n) ← PQ.top;
16         C ← τ where p_m ∈ τ;
17         if all p_m ∈ τ is accessed by q_n then
18             τ is partial-scanned by q_n;
19             Update LBD(τ_q, τ) if necessary;
20         if τ is partial-scanned by all q_n ∈ τ_q then
21             τ is full-scanned by τ_q;
22             Update LBD(τ_q, τ) if necessary;
23         if τ is full-scanned then
24             Dist(τ_q, τ) ← ExactDist();
25             if |R| < k then
26                 Insert τ into R;
27             else
28                 Update R if necessary;
29                 if R.max < current Dist(q_n, p_m) then
30                     break;
31 Same as line 6–11 in Algorithm 1;
32 return R;
```

utilized to update Scan Flag and prune points (lines 14–24), afterwards Lemma 5.2 terminates the iteration (lines 30 and 31), and at last Algorithm 1 is applied to determine the final results (line 32). Compared to ILA, our GTA is more efficient. Though the time complexity of GTA is almost the same to that of ILA, the GridTree index and pruning strategies remarkably save the computation, which is also proved in the experiments.

6 PARALLEL APPROACH

Through in-depth observation, we find that candidate points of each query point are processed discretely and independently before the termination condition is encountered during the iteration.

ALGORITHM 3: ParallelGTA algorithm

Input: τ_q: a query exemplar
Output: R: a set of top-k trajectories
1 Initialize GridTree T;
2 Initialize $R \leftarrow \emptyset$;
3 **foreach** $q_i \in \tau_q$ **do**
4 Initialize Priority Queue $PQ(q_i)$;
5 Find an approximate R among λk nearby trajectories;
6 **foreach** $PQ(q_i)$ **do**
7 Same as lines 7–22 in Algorithm 2 in parallel;
8 **if** *approximate* $R.max$ < *current* $\text{Dist}(q_i, p_j)$ **then**
9 End thread;
10 **if** *all threads end* **then**
11 **foreach** $\tau \in C$ **do**
12 **if** τ *is full-scanned* **then**
13 $\text{Dist}(\tau_q, \tau) \leftarrow \text{ExactDist}()$;
14 Update R if necessary;
15 Same as lines 6–11 in Algorithm 1;
16 return R;

Therefore, the query processing can be implemented in parallel according to each query point q_i. Based on this observation, we propose a parallel approach, Parallel GridTree Threshold Approach (PGTA), for a further improvement of k-SATE query.

Global Termination Threshold. It is worth noting that in the parallel approach, a Global Termination Threshold that is an upper bound of result set R is required to terminate all parallel processes. Different from GTA, we do not have to wait for the serial process producing the termination threshold $R.max$ spontaneously after R reserves enough results, which can be seen in lines 26 and 27 in Algorithm 2. In that case, the parallel processes wait on each other, which makes implementing the query in parallel meaningless. Therefore, a Global Termination Threshold should be determined beforehand. According to lemma 5.2, any $R.max$ can be an upper bound of R. So if we find an approximate R in advance, the $R.max$ can play the role of the Global Termination Threshold. However, an arbitrary upper bound is not feasible if it is too loose to prune efficiently. Hence, we consider to acquire the top-k results of λk trajectories nearby the query trajectory as an approximate R by using ILA. In this approach, the larger the value of λ is, the closer the approximate R is to the final results, and the higher the pruning power is. But if the value of λ is too large, the computation cost of finding the approximate $R.max$ will be expensive. In summary, when the value of λ is small, the pruning power is low while the computation cost is low. However, the pruning power is high while the computation cost increases.

Parallel GridTree Threshold Approach (PGTA). In the PGTA, as shown in Algorithm 3, PQs are prepared for each query points (lines 3 and 4). After an approximate $R.max$ is found among λk candidate trajectories near the query trajectory, PQs are processed in parallel (lines 6–9). The approximate $R.max$ is leveraged to terminate all the processes (lines 8 and 9). At last, ILA is applied to determine the final results. Compared to GTA, the parallel approach improves the efficiency explicitly. Although finding the Global Termination Threshold incurs additional computation costs, points of the λk trajectories can be skipped in the PQ procedure. In general, PGTA is more efficient than GTA, which is also proved in experiment results.

