The following publication X. Pan, S. Nie, H. Hu, P. S. Yu and J. Guo, "Reverse Nearest Neighbor Search in Semantic Trajectories for Location-Based Services," in IEEE Transactions on Services Computing, vol. 15, no. 2, pp. 986-999, March-April 2022 is available at https://doi.org/10.1109/TSC.2020.2968309.

Reverse Nearest Neighbor Search in Semantic Trajectories for Location-based Services

Xiao Pan, Shili Nie, Haibo Hu, Philip S. Yu, Jingfeng Guo

Abstract—In resource planning scenarios, reverse k nearest neighbor search plays an important role. However, existing reverse k nearest neighbor search on trajectories only supports spatial features of trajectories. In this paper, we introduce Reverse k Nearest Neighbors query on Semantic Trajectories (RkNNST). Given a query point from a set of geo-textual objects (e.g., POIs), the query finds those trajectories which take this query point as one of their k nearest geo-textual correlative objects. To efficiently answer RkNNST queries, we propose a novel index IMC-tree, which organizes the global and local geo-textual information on semantic-enriched trajectories. A branch-and-bound search algorithm DOTA is then designed to traverse IMC-tree with various pruning rules. To speed up the computation of correlative distance, we also design an inverted-file-based algorithm to compute without enumerating all combinations of geo-textual objects. Experiments on a real dataset validate the effectiveness and efficiency of our proposed algorithms.

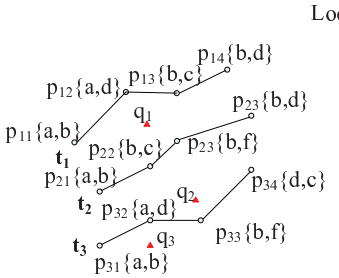
Index Terms—Location-based services, reverse nearest neighbor queries, semantic-enriched trajectories, geo-textual objects

1 Introduction

With the proliferation of smartphones, wearable devices and geo-social networks (e.g., Foursquare and Facebook Place), massive amounts of semantic trajectory data are generated everyday [11], [13], [20], [32]. A semantic trajectory [7], [26], [30] is a sequence of geo-textual locations where each location is associated with a timestamp and some semantic label(s), such as texts, photographs, and video. Since such semantic trajectories provide rich knowledge about the geo-textual locations, trajectory retrieval becomes important in many location based services [14], [15], [23].

In this paper, we study the reverse k nearest neighbor query problem on semantic trajectories (RkNNST). In a nutshell, RkNNST can be described as follows. The system has a set of semantic trajectories and geo-textual objects (i.e., a location with textual labels). The trajectories[1] can be historical or upcoming. RkNNST would like to retrieve the reverse k nearest neighbors (RkNN) of a query object, i.e., trajectories to which the query object is among the k semantically closest objects, or equivalently, the k most correlative geo-textual objects.

Figure 1 illustrates an example for our proposed scenarios, where (a) illustrates three trajectories t_1, t_2, t_3 with the semantic keywords of all geo-textual objects, and three query points q_1, q_2 and q_3, (b) plots the spatial locations of the three trajectories, and the semantic keywords and locations in the three queries, and (c) plots both the spatial and the correlative distances between a query q and each trajectory. In terms of spatial distance (i.e., the sum of Euclidean distances between each point in the trajectory to the query point), the R1NNs of q_1 are $\{t_1, t_2\}$; in terms of correlative distance (to be defined in Definition 2 as a combination of spatial distance and textual consistency), the R1NNST of q_1 is $\{t_1\}$ only.



(a)

Locations of semantic trajectories

	p_{i1}	p_{i2}	p_{i3}	p_{i4}
t_1	(0,4)	(2,6)	(4,6)	(6,7)
t_2	(1,3)	(3,4)	(4,5)	(7,6)
t_3	(1,1)	(3,2)	(5,2)	(7,4)

Geo-textual queries

	Labels	Locations
q_1	a,b,d	(3, 5)
q_2	a,b,c	(5, 3)
q_3	b,c,d	(3, 1)

(b)

Spatial dist.	t_1	t_2	t_3
q_1	**9.6**	**9.0**	15.2
q_2	16.6	12.1	**10.0**
q_3	21.1	16.4	10.2

Cor. dist.	t_1	t_2	t_3
q_1	**2.82**	9.0	5.6
q_2	7.4	**6.24**	**5.5**
q_3	10.2	13.5	7.2

(c)

Fig. 1. An RkNNST on semantic trajectories

RkNNST has many applications in geo-textual and geo-social services such as resource planning.

(1) Each user in location-based social networks such as Facebook and Foursquare leaves historical traces and forms activity trajectory. Each location in an activity trajectory contains both spatial location and activity tags (e.g., sport, dining and entertaining) that indicate user's interests. An RkNNST query can help a franchise or chain business to study the demographics of potential customers served by each store. These demographics can help make customized marketing and other businesses decisions.

(2) As another example, personalized online services such as Airbnb recommend a home to those travelers whose

- *X. Pan is with Shijiazhuang Tiedao University, China, 050043. E-mail: smallpx@stdu.edu.cn.*
- *S. Nie and J. Guo are with Yanshan University, China, 066004. E-mail: {nieshili,jfguo}@ysu.edu.cn*
- *H. Hu is with Hong Kong Polytechnic University, Hong Kong. E-mail: haibo.hu@polyu.edu.hk.*
- *P. S. Yu is with University of Illinois at Chicago, Chicago, IL, USA. E-mail: psyu@uic.edu*

1. We use trajectory to denote a semantic trajectory when the context is clear in the following sections.

ongoing traveling route treats this home as one of closet among all homes in terms of spatial distance and textual requirement. The textual requirement indicates the features of the traveling routes match the home owner's specialty mostly (e.g, religious culture, natural beauty, shopping and eating, etc.). Meanwhile, a home owner can discovery her potential tenants by issuing an RkNNST query.

Classical RNN involves two point sets: facilities and users [10], [27]. However, in RkNNST, the user sets are semantic trajectories (i.e. sequences of locations with semantic labels). As such, classical RNN methods cannot be applied to answer RkNNST. [23] is the first work to investigate RNN on trajectories for route planning. However, this problem is different from RkNNST in two aspects. First, [23] returns all points that take a trajectory as one of its k nearest travel routes, which is opposite to RkNNST. Second, the trajectories and points do not consider any textual or semantic information.

In this paper, we study RkNNST query processing. We face two main challenges. First, the RkNNST answers depend on sub-sequences of the semantic trajectories w.r.t the queried keywords. Thus, answering an RkNNST query needs to explore a huge number of geo-textual objects in an efficient manner. Specifically, a trajectory t and a query q are *correlative* if all keywords of q appear in t. For instance, t_1 is correlative with q_1 in Figure 1. Then, there could exist many sub-trajectories of t_1 that are also correlative with q. For example, $t_1{}_1^2(=< p_{11}, p_{12} >), t_1{}_2^3(=< p_{12}, p_{13} >)$ and the super-sequences of $t_1{}_1^2$ and $t_1{}_2^3$. All these sub-sequences can influence on RkNNST computation.

Second, RkNNST requires the geo-textual information in both a global (i.e., trajectory) level and a local (i.e., point) level. To efficiently access candidate sub-sequences, a semantic trajectory index should have both of them. However, existing indexes on raw trajectories, e.g., STR-tree [16], and MV3R-tree [22], only focus on the spatial feature without any textual information. On the other hand, the geo-textual indexes, e.g., IR-tree [3] and I^3 [29], only focus on geo-textual points as separate objects, not as a sequence of objects, i.e., trajectories.

To address these challenges, we propose a novel index IMC-tree, which preserves the global and local geo-textual information for the semantic trajectories. The IMC-tree is constituted by an inverted file and MC-trees, each corresponding to a keyword. Our proposed MC-tree stores the global spatial feature of the trajectories, while the local feature is embedded in the tree node. In order to answer an RkNNST query, an R-tree-based geo-textual index [24] is employed to index the geo-textual query points. With both trees, a branch-and-bound RkNNST algorithm DOTA is proposed to obtain a candidate set in the best-first paradigm. We also derive several important correlative distance bounds between a node in the IMC-tree and a node in the query index, so that trajectories can be pruned in an early stage.

The contributions are summarized as follows.

- We propose the problem of reverse k nearest neighbor queries on semantic trajectories. (RkNNST).
- We propose a novel index IMC-tree incorporating with both global and local geo-textual information of semantic trajectories.
- We propose an efficient algorithm DOTA for answering RkNNST. It is a branch-and-bound algorithm that can prune irrelevant trajectories effectively.
- A series of experiments are conducted on a real dataset to evaluate the performance of our proposed algorithms.

The rest of the paper is organized as follows. Section 2 formally defines the problem and gives an overview of the proposed method. The IMC-tree is introduced in Section 3. The query algorithm DOTA is presented in Section 4. Section 5 specifies several important correlative distance bounds and the pruning rules. We review the related work in Section 6. Section 7 details the performance evaluation results. Finally, the paper is concluded in Section 8.

2 Preliminarily and Method Overview

2.1 Definitions

We denote all the geo-spatial objects as (l, ws), whose location l is associated with a set of keywords ws. A semantic trajectory t is a sequence of geo-textual points, i.e., $t = (p_1, p_2, ..., p_n)$. A sub-trajectory of t is denoted as t_s^e, where $s \geq 1$ is the start position and $e \leq |t| = n$ is the end position of the sub-trajectory. Recall that, if a trajectory t and a query q are correlative, there could exist sub-trajectories of t that are also correlative with q. As such, we define the minimum correlative sub-trajectories.

Definition 1. *(Minimum correlative sub-trajectory) Given a (sub-)trajectory $t_s^e \subseteq t$ correlating with a query q, t_s^e is minimum if and only if $\nexists t' \subset t_s^e, q.ws \subseteq \bigcup_{\forall p \in t'} p.ws$. t_s^e is a minimum correlative (sub-)trajectory w.r.t. q.*

For instance, $t_1{}_1^2$ and $t_1{}_2^3$ are both the minimum correlative sub-trajectories (mcs) w.r.t q_1 in Figure 1. Obviously, the minimum correlative sub-trajectory of a trajectory w.r.t. a query is not unique.

We consider the average distance between the points in trajectory and the query as the distance from a trajectory to a query point. For convenient computation, we assume the trajectory lengths are same. In case the lengths are different, we can employ linear interpolation or truncate them into equal sizes. Thus, similar to [28], the distance between a minimum correlative sub-trajectory and a query q is defined to be the sum of the distance between each geo-textual point in the sub-trajectory and q.

Then the correlative distance between a trajectory and q is the minimum distance between any minimum correlative sub-trajectory and q. Or formally,

Definition 2. *(Correlative distance) Let mcs denote the set of the minimum correlative sub-trajectories of a trajectory t w.r.t. a query q. The correlative distance between t and q*

$$cd(t, q) = \underset{\forall t_s^e \in mcs}{MIN} \sum_{\forall p \in t_s^e} d(p, q).$$

For example in Figure 1, $cd(t_1, q_1) = MIN\{\sum_{\forall p \in t_1{}_1^2} d(p, q_1), \sum_{\forall p \in t_1{}_2^3} d(p, q_1)\}) = 2.82$. The correlative distance is symmetrical, that is, $cd(t, q) = cd(q, t)$.