Table 2. Summary of Datasets

Dataset	$\|D\|$	$\|P\|$	$avg\|D.P\|$	$\|\Phi\|$	$avg\|P.\varphi\|$
CA	46000	383125	8.33	40	2.12
NY	69744	239547	3.43	661	7.46
ST	99204	981747	9.90	1033	4.37

Table 3. Choices of Parameters

Parameter	Range	Default Value
d	4,5,6,7,8,9,10,11,12	10
k	10,20,50,100,200	50
$\|q\|$	4,6,8,10,12	8
α	0.1,0.3,0.5,0.7,0.9	0.5
λ	10	10

7 EXPERIMENTAL STUDY

In this section, we report our experiment results and provide efficiency evaluations and analysis from our experimental study.

7.1 Experiment Setup

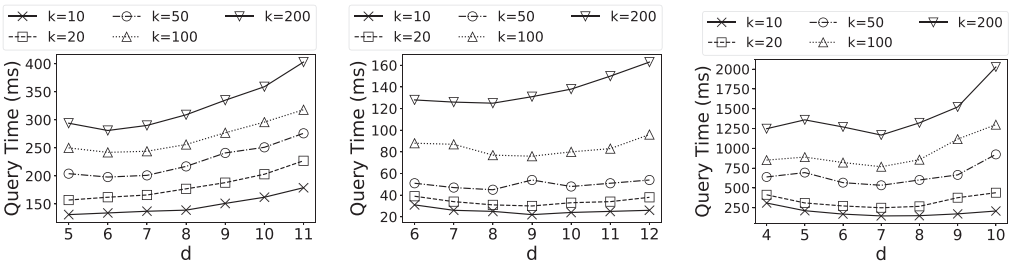
Datasets. We conduct our experiments on two real-world activity trajectory datasets from Foursquare, i.e., California (CA) and New York (NY), and a synthetic dataset (ST) generated by a normal distribution. Table 2 summarizes the number of trajectories (D), the total number of trajectory points (P), the average length of trajectory ($avg|D.P|$), the number of keywords in the trajectories ($|\Phi|$), and the average number of keywords in each trajectory point ($avg|P.\varphi|$). Our experiments are conducted on a PC with a 3.90-GHz CPU and 8-GB RAM using Windows 10 and implemented in C++ using 64 bit addressing.

Parameters. The parameter settings of the experiments are presented in Table 3. d is the depth of GridTree, k is the value of top-k, $|q|$ is the number of points in the query trajectory, α is the coefficient defined in Equation (1), and λ is the approximate search region coefficient defined in PGTA.

Algorithms for Evaluation. We compare the following approaches to process SATE query over three datasets. (1) ILA: The baseline approach using an inverted list-based index to organize trajectory data proposed in Section 4. (2) GTA: The approach using an activity gridtree index and a priority queue to access trajectory points proposed in Section 5. (3) PGTA: The parallel version of GTA proposed in Section 6. We use query time of above algorithms to evaluate computational efficiency. We explore the effect of different parameters in Table 3 on overall performance.