Theorem 1 below gives the lower and upper bounds of correlative distance $cd(t, q)$. The lower bound is obtained

TABLE 1
Interpretation of symbols

term	description
t_s^e	a semantic trajectory from the start point s to the end point e
$ncn_k(t)$	k most nearest correlative neighbors of t
$rnn_k(q, QS)$	k reverse nearest neighbor of q
$d(q,t)$	the spatial distance between q and t
$cd(t,q)$	the correlative distance between q and t
$\#iwq$	the minimum number of keywords in the queries below a node
$ss = (sig, skt)$	the abstract, sig is a trajectory signature, and skt is the location sketch
ET (E_T)	a trajectory nodes set ET in the IMC-tree ($E_T \in ET$)
E_Q	a query node in the query index
$MINcd$	minimum correlative distance
$MAXcd$	maximum correlative distance
mcs	a set of the minimum correlative sub-trajectories

when the labeled keywords set of the nearest point on t to q contains at least one of keywords in $q.ws$. The upper bound $d(q,t)$ is the case when the labeled keywords $q.ws$ scatter at each position of t. Formally,

Theorem 1. *For a trajectory t and a query q,*

$$\underset{\forall p \in t \& p.ws \cap q.ws \neq \phi}{MIN} d(p,q) \leq cd(t,q) \leq d(q,t),$$

where $d(q,t) = \sum_{\forall p \in t} d(p,q)$ *is the* spatial distance *between q and t.*

The lower and upper bounds will be tightened and applied in the optimization process in Section 5.

2.2 Problem formalization

Before formally introducing RkNNST, we define the k nearest correlative neighbors given a trajectory t and a geo-textual object set.

Definition 3. *(k Nearest Correlative Neighbors, kNCN) Given a trajectory t, CO is the set of geo-textual objects which correlate with t. An object $o \in CO$ is one of the k most nearest correlative neighbors of t within CO, denoted by $o \in ncn_k(t, CO)$, if and only if $|\{o' | \forall o' \in CO \text{ and } cd(t,o') \leq cd(t,o)\}| < k$.*

Definition 4. *(RkNNST) Given a query set QS, a query $q(\in QS)$ and a trajectory set TS, our goal is to retrieve trajectories from TS whose k most nearest correlative neighbors within QS include q. That is, $rnn_k(q, QS)$=$\{t | q \in ncn_k(t, QS), t \in TS\}$.*

The output of the problem is a set of the minimum correlative sub-trajectories, where each item is $\{(t_{id}, s, e, cd)\}$. t_{id} is the trajectory identity, s and e are the start and end positions on t, and cd is the correlative distance between t_{id} and q.

From Definition 4, finding a query point's RkNNST is inevitable to find the k nearest correlative neighbors $ncn_k(t, QS)$ of each trajectory $t \in TS$, and then check if the query point is in $ncn_k(t, QS)$. Thus, the simplest method of RkNNST is to implement k nearest neighbors search for each trajectory iteratively. Specifically, given a trajectory $t \in TS$, the nearest correlative neighbors $ncn_k(t)$ of t are found from QS. Then, for each $q \in ncn_k(t)$, t is one of RkNNST for q.

The *drawback* of the method is obvious. Each time we want to find the RkNNST of a query, ALL trajectories should be scanned to compute nearest correlative neighbors over ALL query points. Both the trajectory set TS and the query points QS are accessed repeatedly. Moreover, computing k nearest correlative neighbors of t is based on enumerating the correlative sub-trajectories w.r.t. $\forall q \in QS$. Thus, the computation of RkNNST is costly, whose time complexity is $O(|TS||QS|) \times O(l^3)$, where l is the trajectory size.

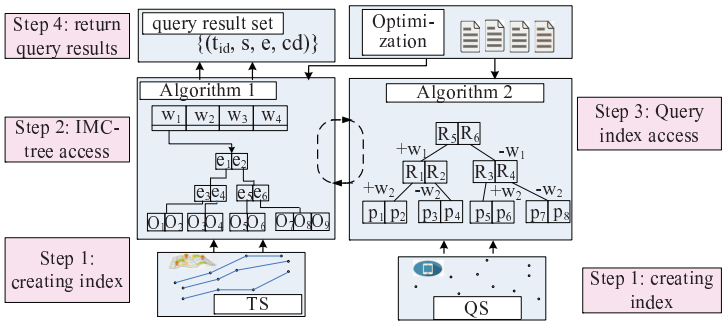


Fig. 2. Outline of DOTA

2.3 Method outline

Our proposed algorithm DOTA is a branch-and-bound algorithm using the best-first paradigm through the double trees access (i.e., the IMC-tree and a query index). The double indexes accesses benefit pruning massive negative trajectories and queries at an early stage. IMC-tree is a hybrid index on trajectories TS integrating the global and local geo-textual information (details in Section 3). We also employ a R-tree based geo-textual index (e.g., WIBR-tree, IR-tree, etc.) which supports boolean top-k queries [2], to index the queries points QS.

DOTA visits the IMC-tree and the index on queries alternately. The basic process is shown in Figure 2. After creating indexes on the trajectories TS and query points QS, we start from the IMC-tree access (i.e., Algorithm 1 in Section 4.1), while the query index access is embedded in each round of IMC-tree node access (i.e., Algorithm 2 in Section 4.2). Finally, the two trees accesses are terminated when RkNNST is obtained. Moreover, we propose a new algorithm (i.e., Section 4.3) to compute the correlative distances. We also propose an improvement strategy for batch process when a batch of queries arrive simultaneously (i.e., Section 4.4).

Assume that the node fanout of IMC-tree and the R-tree based query index are B_m and B_r respectively. Then, the time complexity of DOTA is $O(|q.ws| \cdot log_{B_m}|TS| \cdot log_{B_r}|QS|) \times O(l^2)$, where $|q.ws|$ is the number of keywords in the query point. For convenience, we list the interpretations of primary symbols throughout the paper in Table 1.

3 IMC-tree: A Hybrid Index on Trajectories

In this section, we propose a novel IMC-tree incorporating with the spatial and textual information, while with global and local geo-textual information of trajectories.

3.1 Basic structure

The IMC-tree consists of an inverted list and MC-trees. The inverted list stores the keywords vocabulary (i.e., the textual information of trajectories). For each keyword, we create an MC-tree (i.e., the spatial information of trajectories). MC-trees extend from traditional m-trees [4] with trajectory

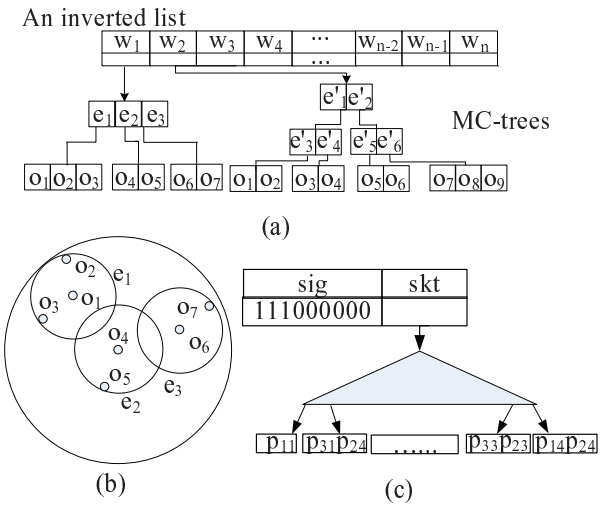


Fig. 3. IMC-tree illustration

abstracts. We will introduce the proposed MC-tree, an important component of IMC-tree, in this section first, and then explain the trajectory abstract in the next section.

MC-trees are based on m-trees since m-trees have a special distance feature, which can help for organizing the spatial distance between any two trajectories. That is, there is a radius r for each node in m-trees. The radius defines a Ball in the desired metric space. Then, every internal node and leaf residing in a particular node is at most distance r from the particular node, and every node and leaf node with node parent keep the distance r from it [4]. We use m-trees to index trajectory objects. Hence, the spatial distance between any two trajectories can be bounded using the radiuses.

M-trees require the distance function between index objects is an metric. That is, the distance function should satisfy the positivity, symmetry, and triangle inequality postulates. Before we prove the spatial distance between trajectories is an metric in Theorem 2, we first define the spatial distance function of trajectories.

Definition 5. *(Spatial distance between trajectories) The spatial distance between two trajectories $t \in TS$ and $t' \in TS$ is defined as the distance sum of the aligned points generally, i.e., $d(t,t') = \sum_{i=1}^{n} d(p_i, p'_i)$, where $p_i \in t$ and $p'_i \in t'$.*

Theorem 2. *Given three trajectories t_1, t_2, t_3, (1) $d(t_1,t_2) \geq 0$; (2) $d(t_1,t_2) = d(t_2,t_1)$;(3) $d(t_1,t_2)+d(t_1,t_3) \geq d(t_2,t_3)$.* The first two properties are obvious. The third one can be proved by summing n inequations of three aligned points on each trajectory. For space limited, the detail proof is omitted here.

MC-tree's structure: The skeleton of an MC-tree is an m-tree. An MC-tree is a tree structure, being composed of internal nodes and leaf nodes. Particularly,

(1) An internal node stores a set of routing trajectories[2]. A routing trajectory is denoted as $o_r = (t_r, pt, d, r, ss)$. Specifically, t_r is the identity of the routing trajectory; pt points to the covering tree MC-tree(o_r) which is an MC-tree below the routing trajectory o_r; d is the spatial distance between t_r with its parent routing trajectory; r is the covering radius where $d(t,t_r) \leq r(\forall t \in \text{MC-tree}(o_r))$; ss is the abstract for the trajectories covered by the MC-tree(o_r).

(2) A leaf node stores the trajectory objects. A trajectory object o_t consists the trajectory identity and the spatial distance $d(o_t, o_r)$ between the trajectory o_t and the parent routing trajectory o_r[3].

Example:Figure 3a illustrates an IMC-tree. The keyword vocabulary includes $\{w_1, w_2, ..., w_n\}$, which is organized as an inverted file. The MC-tree of keyword w_1 has two levels (i.e., the left MC-tree in Figure 3(a)). The MC-tree of w_1 is also illustrated by a visualized figure (i.e., Figure 3(b)), where each solid circle represents a trajectory object in the leaf nodes. The set of routing trajectory for the root is $\{e_1(o_1), e_2(o_4), e_3(o_6)\}$. That is, o_1 is the routing object of e_1, whose covering radius is assumed to be 2. Thus, the spatial distances from the trajectories below e_1 (i.e., $o_2.t_2$ or $o_3.t_3$) to the routing trajectory (i.e., $o_1.t_1$) are not larger than 2.

Overall, the global textual information is in the inverted list. If a trajectory doesn't contain any keyword in the query, the trajectory will not be checked in the query process. Meanwhile, the global spatial information concentrates in MC-trees. The covering radius in each node is the upper bounds of spatial trajectories between any two trajectories below the node. The radiuses and the exact locations of the routing trajectory are used to compute the upper bounds of the correlative distances between the nodes in MC-trees and the nodes in query index(in Section 5.2).

3.2 Trajectory abstract

Besides global information, local information of points in trajectories is also condensed into internal nodes of MC-trees, representing as trajectories abstracts. The abstract ss includes two components,i.e., $ss = (sig, skt)$.

(1) Trajectory signature sig identifies the existence of trajectories below the branch of the MC-tree. sig is represented as a bit vector, where the i-th bit represents whether the subtrees of this node contain the trajectory or not. In particular, each trajectory identity will be mapped onto one bit by a hash function. Then its i-th bit is 1 if there exists a trajectory point within this node and its identity is mapped to the i-th position. Otherwise, the i-th bit is set to 0. sig in internal nodes is the trajectory signature union of its sub-nodes.

(2) Location sketch skt points to a linear quad-tree [31]. Employing linear quad-trees, correlative points on trajectories w.r.t the keyword w are organized. In particular, the space is divided into equal-size cells, whose locations are encoded based on the Morton code. The linear quad-tree keeps the non-empty cells with the codes.For instance, the location of p_{31} on t_3 in Figure 1 is only in the MC-trees w.r.t. the keywords a and b. As a result, the nearest correlative point on trajectories to a query can be found easily using the method in [31].

Figure 3c illustrates the abstract in e_1. We assume there are 9 trajectories. sig (i.e., a bitvector) represents t_1, t_2 and t_3 are contained in e_1. The location sketch skt points to a

2. They are called routing objects in m-trees, which means the distance between the objects below this node and the routing object is less than the radius r.

3. When the context is clear, we do not distinguish trajectories and trajectory objects.

linear quadtree storing the non-empty cells of Morton code. That is, the cells in the linear quadtree are traversed by t_1, t_2, and t_3. Meanwhile, the points of the three trajectories in the cells contain the keyword w_1.

4 DOTA:RkNNST QUERY ALGORITHM ON DOUBLE TREES ACCESS

DOTA visits the IMC-tree and the index on queries alternately. In the first place, we need to elaborate an important variable $cand_k$, which is the intermediated computation result. $cand_k$ stores the k nearest correlative neighbor candidate queries for a trajectory (set). As we know that if the query q is far from a trajectory t than the existing candidates queries in $cand_k$, the trajectory t has no chance to be the RkNNST of q. This intuition triggers us maintaining $cand_k$ for visited trajectories (nodes[4]). As a result, the irrelevant trajectories can be pruned.

IMC-tree access is the entrance of DOTA, and the query index access is embedded in IMC-tree access. The basic procedure of DOTA is as follows. The roots of MC-trees w.r.t. the keywords in $q.ws$ are inserted into a queues set. A queue front E_T is popped up. With the queue front E_T, the query index is visited as the best-first paradigm to confirm the state of the trajectory node E_T (i.e., pruned, visit, or not-sure) and maintain the k nearest correlative neighbor candidates $cand_k$ for E_T. If the state of E_T is not pruned, the sub-nodes of E_T are inserted into the queues set. The above procedure is repeated. Finally, specific trajectories will be retrieved from leaf nodes of MC-trees and inserted into the queues set. When a specific trajectory is popped up, we visit the query index with specific trajectories. Then, RkNNST of the query q is updated. The whole procedure is stopped when the queues set becomes empty.

In this section, we first elaborate the detail access procedures on two trees in Section 4.1 and Section 4.2 respectively. Then, the algorithm for computation the correlative distances is proposed in Section 4.3. Finally, we summarize and improve DOTA for batch process in Section 4.4.

4.1 Access on trajectories index

This section begins with embodying $cand_k$. $cand_k$ is a set with k pairs, and each pair is in the form of $< q_i, md_i >$ $(1 \leq i \leq k)$. For a specific trajectory, $cand_k$ stores k queries who are the k-th nearest correlative candidate queries, and md_i is the correlative distance between q_i and this trajectory. For a trajectory set, $cand_k$ stores k queries as well, while md_i is the maximum correlative distance (i.e., the upper bound) between this query and the trajectory set (i.e., defined in Section 5.2).

The IMC-tree is visited as the best-first paradigm. Algorithm 1 shows the detailed visiting process. Since each keyword in the query $q.ws$ corresponds to an MC-tree, we use a queues set SET to maintain the accessed nodes from the IMC-tree (shown in Figure 4). Each item in SET is a queue. The queue QE_{wi} contains the accessed nodes in the MC-tree corresponding to the keyword $wi \in q.ws$. Each queue is sorted by the minimum correlative distance, $MINcd_{E_T\_q}$, between a trajectory set E_T and a query q (details refers to Corollary 1). The queue fronts in SET constitute the set ET, which is sorted by $MINcd_{E_T\_q}$ logically.

4. A trajectory set is covered by a trajectory node from an IMC-tree. Thus, we use a trajectory node and a trajectory set alternatively when the context is clear.

Algorithm 1 Algorithm of IMC-tree Access

Input: an IMC-tree $tree_1$, a WIBR-tree $tree_2$, a query q
Output: RkNNST of q

1: put the roots of MC-trees in $tree_1$ w.r.t. $q.ws$ into SET;
2: initialize $sig_\cap$ by the roots of MC-trees w.r.t. $q.ws$;
3: **if** $sig_\cap = \varnothing$ **then**
4: **break**; {Rule 1 in Section 5.3}
5: **while** SET is not empty **do**
6: compute $MINcd_{ET\_q}$;
7: DeQueue(ET, E_T);
8: **if** E_T is an internal node **then**
9: state=apply Alg. 2 to check the state of E_T using $tree_2$;
10: **if** $state \neq pruned$ **then**
11: **for** each child CE_T below E_T **do**
12: **if** CE_T is not pruned by Rule 1 **then**
13: copy $E_T.cand_k$ to CE_T and update $MAXcd_{CE_T\_q_i}$;
14: insert CE_T into the queue QE_T in SET;
15: **else**
16: **if** E_T is a leaf node **then**
17: **for** each $t \in E_T$ **do**
18: copy $E_T.cand_k$ to t;
19: compute $cd_{t,q}$;
20: **if** $cd_{t,q} > t.cand_k.md_k$ **then**
21: t is pruned; {Rule 2 in Section 5.3}
22: **else**
23: update $cand_k$ with $cd_{t,q}$;
24: insert t into the queue QE_T in SET;
25: **else**
26: t is used to update $rnn_k(q)$ in $tree_2$;

Initially, the roots of MC-trees w.r.t. the keywords in $q.ws$ are inserted into SET, and $sig_\cap$ is initialized as the intersection of trajectory signatures in the MC-tree roots (Line 1 and Line 2). If $sig_\cap = \varnothing$, none of the trajectories contains all the keywords in $q.ws$ (Rule 1 in Section 5.3); otherwise, the minimum queue front E_T is popped from ET (Line 7). With an internal node E_T, the query index is visited in Algorithm 2 (Line 9). After visiting the query index, the state of E_T is confirmed. If E_T is pruned, E_T and the branches under E_T are neglected. If E_T cannot be pruned (Line 10), the children of E_T are inserted into SET. With a leaf node E_T, $cd(t, q)$ for each trajectory $t \in E_T$ is computed. The state of t is checked by Rule 2 in Section 5.3. If t is not pruned, t is inserted into SET (Lines 16∼24). With a specific trajectory, k nearest correlative neighbors $ncn_k(t)$ of t and k reverse nearest neighbors $rnn_k(q)$ of q are updated during the query index access(Line 26). The process is repeated until SET is empty.

What needs to be explained further is that, if a trajectory node E_T is un-pruned (Line 10), the sorted list $cand_k$ of E_T is maintained after the access of the query index. $E_T.cand_k$ maintains k sorted pairs. When the child CE_T of E_T is inserted into the queue QE_T, CE_T inherits the queries in $cand_k$ of E_T (Lines 11∼14). The queries in inherited $cand_k$ are the intermediate results, which are benefit for pruning the negative trajectories (Rule 2 in Section 5.3). When E_T is a leaf node, each trajectory $t \in E_T$ inherits the $cand_k$ of E_T (Line 18) as well. With the specific trajectories and queries, the correlative distances are computed for $t.cand_k$(Line 19).

If $cd(t, q)$ is larger than the correlative distance of k-th item in $t.cand_k$, t is pruned (Lines 20∼21); otherwise, t with the new $cand_k$ is inserted into SET (Lines 23∼24).

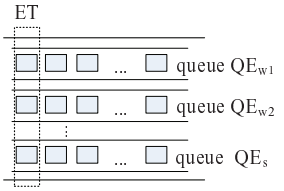


Fig. 4. SET and ET illustration

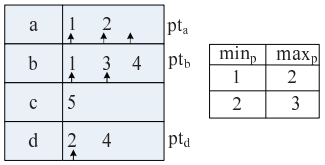


Fig. 5. Corr. dist. computation

4.2 Access on query index

Recall that we use an R-based geo-textual index on queries points, which can support boolean top-k search. In our experiments, we use a WIBR-tree [24], however, which it is not limited to. The effect of query index access includes two folds: confirming the state of the trajectories and maintaining the k nearest correlative neighbor candidates $cand_k$. The query index is also visited as the best-first paradigm, under two kinds of inputs: a trajectory node E_T(Line 9 in Algorithm 1) and a trajectory t (Line 26 in Algorithm 1). Algorithm 2 shows the main process of the query index access with the trajectory node E_T. The basic process of the query index access with a specific trajectory t is almost same except several correlative distances revision. After that, we will discuss the difference between access with a trajectory node and with a trajectory.

In Algorithm 2, we use a priority query queue $QueQ$ to maintain the nodes of the query index. $QueQ$ is sorted by the minimum correlative distance, $MINcd_{E_T\_E_Q}$, between a trajectory node E_T and a query node $E_Q(\in QueQ)$ (details refers to Theorem 3 in Section 5.1). Algorithm 2 begins with checking the node E_T by Rule 2 (details in Section 5.3). If E_T is not pruned, the root of query index is inserted into $QueQ$ (Lines 4∼5). While $QueQ$ is not empty, the state of E_T is checked further by the other rules in Section 5.3 (i.e., Algorithm 5). If E_T can be pruned or must be visited, the algorithm returns (Lines 8∼9); otherwise $QueQ$ and $E_T.cand_k$ are updated (Lines 10∼17). Then, E_T is re-checked with the new $QueQ$.

The basic process of the query index access with a specific trajectory t is almost same as Algorithm 2 only with several correlative distances revision. In particular, firstly, the conditional judgment in Line 1 changes to $cd(q, t) > t.cand_k.md_k$ as Rule 2. If the correlative distance between q and t is larger than the k-th candidate item in t, the query q is not in $ncn_k(t)$. Secondly, the minimum correlative distance, $MINcd_{E_T\_E_Q}$, between a trajectory node E_T and a query node E_Q reduces to the minimum correlative distance $MINcd_{t\_E_Q}$ between a query node E_Q and a trajectory t (Definition refers to Theorem 4), and $E_T.cand_k$ changes to $t.cand_k$. Thirdly, the candidate $ncn_k(t)$ for t can be obtained from Line 15 to Line 17 with specific trajectories and queries.