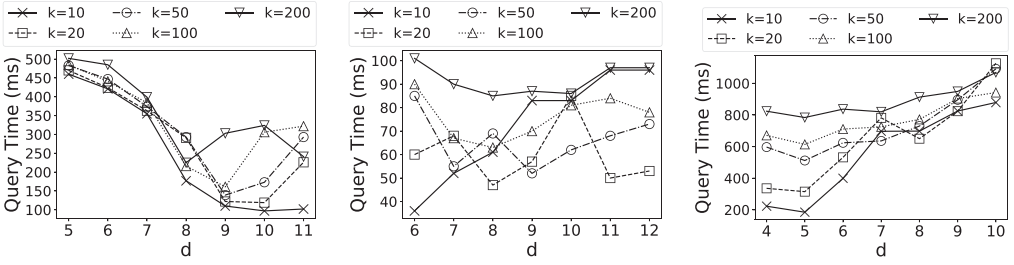
7.2 Efficiency Evaluation

Efficiency of d. In the first set of experiments, we study the effect of different depth of the AGI. We vary parameter d to explore the efficiency of the algorithms, and we choose the results with a default value of α, $|q|$ and different k, shown in Figure 2 and Figure 3, to illustrate the efficiency and the best choice of d. Because ILA has no tree structure, so it is not displayed in these two figures. We can observe in these two figures that the query time curves of GTA and PGTA over three datasets are almost concave. That means, with the increase of d, the query time first decreases to a



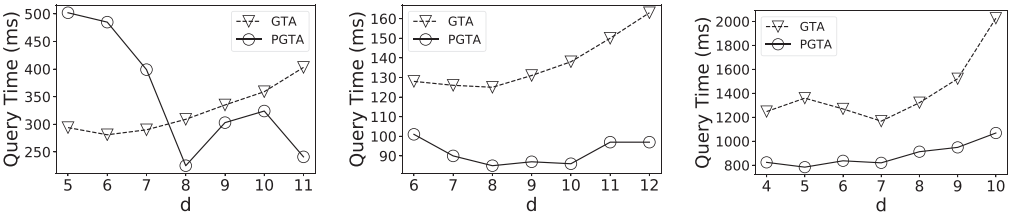
(a) CA dataset (b) NY dataset (c) ST dataset

Fig. 2. Query time of GTA with varying d and k, $\alpha = 0.5$, and $|q| = 8$ on the CA (left), NY (middle), and ST (right) datasets.



(a) CA dataset (b) NY dataset (c) ST dataset

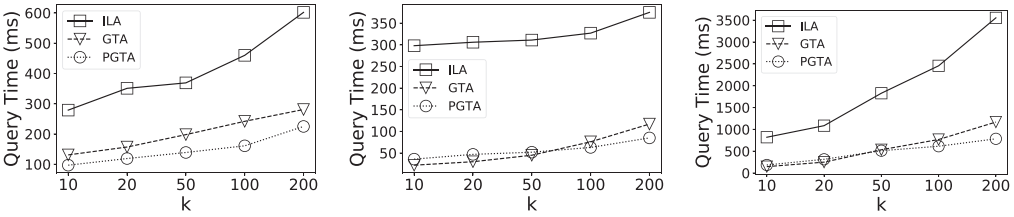
Fig. 3. Query time of PGTA with varying d and k, $\alpha = 0.5$, and $|q| = 8$ on the CA (left), NY (middle), and ST (right) datasets.



(a) CA dataset (b) NY dataset (c) ST dataset

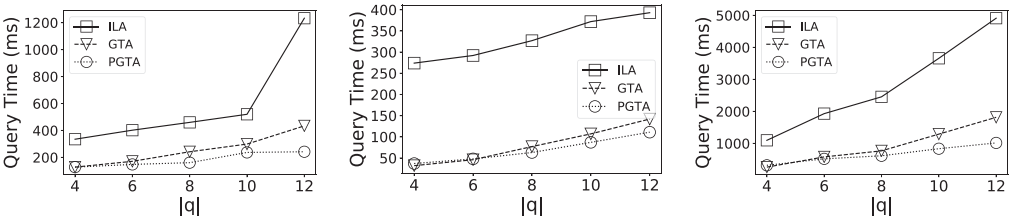
Fig. 4. Comparison of GTA and PGTA on the query time with $\alpha = 0.5$, $k = 200$, and $|q| = 8$ on the CA (left), NY (middle), and ST (right) datasets.