Updating cascade: Remark that $t.cand_k$ is updated when we use a specific trajectory t to traverse the query index (Line 26 in Algorithm 1). When $t.cand_k$ is updated, the $cand_k$ of the leaf node of MC-tree covering t can also be updated. Similarity, the $cand_k$ of the ancestor nodes of this leaf node should also be updated. The update is repeated until we backtrack to the root of MC-tree. Such that, the bounds of the IMC-tree nodes become more constrained. As a result, when a new query arrives, the pruning rules and the visiting rules (in Section 5.3) will be more effective. In order to reduce the cost of updating cascade, we delay the $cand_k$ updating to the moment when all trajectories below a leaf node complete $cand_k$ updating. Similarly, for a trajectory node in an MC-tree, the updates operation is delayed until the $cand_k$ of all its children completes.

Algorithm 2 Query index access with the node E_T

Input: an MC-tree node E_T, a query index $tree_2$, the query q
Output: the state of E_T, $E_T.cand_k$

```
1: if MINcd_{ET_q} > E_T.cand_k.md_k then
2:    return prune; {Rule 2 in Section 5.3}
3: else
4:    rt=the root of tree_2;
5:    insert (rt, MINcd_{E_T_rt}) into QueQ;
6:    while QueQ is not empty do
7:       state=invoke Alg. 5 to prune E_T
8:       if state=prune or state=visit then
9:          return state;
10:      pop the front (E_Q,MINcd_{E_T_E_Q}) from QueQ;
11:      if E_Q is an internal node then
12:         for each child CE_Q of E_Q do
13:            insert (CE_Q, MINcd_{CE_Q_E_T}) into QueQ;
14:      else
15:         for each q' in E_Q do
16:            if MAXcd_{q'_E_T} < E_T.cand_k.md_k then
17:               updating E_T.cand_k with q';
18: return visit;
```

4.3 Correlative distances computation

An important step in Algorithm 1 is to compute the correlative distance between a trajectory and a query (Line 19). From Definition 2, we need to find all the minimum correlative sub-trajectories w.r.t q for computing correlative distances. Since a minimum correlative sub-trajectory is a sub-sequence of the trajectory. Thus, a straightforward method is to enumerate all sub-trajectories through a three-layer nested loop, and to choose the one whose correlative distance is minimum. The time complexity of naive method is O(l^3). In this section, we propose an algorithm PI employing several pointers and an inverted file. PI goes through the whole trajectory by jumping on segments.

The algorithm contains three main steps. First, we create a keyword-position inverted file for the trajectory. Each item in the inverted file contains a keyword and a sorted posting list. The posting list is constituted by the positions on the trajectory which contains this keyword. Figure 5 illustrates the inverted file for the trajectory t_1 in Figure 1. The keyword a is contained in the position 1 and the position 2.

Second, we employ $|q.ws|$ pointers to record the current checking positions in the inverted file for the different keywords. The minimum and the maximum positions bound a segment. The sample positions in this segment constitute a correlative sub-trajectory candidate to q. For example, we want to find the correlative sub-trajectories for q_1({a, b, d}) on t_1. Initially, the pointers pt_a and pt_b both point to the position 1, and the pointer pt_d is at the

position 2. Thus, $min_p = MIN\{pt_a, pt_b, pt_c\}$=1, $max_p = MAX\{pt_a, pt_b, pt_c\} = 2$, and the current segment is [1, 2]. The candidate sub-trajectory is t_1^2. Then, $\sum_{p \in t_1^2} d(p,q)$ is set as the $cd(t,q)$.

Third, the pointers with smallest position number move to the next positions in the posting list. The locations in the new segment constitute another correlative sub-trajectory candidate $t_{min_p}^{max_p}$. If $\sum_{p \in t_{min_p}^{max_p}} d(p,q)$ is less than the current $cd(t,q)$, $cd(t,q)$ is updated by $\sum_{p \in t_{min_p}^{max_p}} d(p,q)$. In the running example, pt_a and pt_b move forward to the next positions (i.e., 2 and 3 in each posting list) respectively. At this moment, $min_p = MIN\{pt_a, pt_b, pt_c\}$=2, $max_p = MAX\{pt_a, pt_b, pt_c\} = 3$. Therefore, the samples in t_2^3 constitute another candidate trajectory. The movement step is repeated until one of the pointers points to the end of the posting list. Continuing with the running example, the algorithm ends when pt_a arrives at the end of the list.

The algorithm is shown in Algorithm 3. The worst time complexity is O(l^2).

Algorithm 3 PI: Computing the correlative distance

Input:an inverted file f for trajectory t, q
Output: $cd(t,q)$

1: $pt_1, ..., pt_m$ point to the first positions whose keyword is contained in $q.ws$;
2: $cd = \infty$;
3: **while** $pt_1 \neq NULL$ and ..., and $pt_m \neq NULL$ **do**
4: $minp$=min($pt_1, pt_2, ..., pt_m$);
5: $maxp$=max($pt_1, pt_2, ..., pt_m$);
6: **if** $cd > \sum_{p \in t_{minp}^{maxp}} d(p,q)$ **then**
7: $cd = \sum_{p \in t_{minp}^{maxp}} d(p,q)$;
8: pointers who equal $minp$ moves to the next position in the list;

4.4 Improvement for batch process

Algorithm 1 is the procedure for one query. When a batch of queries arrives simultaneously, we don't need to run Algorithm 1 for every query, since $cand_k$ for each trajectory changes little after parts of queries run through the IMC-tree and the tree of the query index. At that moment, we only traverse IMC-tree breadth-firstly without the query index access. The detail algorithm is shown in Algorithm 4.

DOTA for batch process begins with creating an IMC-tree and a query index on a trajectory set and a query set respectively (Line 1). For each query q, Algorithm 1 is invoked initially (Line 4). When every trajectory owns k nearest candidates $cand_k$, the MC-trees associated with $q.ws$ are traversed breadth-firstly (Lines 6∼19).

5 Optimization

In general, given a query q, a trajectory nodes set ET (i.e., a trajectory set) and a query node E_Q (i.e., a query set) are being accessed. We divide the situation into two cases according to the number of trajectories in ET. **Case 1**, a trajectory set ET and a query set are being accessed; **Case 2**, a trajectory t and a query set are being accessed. The lower bounds and the upper bounds of the correlative distances are discussed under the two cases in Section 5.1 and Section 5.2 respectively. Then, the pruning rules are elaborated in Section 5.3.

Algorithm 4 DOTA for batch process

Input:a queries set QS, a trajectories set TS
Output: RkNNST for each query

1: create an IMC-tree on TS and a query index on QS;
2: **for** each q in QS **do**
3: **if** NOT ALL trajectories in TS are being visited **then**
4: invoke Algorithm 1 to find RkNNST for q on two trees;
5: **else**
6: initialize $sig_\cap$ and SET with the roots of the MC-trees
7: **if** $sig_\cap = \varnothing$ **then**
8: **continue**; {Rule 1 in Section 5.3}
9: **while** SET is not empty **do**
10: E_T=pop the minimum front from SET;
11: **if** $E_T.cand_k.md_k < MINcd_{E_T\_q}$ **then**
12: **continue**; {Rule 2 in Section 5.3}
13: **else**
14: **if** E_T are NOT leaf node **then**
15: insert children of E_T passing Rule 1 into SET;
16: **else**
17: **for** each t in E_T **do**
18: **if** $cd(q,t) \leq t.cand_k.md_k$ **then**
19: t is one of the RkNNST of q;

5.1 The lower bounds of the correlative distances

5.1.1 Case 1: a trajectory set and a query set

We define the minimum correlative distance between a trajectory set ET and a query set E_Q in Definition 6 under the general case (i.e., Case 1).

Definition 6. *(Minimum correlative distance between a trajectory set and a query set, $MINcd$)*
$\forall t \in ET, \forall q \in E_Q$, $cd(t,q) \geq MINcd$.

Theorem 3 will specify $MINcd$ computation. We compute $MINcd$ based on the spatial distance between c_{nn} and the query set E_Q. c_{nn} is the nearest cell obtained from the abstract (i.e., linear quad-trees) of E_T to E_Q employing the existing technique [31].

Theorem 3. ***Minimum correlative distance between a trajectory set and a query set computation***

$$MINcd = \underset{\forall E_T \in ET}{MIN}(MINcd_{E_T\_E_Q}),$$

where $MINcd_{E_T\_E_Q} = d(E_Q, c_{nn})$. Generally, a minimum bounding rectangle is used to represent the location of E_Q in an R-tree based index. $d(E_Q, c_{nn})$ is the traditional minimum spatial distance between two rectangles.

Proof. $\forall t \in E_T, \forall q \in E_Q, cd(t,q) \geq d(p,q)$ as Theorem 1, where p is the nearest correlative point to q on t. c_{nn} is the nearest cell in the location sketch of E_T to E_Q. Obviously, $d(p,q) \geq d(c_{nn}, E_Q)$. $\forall E_T \in ET, \forall t \in E_T, \forall q \in E_Q$, $cd(t,q) \geq \underset{\forall E_T \in ET}{MIN} d(E_Q, c_{nn})$. □

Particularly, we denote the minimum correlative distance between a trajectory set ET and a query q as $MINcd_{ET\_q}$ when E_Q degenerates to a query. Corollary 1 is derived from Theorem 3

Corollary 1. ***Minimum correlative distance between a trajectory set and a query computation***

$$MINcd_{ET\_q} = \underset{\forall E_T \in ET}{MIN}(MINcd_{E_T\_q}),$$

where $MINcd_{E_T\_q} = d(q, c_{nn})$ *and* $d(q, c_{nn})$ *is the traditional spatial distance between a point to a rectangle.*

5.1.2 Case 2: a trajectory and a query set

Though Case 2 is a special case of Case 1 when ET covers only one trajectory, we do not compute $MINcd_{t\_E_Q}$ by concretizing Theorem 3. That is because we can use the exact locations instead of location sketch in the specific trajectory.

The distance between the nearest point p_{nn} on t to E_Q is the lower bound of $MINcd_{t\_E_Q}$. The best case is that the labeled keywords set of p_{nn} covers $q.ws$. Based on this intuition, Theorem 4 shows the computation method.

Theorem 4. *Let* $p_{nn} \in t$ *be the nearest point to* E_Q *considering the spatial feature only, then*

$$MINcd_{t\_E_Q} = \lceil \frac{E_Q.\#iwq}{t.\#awp} \rceil \times d(p_{nn}, E_Q).$$

$\#awp$ is the maximum number of keywords labeled with each location on a trajectory t. Then, at least $\lceil \frac{E_Q.\#iwq}{t.\#awp} \rceil$ points on t are required to form a correlative sub-trajectory.

However, $MINcd_{t\_E_Q}$ in Theorem 4 depends on the position of p_{nn} heavily. If p_{nn} is not in any correlative sub-trajectory, $MINcd_{t\_E_Q}$ is loose. Since the trajectory t and the keywords set associated with E_Q are both specific, we begin with finding the correlative sub-trajectories mct of t w.r.t. E_Q. Then, we use Theorem 4 on each correlative sub-trajectory in mct to tighten $MINcd_{t\_E_Q}$. Thus, we revise Theorem 4 in Theorem 5.

Theorem 5. ***Minimum correlative distance between a trajectory and a query set computation***

$$MINcd_{t\_E_Q} = \underset{t_s^e \in mct}{MIN}(MINcd_{t_s^e\_E_Q}).$$

mct is the set of minimum correlative sub-trajectories of t w.r.t. the keywords set of E_Q. $MINcd_{t_s^e\_E_Q}$ is computed as Theorem 4.

5.2 The upper bounds of the correlative distances

In this section, we will define the maximum correlative distances and give the computation methods under the above two cases. Before that, we discuss two spatial triangle inequalities among trajectories and queries.

Theorem 6. *Spatial triangle inequalities among trajectories and queries.*

- *Given a trajectory t and two queries q_1, q_2, $d(q_1, t) + d(q_2, t) \geq n \times d(q_1, q_2)$, and $d(q_1, t) \leq d(q_2, t) + n \times d(q_1, q_2)$.*
- *Given two trajectories t_1, t_2 and a query q, $d(q, t_1) + d(q, t_2) \geq d(t_1, t_2)$ and $d(q, t_1) + d(t_1, t_2) \geq d(q, t_2)$.*

The proof is straightforward. Due to space limitations, we omit it here. The two properties will be used in maximum correlative distances computation.

5.2.1 Case 1: a trajectory set and a query set

Definition 7. *(Maximum correlative distance between a trajectory set and a query set, $MAXcd$)*
$\forall t \in ET, \forall q \in E_Q$, $cd(t, q) \leq MAXcd$.

Recall that the upper bounds of any two trajectories can be obtained through the radiuses marrying up with the specific locations of routing trajectory. Employing this property, we can compute the maximum correlative distance between a trajectory set E_T and a query set E_Q. ET is constituted by several trajectory sets. Therefore, the minimum one among the maximum correlative distances between a trajectory set E_T and a query set E_Q is set as $MAXcd$. Theorem 7 will show the computation method.

Theorem 7. ***Maximum correlative distance between a trajectory set and a query set computation***

$$MAXcd = \underset{E_T \in ET}{MIN}(MAXcd_{E_T\_E_Q}),$$

where

$$MAXcd_{E_T\_E_Q} = d(q_c, o.t) + \frac{n \times L}{2} + o.r.$$

q_c is the center of the minimum bounding rectangle of E_Q, L is the diagonal length of the minimum bounding rectangle of E_Q, and o is the routing trajectory of E_T.

Proof. From Theorem 1, $\forall q' \in E_Q, \forall t \in E_T, cd(t, q') \leq d(q', t)$. From the second in-equation of Theorem 6 and the m-tree feature, $d(q', t) \leq d(q', o.t) + d(o.t, t) \leq d(q', o.t) + o.r$. From the first in-equation of Theorem 6, $d(q', o.t) < d(q_c, o.t) + n \times d(q_c, q')$. Since $d(q_c, q') \leq L/2$, $d(q', t) \leq d(q_c, o.t) + \frac{n \times L}{2} + o.r$. Since the AND semantics, the in-equation for each $E_T \in ET$ should be satisfied. Therefore, the minimum value is employed. □

In Theorem 7, we don't need to access each trajectory below an inter-node E_T to compute the upper bounds. Instead, we only use the dominated trajectory with the help of the spatial property of MC-tree, which is efficient.

When the query set E_Q reduces to a query q, the maximum correlative distance between ET and q is denoted as $MAXcd_{ET\_q}$. $MAXcd_{ET\_q}$ is computed as a special case of Theorem 7 with $q_c = q$ and L=0, which is shown in Corollary 2.

Corollary 2. ***Maximum correlative distance between a trajectory set and a query computation***

$$MAXcd_{ET\_q} = \underset{E_T \in ET}{MIN}(MAXcd_{E_T\_q}),$$

where $MAXcd_{E_T\_q} = d(q, o.t) + o.r$ and o is the routing trajectory of E_T.

5.2.2 Case 2: a trajectory and a query set

The maximum correlative distance between a trajectory and a query set is denoted as $MAXcd_{t\_E_Q}$. We employ the maximum spatial distance between p and the minimum bounding rectangle of E_Q computing the maximum correlative distance. Like Theorem 5, we employ the minimum correlative sub-trajectories to tighten $MAXcd_{t\_E_Q}$ computation.

Theorem 8. *(Maximum correlative distance between a trajectory and a query set computation)*

$$MAXcd_{t\_E_Q} = \underset{t_s^e \in mct}{MIN}(MAXcd_{t_s^e\_E_Q})$$

mcs is the set of the minimum correlative sub-trajectories to $E_Q.ws$.

$$MAXcd_{t_s^e\_E_Q} = \sum_{\forall p \in t_s^e} mxd(p, E_Q),$$

where $mxd(p, E_Q)$ is the maximum spatial distance between p and E_Q.

Proof. Assume $MAXcd_{t\_E_Q}$ is computed on the sub-trajectory t_s^e. We aim to prove $cd(t, q') \leq MAXcd_{t_s^e\_E_Q}$, for $\forall q' \in E_Q$, .

$\forall q' \in E_Q$, the correlative distance $cd(t, q')$ is computed on the sub-trajectories in mct, since $q'.ws \subseteq E_Q.ws$. Then, the computation has two cases. One case is that $cd(t, q')$ is computed from the sub-trajectory t_s^e; the other case is that $cd(t, q')$ is computed from another sub-trajectory $nt_s^e (\in mcs)$.

For the first case, the actual minimum correlative sub-trajectory w.r.t. q' is the subset of t_s^e. Meanwhile, $\forall p \in t_s^e$, $d(p, q') \leq mxd(p, E_Q)$. Thus, $cd(t, q') \leq \sum_{\forall p \in t_s^e} d(p, q') \leq MAXcd_{t_s^e\_E_Q}$.

For the other case, the minimum correlative sub-trajectory w.r.t. q' is the subset of nt_s^e, denoted as nt_{s+i}^{e-j}. $cd(t, q') = \sum_{p' \in nt_{s+i}^{e-j}} d(p', q')$. Since $cd(t, q')$ is computed on nt_{s+i}^{e-j}, $\sum_{p' \in nt_{s+i}^{e-j}} d(p', q') < \sum_{p \in t_s^e} d(p, q')$. Otherwise, $cd(t, q')$ would be computed from t_s^e. From the above proof, we know $\sum_{\forall p \in t_s^e} d(p, q') \leq MAXcd_{t_s^e\_E_Q}$. Thus, $cd(t, q') \leq MAXcd_{t_s^e\_E_Q}$.

□

5.3 Pruning and visiting rules

In order to improve the query efficiency, the algorithm expects to visit the necessary nodes as early as possible, and the unnecessary nodes as less as possible. During the access of the IMC-tree (in Algorithm 1), we expect that less nodes in the IMC-tree are inserted into SET, such that less rounds of query index access being invoked. If a trajectory node E_T is pruned, the children nodes of E_T will not be inserted into SET. Meanwhile, during the query index access (in Algorithm 2), we expect a trajectory node E_T is pruned earlier, such that the query index access breaks off earlier. The essence of the two expectations is same. That is, we should prune negative trajectory nodes E_T as early as possible.

Based on this intuition, we propose several access rules (i.e., pruning rules and visiting rules) to achieve this target. Rule 1 is proposed according to the AND semantics implied by the textual requirements in the query. Each keyword w points to an MC-tree in the IMC-tree. Assume that E_{T_w} is the root of the MC-trees for $w \in q.ws$. Let $sig_\cap = \bigcap_{w \in q.ws} E_{T_w}.sig$. In the extreme case, $sig_\cap = \varnothing$ means that none of trajectories contains the keywords $q.ws$ meanwhile. We generalize this observation in Rule 1 to prune the trajectories containing partly or none of the keywords in $q.ws$.

Rule 1. *(Pruning rule) Assume E_{T_w}, is the current checked node in the MC-tree corresponding to the keyword w'. If*

$$\bigcap_{i=1 \& w_i \neq w'}^{|q.ws|} QE_{w_i}.sig \cap E_{T_w}.sig = \varnothing,$$

E_{T_w}, is pruned. $QE_{w_i}.sig$ is the queue signature for the keyword w_i. That is, $QE_w.sig$ is the signatures union for all the nodes in the queue QE_w, i.e. $QE_{w_i}.sig = \bigcup_{j=1}^{|QE_{w_i}|} E_{T_{w_j}}.sig$, where $E_{T_{w_j}} \in QE_{w_i}$.

Rule 1 shows that the signature of the new node $E_{T_{w'}}$ is checked by being intersected with the union of the queue signatures. Readers would prefer E_{T_w}, intersecting with the signatures of the queue fronts in ET like MC-tree roots checking at the beginning. However, this way will lead to missing results. For example, $E_{T_{w'}}$ contains t_1. Unfortunately, t_1 is covered by the second nodes in the other queues instead of the fronts in ET. If only the queue fronts are checked,t_1 would be missing. Rule 1 is used in Algorithm 1 (i.e., Lines 2∼4 and Line 12).

It is obvious that if there exist k queries, whose correlative distances to the trajectory t are less than $cd(q, t)$, $q \notin ncn_k(t)$. Even if the k queries are not in $ncn_k(t)$, the observation still holds. We generalize this intuition by employing the intermediate result $cand_k$ in E_T.

Rule 2. *(Pruning rule)*

- *Rule 2.1: For a trajectory t and a query q, if $cd(q, t) > t.cand_k.md_k$, t is pruned.*
- *Rule 2.2: For a trajectory node E_T, if $MINcd_{ET\_q} > E_T.cand_k.md_k$, E_T is pruned.*

Rule 2.1 is the special case of Rule 2.2. $MINcd_{E_T\_q} > E_T.cand_k.md_k$ implies that for any $t \in E_T$, the minimum correlative distance between t and q is larger than the k-th maximum correlative distance of E_T.Rule 2.1 is used in Algorithm 1 (Line 20), and Rule 2.2 is applied in Algorithm 2 (Line 1).

The following proposed Rule 3 and Rule 4 are only used in Algorithm 2 to check the state of the trajectory node E_T. Rule 3 is a pruning rule and Rule 4 is a visiting rule. Before Rule 3 exposition, we first explain the theorems that Rule 3 is based on.

Theorem 9. *Given a query q, a trajectory node E_T in the IMC-tree and a query node $E_Q \in QueQ$, if $MAXcd_{E_T\_E_Q} < MINcd_{ET\_q}$ and $|E_Q| \geq k$, E_T and the subtrees of E_T cannot contain RkNNST of q .*

Proof. If $MAXcd_{E_T\_E_Q} < MINcd_{ET\_q}$, the queries in E_Q are all nearer to the trajectories in E_T than q. Meanwhile, the number of queries in E_Q is not less than k. Thus, q has no chance to be one of kNCN for trajectories in E_T. □

When E_T is a specific trajectory, Theorem 9 also comes true. However, the related correlative distances can be computed tightly. Thus, Corollary 3 is formalized as follows.