trough value and then increases. For example, in Figure 2 the query time reaches the trough when $d = 6$ on CA, $d = 8$ or 9 on NY, and $d = 7$ on ST. For PGTA in Figure 3, the query time reaches the trough when $d = 8$ or 9 on CA and $d = 5$ on ST, while the query time on NY reaches the trough around $d = 8$ to 10, which is not that obvious. The concavity of the query time curve we considered is associated with the spatial density of points in a leafnode. When the value of d is too small, the number of points in a leafnode is large, thus the efficiency of retrieval is low. On the contrary, when the value of d is too large, the number of points in a leafnode is small, thus the retrieval time of one leafnode gets shorter but more leafnodes to be accessed to find the results. In addition, the comparison on query time of GTA and PGTA is shown in Figure 4. As shown, in the most cases the query time of PGTA is less than that of GTA, and the minimum query time of PGTA is always



(a) CA dataset (b) NY dataset (c) ST dataset

Fig. 5. Comparison of ILA, GTA, and PGTA on the query time with $\alpha = 0.5$, $|q| = 8$, and different k on the CA (left), NY (middle), and ST (right) datasets.



(a) CA dataset (b) NY dataset (c) ST dataset

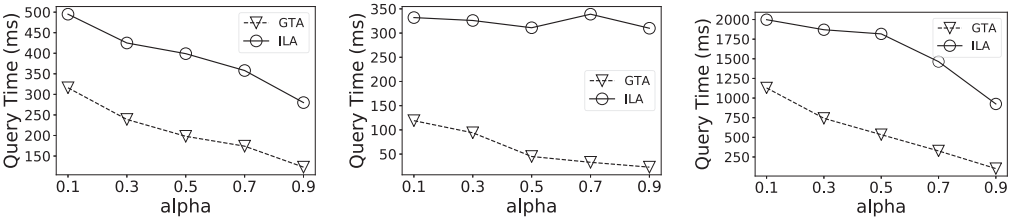
Fig. 6. Comparison of ILA, GTA, and PGTA on the query time with $\alpha = 0.5$, $k = 100$, and different $|q|$ on the CA (left), NY (middle), and ST (right) datasets.

less than that of GTA on all three datasets. It demonstrates that PGTA always achieves a better performance than GTA does.

Efficiency of k. Next, we study the effect of the intended number of results k, the query time with a best choice of d, a default α value and a default $|q|$ value on the three datasets are presented in Figure 5. We find the influence of k to the query time is not heavily, the query time of all approaches on each dataset increases slightly when the value of k is enlarged. This is expected that more trajectories need to be accessed as k increases. As shown in Figure 5, GTA and PGTA outperform ILA significantly, particularly on NY and ST datasets, and they are close to one order of magnitude faster than ILA. In addition, PGTA is always faster than GTA on CA. On NY and ST, it performs almost the same with GTA when the value of k is small and performs better than GTA when the value of k goes up.

Efficiency of $|q|$. Figure 6 shows the effect of $|q|$ with a default α value and $k = 100$ over the three datasets. As shown, the query time of ILA and GTA increases linearly while the query time of PGTA increases slower when the value of $|q|$ goes up. This is expected that the computation of the query is linearly increases with the number of query points. And the parallel method PGTA is more compatible to the increasing of $|q|$.

Efficiency of α. We vary the coefficient parameter α to compare the efficiency of GTA and ILA with a default value of k and $|q|$ over three datasets. The results are shown in Figure 7. The query time of GTA and ILA reduces synchronously when α increases. That means, if users want to specify more weight on spatial similarity, the efficiency improves. And if users want a spatial-only query, setting $\alpha = 1$, then our approach is still feasible. However, if users want to specify more weight on temporal similarity, then the efficiency of our approach reduces, but it is still better than the baseline.



(a) CA dataset (b) NY dataset (c) ST dataset

Fig. 7. Comparison of ILA and GTA on the query time with $k = 50$, $|q| = 8$, and different α on the CA (left), NY (middle), and ST (right) datasets.

Efficiency of λ. We use a fixed value of λ for the experiments shown in Table 3. λ is the search region coefficient in PGTA to find an approximate $R.max$ as the global upper bound. If λ is too small, then the pruning efficiency is low; if λ is too large, then the time consumption of finding the approximate $R.max$ is expensive. So a suitable value of λ improves the performance of PGTA. Henceforth, we set $\lambda = 10$.

Analysis behind Efficiency. Through all the comparison of experiments, we observe that our algorithms GTA and PGTA preform well and stably on ST and perform noticeably well except for a few fluctuations on NY and perform not that outstanding on CA compared to other two datasets. On ST, PGTA always outperforms the GTA, and it is less sensitive than GTA to the value of $|k|$ and $|q|$. On NY, the query times of PGTA and GTA are both low, less than 100 ms in the most cases, and reach 20 ms in the best case. Sometimes GTA even performs better than PGTA, although a few milliseconds. On CA, PGTA is always faster than GTA, but they are only 2× to 3× faster than ILA. Generally speaking, PGTA always performs well while GTA is more sensitive to dataset. It seems that there is no relationship between the performance of our algorithms and the statistic of datasets shown in Table 2. ST has the maximum number of trajectories, points, keywords, and average points of each trajectory, and NY has the maximum number of average keywords of each point. We consider the difference in performance of our algorithms is related to two more aspects: (1) the data distribution and (2) the final upper bound when query processing terminates. ST is generated by a normal distribution, and the distribution of CA is more concentrated than that of NY. The final upper bound is determined in advance in PGTA, while it is produced spontaneously in the PQ process, which is unpredictable in GTA. To summarize, compared to ILA, GTA and PGTA have better performance with less query time in all settings.

8 CONCLUSION

In this article, we study a novel problem of searching trajectories by exemplar with activities and spatial and temporal information. To process the problem efficiently, we first propose an inverted-index-based approach ILA, which traverses all the candidates to achieve a lower bound of each candidate. This approach suffers from redundant computation. Then, we develop an algorithm named GTA to compute the ranking of trajectory matching more efficiently. In this method, an activity gridtree index is utilized to organize trajectories data and prune the search space, a priority queue is used to process points sequentially. Furthermore, a parallel version PGTA is provided to improve the performance. Our algorithms are able to handle both spatial-only search and spatial-temporal search. The results of experimental studies show both the proposed gridtree-based algorithms are capable of answering the top-k SATE query efficiently, while PGTA achieves higher efficiency and scalability.

REFERENCES

[1] Xin Cao, Gao Cong, and Christian S. Jensen. 2010. Retrieving Top-k prestige-based relevant spatial web objects. *Proc. VLDB* 3, 1 (2010), 373–384. DOI: https://doi.org/10.14778/1920841.1920891

[2] Xin Cao, Gao Cong, Christian S. Jensen, and Beng Chin Ooi. 2011. Collective spatial keyword querying. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'11)*. ACM, 373–384. DOI: https://doi.org/10.1145/1989323.1989363

[3] Lisi Chen, Gao Cong, Christian S. Jensen, and Dingming Wu. 2013. Spatial keyword query processing: An experimental evaluation. *Proc. VLDB* 6, 3 (2013), 217–228. DOI: https://doi.org/10.14778/2535569.2448955

[4] Lei Chen and Raymond T. Ng. 2004. On the marriage of lp-norms and edit distance. In *Proceedings of the 30th International Conference on Very Large Data Bases*. Morgan Kaufmann, 792–803.