Corollary 3. *A trajectory t cannot be the RkNNST of the query q if $MAXcd_{t\_E_Q} < cd(t, q)$ and $|E_Q| \geq k$.*

Theorem 9 and Corollary 3 consider the queries under the query node E_Q in $QueQ$ only. Incorporating the query nodes in $QueQ$ and queries in $E_T.cand_k$, we propose Rule 3.

Rule 3. *(Pruning rule): Given the query nodes set $QueQ$ and a query q,*

- Rule 3.1*: For a trajectory node E_T, assume that num_1 is the number of queries in the nodes of $QueQ$ whose $MAXcd_{E_T\_E_Q} < MINcd_{ET\_q}$, and num_2 is the number of queries in $E_T.cand_k$ whose $MAXcd_{E_T\_q'} < MINcd_{ET\_q}$. If $num_1 + num_2 \geq k$, E_T is pruned.*
- Rule 3.2*: For a trajectory t, assume that num_1 is the number of queries in the nodes of $QueQ$ whose $MAXcd_{t\_E_Q} < MINcd_{ET\_q}$, and num_2 is the number of queries in $t.cand_k$ whose $cd(t, q') < MINcd_{ET\_q}$. If $num_1 + num_2 \geq k$, t is pruned.*

All the above rules are pruning rules. On the other hand, if we know a trajectory node E_T must be visited, the query index doesn't need to be traversed further in Algorithm 2. Specifically, the loop (i.e., Lines 7∼17) of Algorithm 2 can be broken off earlier. Let's consider an extreme case: Given a query q and a trajectory node E_T, $\forall E_Q \in QueQ$, $MINcd_{E_T\_E_Q} > MAXcd_{ET\_q}$. This extreme case indicates that q is nearest to the trajectories in E_T. Rule 4 is generalized from the extreme case.

Rule 4. *(Visiting rule) E_{Q_m} is assumed to be the m-th item in the ordered $QueQ$, s.t.*

- Rule 4.1*: Given a trajectory node E_T and a query q, $MINcd_{E_T\_E_{Q_{m-1}}} \leq MAXcd_{ET\_q}$ and $MINcd_{E_T\_E_{Q_m}} > MAXcd_{ET\_q}$. If $\sum_{i=1}^{m-1} |E_{Q_i}| < k$, E_T must be visited.*
- Rule 4.2*: Given a trajectory t and a query q, $MINcd_{t\_E_{Q_{m-1}}} \leq cd(t, q)$ and $MINcd_{t\_E_{Q_m}} > cd(t, q)$. If $\sum_{i=1}^{m-1} |E_{Q_i}| < k$, t is one of the RkNNST.*

Algorithm 5 shows the process that Rule 3.1 and Rule 4.1 are applied to check the state of a trajectory node E_T. Under the special case of E_T being a trajectory, Rule 3.2 are used to prune t and Rule 4.2 is employed to return t as the query result. The basic procedures are same. Thus, we only show the specific algorithm of checking the state of E_T.

In Algorithm 5, firstly, the number of queries num_1 in $QueQ$, whose maximum correlative distance is smaller than $MINcd_{ET\_q}$, and the number of queries num_2 in $E_T.cand_k$ whose maximum correlative distance is smaller than $MINcd_{ET\_q}$, are computed. If the sum of num_1 and num_2 is larger than k, E_T is pruned (Lines 1∼4). Then, the maximum correlative distance between q and ET is computed. The number of queries in the nodes whose minimum correlative distance is less than $MAXcd_{ET\_q}$ are computed from the queue $QueQ$. If the number of such queies is less than k, E_T must be visited (Lines 5∼10).

6 Experiments

In this section, the effectiveness and efficiency of our proposed algorithm are experimentally evaluated under various system settings. All the algorithms were implemented in C++, and were conducted on a PC with an Intel Core i5-3210M 2.5GHZ CPU and 12 GB RAM, running on WIN 7 desktop edition.

Algorithm 5 Checking the state of E_T

Input:E_T, $QueQ = (E_{Q_1}, E_{Q_2}, ..E_{Q_n})$ and q
Output: the state of E_T

1: num_1=the number of nodes in $QueQ$ whose max correlative distance is smaller than $MINcd_{ET\_q}$;
2: num_2=the number of queries in $E_T.cand_k$, whose max correlative distance is smaller than $MINcd_{ET\_q}$;
3: **if** $num_1 + num_2 \geq k$ **then**
4: **return** prune; {Rule 3}
5: compute $MAXcd_{ET\_q}$;
6: **for** each E_Q in $QueQ$ **do**
7: **if** $MINcd_{E_T\_E_Q} \leq MAXcd_{ET\_q}$ **then**
8: put E_Q into $SubQ$;
9: **if** $0 < \sum_{E_Q \in SubQ} |E_Q| < k$ **then**
10: **return** visit; {Rule 4}
11: **return** notsure;

6.1 Experimental setup

Since WIBR-tree is proved to scale well with the number of keywords for boolean top-k queries [2], we employ a WIBR-tree to index the queries points. We compare four algorithms, namely DOTA, NA, W and WI. DOTA, W and WI only differ on the index structure, whereas the query procedures are the same. DOTA is our proposed algorithm. NA is the naive method with indexes neither on trajectories nor on queries. W creates a WIBR-tree on the queries only, and WI uses a WIBR-tree on queries and a keyword inverted file on trajectories. The four algorithms use the same method to compute the minimum correlative distance. To the best of our knowledge, none of the existing algorithms considers reverse nearest neighbors on semantic trajectories. Hence, we do not include any existing algorithms for comparison. We will explore the average process time, the average IMC-tree size, and the effectiveness of pruning rules on different experimental settings.

We employ the real check-in data crawled form Foursquare within the areas of New York (NY) [1], [25]. In total, the dataset contains 424,649 checked-in points and 630,691 keywords. Each points of interested (POI) is associated with up to 18 keywords. The POIs checked-in by one user are ordered by the check-in timestamps. The total number of trajectories is 49,062. By default, $l = 6$ consecutive checked-in POIs are extracted to form the trajectory of this user. 5,000 trajectories are picked up randomly. Then, the number of points on the trajectories is 30,000. 5,000 queries are randomly picked from the checked-in POIs, that are not extracted as the positions in the trajectories. The default IMC-tree degree is 50. The default parameters are summarized in Table 2.

TABLE 2
Default system settings

parameter	default setting
number of points on trajectories	30,000
number of queries ($\|QS\|$)	5,000
number of results (k)	6
trajectory length (l)	6
IMC-tree degree	50

6.2 Efficiency Measurement

We compare the time costs of the four algorithms under different constraints, such as the number of results k, the

number of query keywords $|ws|$, the trajectory length l, the size of queries set $|QS|$ and the size of trajectory set $|TS|$.

Effect of k: First, we explore the effect of the required number of results k on the efficiency of the four solutions. As shown in Figure 6a, DOTA significantly outperforms all other three solutions. On average, DOTA is at least two orders of magnitude faster than NA and 19 times faster than W and WI. The performance difference between DOTA and WI (W) origins from the effect of IMC-tree and pruning rules. With the help of the IMC-tree and the pruning rules, a large number of trajectories under the trajectory node are pruned during the query process. As a result, the number of query index access decreases. The lines gap between NA and W(WI) shows the effect of query index. The performance of all algorithms degrades as k increases since more candidates need to be retrieved and refined. When k is between 10 and 30, the running times W and WI are close. When k increases to 40 or larger, WI is slighter faster than W with the benefit of the inverted index. However, the difference is hard to be noticed in Figure 6a. This phenomenon shows that the pruning effect of the keywords on the trajectories is limited. The time cost of query index access and computing the correlative distance dominate the performance of W and WI.

Effect of $|ws|$: Next, we expect to set the number of keywords in the queries at five ranks, i.e., [1-5], [2-6], [3-7], [4-8], and [5-9]. The effect of keyword number on the queries is validated. The number of queries for each type is 1000, then every rank has 5000 queries. However, From Figure 6c, 99% of POIs in the raw data set associates with less than 6 keywords. Thus, when the number of queries of a certain type is less than 1,000, we complement the queries by appending the most frequent keywords to the POIs with the closest number of keywords required iteratively. For instance, in order to get a query with 7 keywords, we add one most frequent keyword to the POI with 6 keywords.

From Figure 6b, DOTA has superior performance than the other three algorithms. The average running time of NA increases with the increase of $|ws|$. NA doesn't consist of any indexes nor pruning rules. Thus, the increased running time mainly results from computation the correlative distances. In contrast, the running time of DOTA, W and WI decreases as $|ws|$ increasing. That implies that the keywords pruning is more effective with large number of keywords. The performance gap between W and WI becomes obvious when $|ws|$ is large. In addition, we observe that the decrease rate of W and WI is larger than that of DOTA. As we know, WIBR-tree is created based on the keywords frequency. That phenomenon implies the the pruning effect of WIBR-tree is enhanced gradually with the increase of keywords in W and WI.

Effect of l: Then, we study the average running time when the trajectory length l varies from 3 to 15. Figure 6d shows that the running time for the four algorithms increases slightly. The order on the average running time is DOTA, WI, W and NA. In the four solutions, more minimum correlative sub-trajectories result from the increase in trajectory lengths. Thus, all solutions consume more time on computing the correlative distance. We still observe that the two lines of W and WI overlaps with each other. That is because the pruning effect of the inverted list on WI becomes not obvious with the increase of trajectory length, since a trajectory has more chance to cover the keywords in the query. For DOTA, since both the spatial distances between trajectories and queries are used in the various upper bounds, the upper bounds become loose with the trajectory being lengthened. Hence, the running time of DOTA increases with l. However, the running time of DOTA is 3.7s when l=15, while W and WI is about 100s.