[5] Lei Chen, M. Tamer Özsu, and Vincent Oria. 2005. Robust and fast similarity search for moving object trajectories. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, 491–502. DOI: https://doi.org/10.1145/1066157.1066213

[6] Wei Chen, Lei Zhao, Jiajie Xu, Kai Zheng, and Xiaofang Zhou. 2014. Ranking based activity trajectory search. In *Proceedings of the 15th International Conference on Web Information Systems Engineering (WISE'14)*, Lecture Notes in Computer Science, Vol. 8786. Springer, 170–185. DOI: https://doi.org/10.1007/978-3-319-11749-2_14

[7] Zaiben Chen, Heng Tao Shen, and Xiaofang Zhou. 2011. Discovering popular routes from trajectories. In *Proceedings of the 27th International Conference on Data Engineering (ICDE'11)*. IEEE Computer Society, 900–911. DOI: https://doi.org/10.1109/ICDE.2011.5767890

[8] Zaiben Chen, Heng Tao Shen, Xiaofang Zhou, Yu Zheng, and Xing Xie. 2010. Searching trajectories by locations: An efficiency study. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'10)*. ACM, 255–266. DOI: https://doi.org/10.1145/1807167.1807197

[9] Gao Cong, Christian S. Jensen, and Dingming Wu. 2009. Efficient retrieval of the Top-k most relevant spatial web objects. *Proc. VLDB* 2, 1 (2009), 337–348. DOI: https://doi.org/10.14778/1687627.1687666

[10] Dong Deng, Yufei Tao, and Guoliang Li. 2018. Overlap set similarity joins with theoretical guarantees. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD'18)*. ACM, 905–920. DOI: https://doi.org/10.1145/3183713.3183748

[11] Ian De Felipe, Vagelis Hristidis, and Naphtali Rishe. 2008. Keyword search on spatial databases. In *Proceedings of the 24th International Conference on Data Engineering (ICDE'08)*. IEEE Computer Society, 656–665. DOI: https://doi.org/10.1109/ICDE.2008.4497474

[12] Kaiyang Guo, Rong-Hua Li, Shaojie Qiao, Zhenjun Li, Weipeng Zhang, and Minhua Lu. 2017. Efficient order-sensitive activity trajectory search. In *Proceedings of the 18th International Conference on Web Information Systems Engineering (WISE'17)* Lecture Notes in Computer Science, Vol. 10569. Springer, 391–405. DOI: https://doi.org/10.1007/978-3-319-68783-4_27

[13] Zhisheng Li, Ken C. K. Lee, Baihua Zheng, Wang-Chien Lee, Dik Lun Lee, and Xufa Wang. 2011. IR-tree: An efficient index for geographic document search. *IEEE Trans. Knowl. Data Eng.* 23, 4 (2011), 585–599. DOI: https://doi.org/10.1109/TKDE.2010.149

[14] Hechen Liu and Markus Schneider. 2012. Similarity measurement of moving object trajectories. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on GeoStreaming (IWGS@SIGSPATIAL'12)*. ACM, 19–22. DOI: https://doi.org/10.1145/2442968.2442971

[15] Huiwen Liu, Jiajie Xu, Kai Zheng, Chengfei Liu, Lan Du, and Xian Wu. 2017. Semantic-aware query processing for activity trajectories. In *Proceedings of the 10th ACM International Conference on Web Search and Data Mining (WSDM'17)*. ACM, 283–292. DOI: https://doi.org/10.1145/3018661.3018678

[16] Siyuan Liu and Shuhui Wang. 2017. Trajectory community discovery and recommendation by multi-source diffusion modeling. *IEEE Trans. Knowl. Data Eng.* 29, 4 (2017), 898–911. DOI: https://doi.org/10.1109/TKDE.2016.2637898

[17] Joel Mackenzie, Farhana Murtaza Choudhury, and J. Shane Culpepper. 2015. Efficient location-aware web search. In *Proceedings of the 20th Australasian Document Computing Symposium (ADCS'15)*. 4:1–4:8. DOI: https://doi.org/10.1145/2838931.2838933

[18] Davide Mottin, Matteo Lissandrini, Yannis Velegrakis, and Themis Palpanas. 2016. Exemplar queries: A new way of searching. *VLDB J.* 25, 6 (2016), 741–765. DOI: https://doi.org/10.1007/s00778-016-0429-2

[19] João B. Rocha-Junior, Akrivi Vlachou, Christos Doulkeridis, and Kjetil Nørvåg. 2010. Efficient processing of Top-k spatial preference queries. *Proc. VLDB* 4, 2 (2010), 93–104. DOI: https://doi.org/10.14778/1921071.1921076

[20] Shuo Shang, Lisi Chen, Christian S. Jensen, Ji-Rong Wen, and Panos Kalnis. 2017. Searching trajectories by regions of interest. *IEEE Trans. Knowl. Data Eng.* 29, 7 (2017), 1549–1562. DOI: https://doi.org/10.1109/TKDE.2017.2685504

[21] Shuo Shang, Ruogu Ding, Kai Zheng, Christian S. Jensen, Panos Kalnis, and Xiaofang Zhou. 2014. Personalized trajectory matching in spatial networks. *VLDB J.* 23, 3 (2014), 449–468. DOI: https://doi.org/10.1007/s00778-013-0331-0

[22] Mehdi Sharifzadeh, Mohammad R. Kolahdouzan, and Cyrus Shahabi. 2008. The optimal sequenced route query. *VLDB J.* 17, 4 (2008), 765–787. DOI: https://doi.org/10.1007/s00778-006-0038-6

[23] Reza Sherkat and Davood Rafiei. 2008. On efficiently searching trajectories and archival data for historical similarities. *Proc. VLDB* 1, 1 (2008), 896–908. DOI: https://doi.org/10.14778/1453856.1453953

[24] Michail Vlachos, Dimitrios Gunopulos, and George Kollios. 2002. Discovering similar multidimensional trajectories. In *Proceedings of the 18th International Conference on Data Engineering*, Rakesh Agrawal and Klaus R. Dittrich (Eds.). IEEE Computer Society, 673–684. DOI: https://doi.org/10.1109/ICDE.2002.994784

[25] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, Timos Sellis, Mark Sanderson, and Xiaolin Qin. 2017. Answering Top-k exemplar trajectory queries. In *Proceedings of the 33rd IEEE International Conference on Data Engineering (ICDE'17)*. IEEE Computer Society, 597–608. DOI: https://doi.org/10.1109/ICDE.2017.114

[26] Yu Ting Wen, Jinyoung Yeo, Wen-Chih Peng, and Seung-won Hwang. 2017. Efficient keyword-aware representative travel route recommendation. *IEEE Trans. Knowl. Data Eng.* 29, 8 (2017), 1639–1652. DOI: https://doi.org/10.1109/TKDE.2017.2690421

[27] Byoung-Kee Yi, H. V. Jagadish, and Christos Faloutsos. 1998. Efficient retrieval of similar time sequences under time warping. In *Proceedings of the 14th International Conference on Data Engineering*. IEEE Computer Society, 201–208. DOI: https://doi.org/10.1109/ICDE.1998.655778

[28] Bolong Zheng, Han Su, Wen Hua, Kai Zheng, Xiaofang Zhou, and Guohui Li. 2017. Efficient clue-based route search on road networks. *IEEE Trans. Knowl. Data Eng.* 29, 9 (2017), 1846–1859. DOI: https://doi.org/10.1109/TKDE.2017.2703848

[29] Bolong Zheng, Nicholas Jing Yuan, Kai Zheng, Xing Xie, Shazia Wasim Sadiq, and Xiaofang Zhou. 2015. Approximate keyword search in semantic trajectory database. In *Proceedings of the 31st IEEE International Conference on Data Engineering (ICDE'15)*. IEEE Computer Society, 975–986. DOI: https://doi.org/10.1109/ICDE.2015.7113349

[30] Kai Zheng, Shuo Shang, Nicholas Jing Yuan, and Yi Yang. 2013. Towards efficient search for activity trajectories. In *Proceedings of the 29th IEEE International Conference on Data Engineering (ICDE'13)*. IEEE Computer Society, 230–241. DOI: https://doi.org/10.1109/ICDE.2013.6544828

Received June 2019; revised December 2019; accepted December 2019