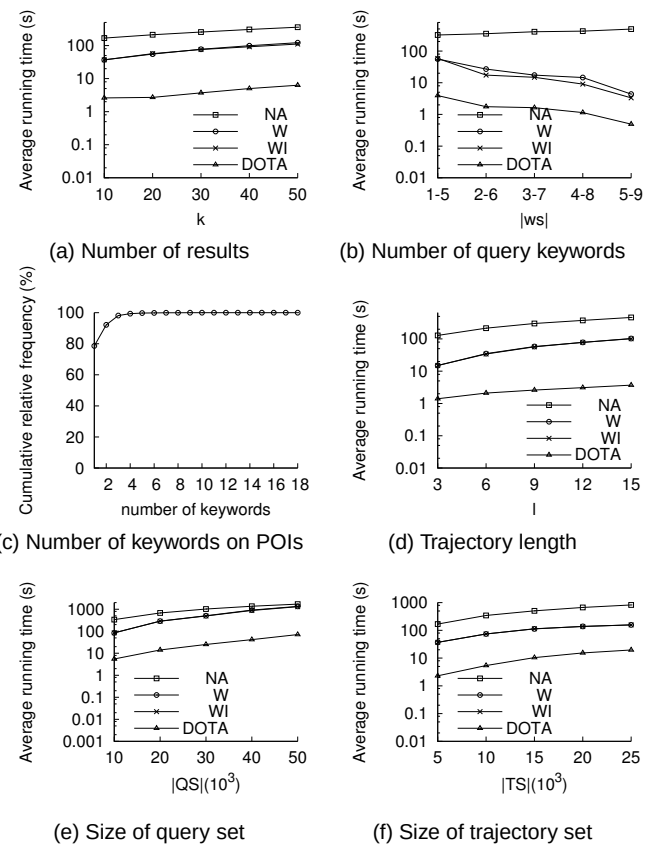
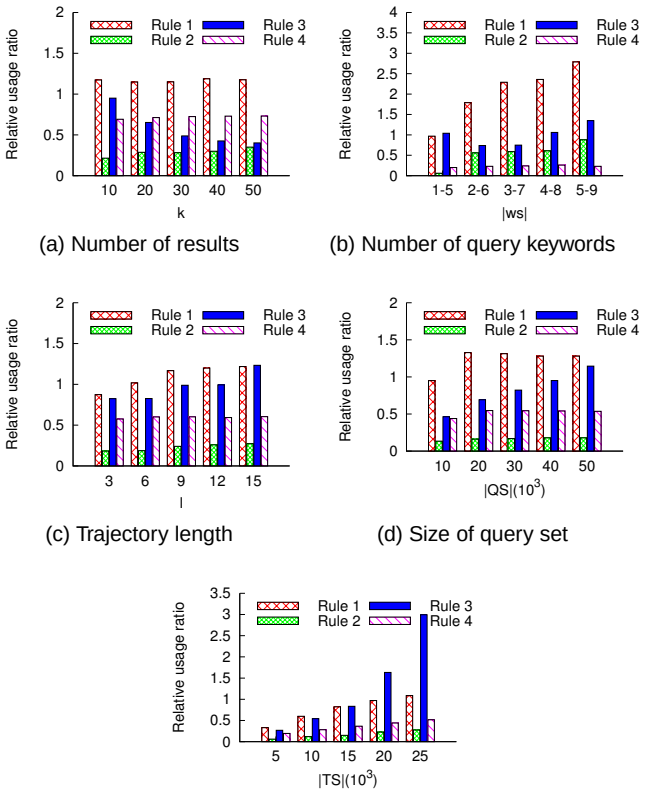


Fig. 6. Algorithm Efficiency

Effect of $|QS|$: We proceed to vary the size of query set $|QS|$ to see how the algorithms perform. The results are presented in Figure 6e. With the query size increasing, the average running times for all algorithms increase. That is because more queries have the chance to be in $ncn_k(t)$, and the average computation, pruning, and query index access time increase as well. W and WI are about 4 times faster than NA at $|QS| = 10,000$. With the performance of W and WI degrades, the running time gap between NA and W (WI) shortens to about 1.3 times. The performance of W and WI becomes worse when the query size is large. Comparing with W and WI, the good performance of DOTA is achieved by using the access rules and the batch process. The pruning rules enable to prune the non-candidate trajectories earlier on the IMC-tree, and meanwhile, to reduce the chance of access on the query index. Moreover, the pruning rules and visiting rules are beneficial for breaking off from the query index access easily. The batch process in Algorithm 4 finds the RkNNST on the IMC-tree and updating $t.cand_k$ directly, when the $cand_k$ of trajectories are fixed. As a result, the unnecessary query index access is avoided.

Effect of $|TS|$: Finally, we proceed to study the effect of

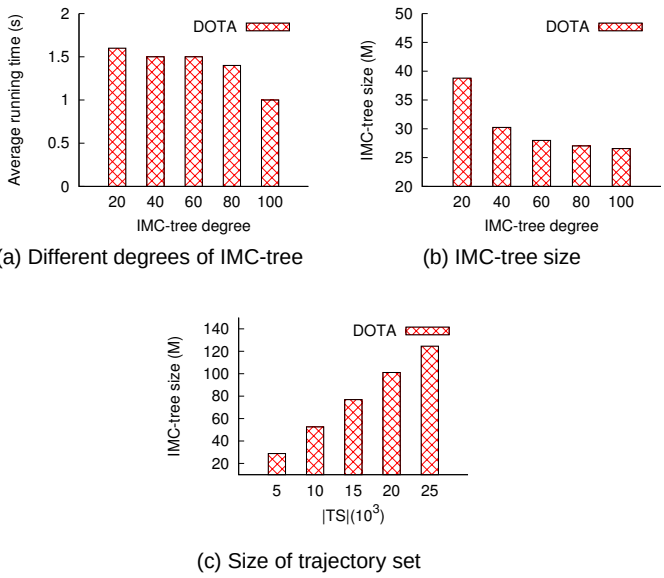


(a) Number of results (b) Number of query keywords

(c) Trajectory length (d) Size of query set

(e) Size of trajectory set

Fig. 7. Effectiveness of Access Rules



(a) Different degrees of IMC-tree (b) IMC-tree size

(c) Size of trajectory set

Fig. 8. Effectiveness of IMC-tree

the trajectory size. 5,000 to 25,000 trajectories are randomly picked from the original trajectory set respectively. Each trajectory size is 6. Thus, the total data size varies from 30,000 to 150,000. Figure 6f shows all methods increase linearly w.r.t $|TS|$. Although the trend line of NA appears to be parallel to the one of W (WI), the performance gap between NA and W(WI) becomes large with $|TS|$ increasing, since the y-axis is in the log scale. When comparing the performance trends of DOTA in Figure 6e and Figure 6f, we observe that DOTA is more sensitive to the number of queries than the number of trajectories. Most points of the performance line in Figure 6e is higher than the one in Figure 6f. That is, the pruning efficiency degradation of IMC-tree with the trajectory size increase is slower than the one of the query index with the query size increase. The average running time of DOTA is 19.6*s* when $|TS|$ is at 25,000.

6.3 Effectiveness Measurement

For effectiveness measurement, we will explore the impact of the rules and the size of the IMC-tree.

Effect of the rules: We tested the four rules (i.e., Rule 1, Rule 2, Rule 3 and Rule 4) over various settings. Recall that Rule 1 indicates the textual requirements testing, that only the trajectories contain all the keywords in the queries are retained. Rule 2 and Rule 3 take the use of the property of minimum correlative distances, maximum correlative distances, as well as the intermediate result in the trajectories. Rule 4 is the visiting rule, which can help breaking off from the query index access earlier, such that the location information in a trajectory node becomes specific with the nodes in IMC-tree being opened. For Rule 2, Rule 3, and Rule 4, two cases (i.e., the rule on a trajectory node and the rule on a trajectory) are combined to count since they are on the basis of the same idea. The y-axis in Figure 7 is the relative usage ratio for the different pruning rules, that is the division of the usage frequency by the third quartile[5] of the usage frequencies. A high relative usage ration implies that the rule is effective.

Figure 7 shows the effectiveness of different rules on various settings. From the overall perspective, the frequency of Rule 1 is highest at most cases. Rule 1 avoids the negative trajectory nodes being inserted into SET, and the number of query index access decreases directly. In Figure 7a we observe that the usage frequencies of Rule 2 and Rule 4 increase slightly with k increasing. The rise of Rule 2 is more obvious than Rule 4. As the number of required results increases, more candidate trajectories traverse over the query index. Hence, Rule 2.2 for a specific trajectory is invoked more frequently. The effectiveness of Rule 3 decreases with k increasing, which echoes the usage trend of Rule 2.2. Since trajectory nodes are not pruned by Rule 3, more specific trajectories under the nodes are retrieved. The increase in k has no effect on the usage of Rule 1.

In Figure 7b, Rule 1 is the most effective rule at all cases. As $|ws|$ increases, Rule 1 is used more frequently. More trajectory nodes which do not contain all the keywords in the query are pruned. Rule 3 is the second most frequently used rule. The number of Rule 3 usage is dominated mainly by the usage of Rule 3.2 (i.e., pruning a trajectory). From Figure 7b, the usage of Rule 3 decreases first, and then increases from [3-7]. When $|ws|$ is small (≤ 6), most of the negative trajectories are pruned by other rules. Hence, Rule 3 is invoked less. However, with $|ws|$ continuing to increase, since more queries are filled by the keywords with the highest frequency, more trajectory candidates appear. Since the specific correlative distance between a trajectory and a

5. The reason, that we don't use the maximum usage frequency, is to avoid the case that an extreme large maximum usage frequency overwhelms the effect of other pruning rules.

query can be computed, the usage of Rule 3.2 increases. Figure 7b shows the usage of Rule 4 remains stable. The usage of Rule 2 increases with $|ws|$ increasing, which mainly results from the growing usage of Rule 2.2 (i.e., pruning rule on a trajectory). The overall usage of the rules rises, which leads into running time reduction in Figure 6b.

Increasing trajectories length implies more trajectory candidates. Hence, Figure 7c shows that the usages of all rules expect Rule 4 increase slowly. The growth of Rule 1 and Rule 3 is most obvious. The usage of Rule 4 is relatively stable again. That is, increasing trajectory length cannot provoke the usage of visiting rule. The propriety order of rule usage in Figure 7d is same with the one in Figure 7c. That is, Rule 1 is the most effective one, and Rule 2 is the least used one. When $|QS|$ increases from 10,000 to 20,000, the usage ratio of all rules increases. Then, only Rule 3 continues to increase, and the usage of the other rules changes little when $|QS|$ become larger than 20,000. That is because we only access IMC-tree in the batch process, where Rule 3 is used frequently. From Figure 7e, we observe that the usages of most rules increase linearly, expect Rule 3 increases exponentially. When $|TS|$ is smaller than 15,000, Rule 1 is most effective. However, the usage of Rule 3 exceeds the usage of Rule 1 when $|TS|$ is 20,000 or larger, since more specific trajectory candidates trigger Rule 3.2.

IMC-tree size: Figure 8a shows the correlation between the IMC-tree degree and the average running time. As we expected, the average running time decreases with the IMC-tree degree increasing, since the height of IMC-tree decreases. Figure 8b shows the sizes of the indexes on the different IMC-tree degrees. When the IMC-tree degree is 20, the size of IMC-tree is 39.7MB. When the IMC-tree degree increases to 100, the size of IMC-tree decreases to 26.6MB. It is meaningless to increase IMC-tree degree blindly. We can observe that the downward tendency of the line becomes slow when the IMC-tree degree increases to 60 or larger. Since the size of IMC-tree is only affected by the size of trajectory data, we vary the trajectory size from 5,000 to 25,000. The memory size of IMC-tree increases linearly when the trajectory data size becomes large. When trajectory data size increases to 25,000, the size of IMC-tree is 124M, which is acceptable.

7 Related Work

Reverse nearest neighbor queries: RNN has been extensively studied under a variety of settings and many sophisticated algorithms have been proposed. The methods of RNN on moving objects can be classified into three categories:pre-computing, filtering-and-verification and branch-and-bound based algorithms [27]. Our proposed DOTA follows the branch-and-bound fashion. The lower and upper bounds for each intermediate node or an object are computed, such that a node will be decided whether the node is a result. The existing work for answering reverse spatial-keyword nearest neighbor queries [10], [12] follows this research line as well. The main difference between our problem and RNN is that our data sets contain trajectories (i.e., geo-textual object sequences) and points respectively. Thus, the existing methods on points data cannot be adapted to our problem. On the other hand, [23] investigates RNN on spatial trajectories and points. As we mentioned in Introduction, the problem definitions and the similarity measurements are totally different.

Indexes on semantics trajectories: There is a significant commercial and research interest in spatial keyword queries over semantic trajectories [17], [18], [19], [21], [34]. Similar to [2], we classify the existing indexes on semantic trajectories into three categories according to the combination scheme: loose combination, spatial-first combination, and text-first combination. Our IMC-tree falls into the last category. IOC-Tree proposed in [6] is the most similar index with our IMCtree. IOC-Tree is based on the inverted indexing and octrees. Trajectory points (i.e., local information) are kept on the leaf nodes of the octree, and the non-empty leaf nodes are organized by a dimensional structure. [8], [9] proposed two indexing structures, i.e., TSP-tree and ESP tree, supporting a STKP query. TSR is a trajectory-based semantic R-tree storing the global information of the trajectories; ESR is an episode-based semantic R-tree saving the local information of the trajectories. However, our proposed index IMC-tree fuses global and local information in "one" tree. The local correlative geo-texual objects are maintained as a trajectory abstract embedded in each internal node. Meanwhile, the spatial distances between two trajectories (i.e., the global spatial information) are maintained in our MC-trees.

8 Conclusion

In this paper, we investigated reverse nearest neighbors queries on semantic trajectories RkNNST, which fuses of spatial information and textual information on trajectories and queries. We showed that none of the existing work can effectively answer RkNNST as none of the indexes can combine both the global and local geo-texutal information at the same time. To address this problem, we propose a novel index IMC-tree incorporating the features of an inverted index and MC-trees. Moreover, we propose a branch-and-bound algorithm DOTA to prune the irrelevant trajectories effectively. A series of experiments has been conducted to evaluate DOTA under various system settings. The experimental results show that the IMC-tree size is about 29M, and the average processing time is about only 3s, which validate the efficiency and effectiveness of the proposed DOTA algorithm.

References

[1] J. Bao, Y. Zheng, and M. F. Mokbel, "Location-based and preference-aware recommendation using sparse geo-social networking data," *in Proc. of 20th Int. Conf. on Advances in Geographic Information Systems*, pp. 199-208, 2012.
[2] L. Chen, G. Cong, C. Jensen, and et al, "Spatial keyword query processing: an experimental evaluation," *in Proc. of 39th Int. Conf. on VLDB Endowment*, vol. 6, no. 3, pp. 217-228, 2013.
[3] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *in Proc. of 35th Int. Conf. on VLDB Endowment*, vol. 2, no. 1, pp. 337-348, 2009.
[4] P. Ciaccia, M. Patella; P. Zezula, "M-tree: an efficient access method for similarity search in metric spaces", *in Proc. of the 23rd VLDB Conference*, pp. 426-435, 1997.
[5] J. Dai, C. Liu, J. Xu, and Z. Ding, "On personalized and sequenced route planning," *World Wide Web*, vol. 19, no. 4, pp. 679-705, 2016.
[6] Y. Han, L. Wang, Y. Zhang, et al, "Spatial keyword range search on trajectories", *in Proc. of 20th Int. Conf. on DASFAA*, pp. 223-240, 2015.
[7] R. H. Güting, F. Valdés, and M. L. Damiani, "Symbolic trajectories". *ACM Trans. on Spatial Algorithms and Systems*, vol. 1, no. 2, pp. , 7:1-7:51, 2015.

[8] F. Gryllakis, N. Pelekis, C. Doulkeridis, and et al, "Spatio-temporal-keyword pattern queries over semantic trajectories with Hermes@Neo4j", *in Proc. of 22th Int. Conf. on EDBT*, pp. 678-681, 2018.
[9] F. Gryllakis, N. Pelekis, C. Doulkeridis, and et al, "Searching for spatio-temporal-keyword patterns in semantic trajectories", *In Proc. of 16th Int. Symposium on Intelligent Data Analysis*, pp. 112-124, 2017.
[10] Y. Lu, G. Cong, J. Lu, and et al, "Efficient algorithms for answering reverse spatial-keword nearest neighbor queries," *in Proc. of the 23rd SIGSPATIAL*, pp. 1-4, 2015.
[11] Y. Li, R. Chen, L. Chen, and et al, "Towards social-aware rideshare group query services", *IEEE Trans. on Services Computing*, vol. 10, no. 4, pp. 646-659,2017.
[12] J. Lu, Y. Lu, and G. Cong, "Reverse spatial and textual k nearest neighbor search," *in Proc. of SIGMOD/PODS*, pp. 349-360, 2011.
[13] L. Li, K. Zheng, and et al, "Go slow to go fast: minimal on-road time route scheduling with parking facilities using historical trajectory". *VLDB Journal* vol. 27, no. 3, pp. 321-345, 2018.
[14] G. Liu, K. Zheng, and et al, "MCS-GPM: multi-constrained simulation based graph pattern matching in contextual social graphs", *IEEE Trans. Knowl. Data Eng*. vol.30, no. 6, pp. 1050-1064, 2018.
[15] D. Lian, K. Zheng, Y. Ge, et al, "GeoMF++: scalable location recommendation via joint geographical modeling and matrix factorization", *ACM Trans. Inf. Syst*. vol.36, no.3, pp 33:1-33:29, 2018.
[16] D. Pfoser, C. Jensen, and Y. Theodoridis, "Novel approaches to the indexing of moving object trajectories," *in Proc. of 26th Int. Conf. on VLDB*, pp. 395-406, 2000.
[17] C. Parent, S. Spaccapietra, and et al, "Semantic trajectories modeling and analysis," *ACM Computing Surveys*, vol. 45, no.4, article 42:1-42:32, 2013.
[18] N. Pelekis, S. Sideridis, and Y. Theodoridis, "Hermessem: a semantic-aware framework for the management and analysis of our LifeSteps," *in Proc. of Int. Conf. on DSAA*, pp.,1-10, 2015.
[19] N. Pelekis, Y. Theodoridis. "Mobility data management and exploration", *Springer*, New York, 2014.
[20] S. Su, Y. Teng, X. Cheng, and et al, "Privacy-preserving top-k spatial keyword queries in untrusted cloud environments", *IEEE Trans. on Services Computing*, vol. 11, no. 5, pp. 796-809,2018
[21] S. Sideridis, N. Pelekis, and Y. Theodoridis, "On querying and mining semantic-aware mobility timelines", *Data Science and Analytics*, vol. 2, no. 1-2, pp. 29-44, 2016.
[22] Y. Tao, and D. Papadias, "Mv3r-tree: A spatio-temporal access method for timestamp and interval queries," *in Proc. of 27th Int. Conf. on VLDB*, pp. 431-440, 2001.
[23] S. Wang, Z. Bao, J.S. Culpepper, T. Sellis, and Gao Cong, "Reverse k nearest neighbor serach over trajectories", *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 4, pp. 757-771, 2018.
[24] D. Wu, M. Yiu, G. Cong, and et al, "Joint top-k spatial keyword query processing", *IEEE Trans. on Knowl. and Data Eng.*, vol. 24, no. 10, pp. 1889-1903, 2012.
[25] L. Wei, Y. Zheng, and W.C. Peng, "Constructing popular routes from uncertain trajectories," *in Proc. of 18th SIGKDD*, pp. 195-203, 2012.
[26] J. Xu, H. Lu, and et al, "Range queries on multi-attribute trajectories", *IEEE Trans. on Knowl. and Data Eng.*, vol. 30, no. 6, pp. 1206-1211, 2018.
[27] S. Yang, M. A. Cheema, and et al, "Reverse k nearest neighbors query processing: experiments and analysis", *in Proc. of 40th Int. Conf. on VLDB Endowment*, vol. 8, no. 5, pp. 605-616, 2014.
[28] K. Zheng, S. Shang, and et al, "Towards efficient search for activity trajectories", *in Proc. of IEEE 29th ICDE*, pp. 230-241, 2013.
[29] D. X. Zhang, K. L. Tan, and et al, "Scalable top-k spatial keyword search", *in Proc. of 16th Int. Conf. on EDBT*, pp. 359-370, 2013.
[30] B. Zheng, N. Yuan, K. Zheng, and et al, "Approximate keyword search in semantic trajectory database", *in Proc. of IEEE 31th ICDE*, pp. 975-986, 2015.
[31] C. Zhang, Y. Zhang, W. Zhang, and et al, "Inverted linear quadtree:efficient top k spatial keyword search", *in Proc. of IEEE 29th ICDE*, pp. 901-912, 2013.
[32] B. Zheng, H. Su, W. Hua, and et al, "Efficient clue-based route search on road networks", *IEEE Trans. Knowl. Data Eng.*, vol. 29, no.9, pp. 1846-1859, 2017.
[33] Y. Zhao, K. Zheng, Y. Li, and et al, "Destination-aware task assignment in spatial crowdsourcing: a worker decomposition approach", *IEEE Trans. Knowl. Data Eng.*, Accepted in June 2019.
[34] K. Zheng, Y. Zhao, D. Lian, et al, "Reference-based framework for spatio-temporal trajectory compression and query processing", *IEEE Trans. Knowl. Data Eng.*, Accepted in April 2019.

Xiao Pan is an associate professor in Shijiazhuang Tiedao University. She was a visiting scholar in the Department of Computer Science, University of Illinois at Chicago, USA. She received her Ph. D. degree from Renmin University of China. Her research interests include data management, mobile computing and privacy-aware computing. She is a member of the Database Society of the China Computer Federation (CCF DBS).

Shili Nie is a currently master student in the School of Information Science and Engineering, Yanshan University. His research interests include data management on moving objects and trajectory computing.

Haibo Hu is an associate professor in the Department of Electronic and Information Engineering, Hong Kong Polytechnic University. His research interests include information security, privacy-aware computing, wireless data management, and location-based services. He has published over 70 research papers in refereed journals, international conferences, and book chapters. As principal investigator, he has received over 9 million HK dollars of external research grants from Hong Kong and mainland China.

Philip S. Yu is a Distinguished Professor in Computer Science at the University of Illinois at Chicago. His research interest is on big data, including data mining, database and privacy. He has published more than 920 papers in refereed journals and conferences. He holds or has applied for more than 250 US patents. Dr. Yu is a Fellow of the ACM and the IEEE. He is the Editor-in-Chief of ACM Transactions on Knowledge Discovery from Data. He is on the steering committee of the IEEE Conference on Data Mining and ACM Conference on Information and Knowledge Management and was a member of the IEEE Data Engineering steering committee. He was the Editor-in-Chief of IEEE Transactions on Knowledge and Data Engineering (2001-2004).

Jingfeng Guo is a Professor in the School of Information Science and Engineering, Yanshan University. He received the Bachelor, Master and Ph. D. degree from Yanshan University. His research interests include database theory and application, data mining and social network analysis, etc. He is a member of the Database Society of the China Computer Federation (CCF DBS). He has published more than 100 technical papers in refereed international journals and conference proceedings.