Spatial Keyword Range Search on Trajectories

Yuxing Han[1(✉)], Liping Wang[1], Ying Zhang[2], Wenjie Zhang[3], and Xuemin Lin[1,3]

[1] Shanghai Key Lab for Trustworthy Computing, East China Normal University, Shanghai, China
yxhan@student.ecnu.edu.cn, lipingwang@sei.ecnu.edu.cn
[2] University of Technology, Sydney, Australia
[3] The University of New South Wales, Sydney, Australia
Ying.Zhang@uts.edu.au, {zhangw,lxue}@cse.unsw.edu.au

Abstract. With advances in geo-positioning technologies and ensuing location based service, there are a rapid growing amount of trajectories associated with textual information collected in many emerging applications. For instance, nowadays many people are used to sharing interesting experience through Foursquare or Twitter along their travel routes. In this paper, we investigate the problem of spatial keyword range search on trajectories, which is essential to make sense of large amount of trajectory data. To the best of our knowledge, this is the first work to systematically investigate range search over trajectories where three important aspects, *i.e.*, spatio, temporal and textual, are all taken into consideration. Given a query region, a timespan and a set of keywords, we aim to retrieve trajectories that go through this region during query timespan, and contain all the query keywords. To facilitate the range search, a novel index structure called IOC-Tree is proposed based on the inverted indexing and octree techniques to effectively explore the spatio, temporal and textual pruning techniques. Furthermore, this structure can also support the query with order-sensitive keywords. Comprehensive experiments on several real-life datasets are conducted to demonstrate the efficiency.

1 Introduction

The proliferation of GPS-enabled devices such as smartphones and the prosperity of location-based service have witnessed an unprecedented collection of trajectory data. Latest work on spatio-temporal trajectories includes travel time estimation [21], trajectory compression [18], route recommendation [19], frequent path finding [12], etc. In addition to spatio-temporal trajectory, semantic trajectories [2] which combine textual information with each spatial location also attract great research attention in recent years. A large amount of semantic trajectories are generated from location-based social networking services (LBSNs), such as Foursquare and Twitter. Representative work includes pattern mining [22] and activity trajectories search [23].

M. Renz et al. (Eds.): DASFAA 2015, Part II, LNCS 9050, pp. 223–240, 2015.
DOI: 10.1007/978-3-319-18123-3_14

Motivation. While significant efforts have been devoted to exploiting trajectory dataset, to the best of our knowledge, none of the existing work considers three critical aspects (*spatio, temporal, textual*) of the trajectory data at the same time during the range search processing. Previous studies either consider the spatio-temporal properties (e.g., [4,16]), or only explore spatial and textual aspects (e.g., [6,23]) of the trajectories. However, we stress that, in many real-life scenarios, three aspects of the trajectories are considered by users at the same time to retrieve desirable results. In particular, users may only be interested in the trajectory points (e.g., activities) within a particular region (e.g., a nearby area or a suburb) during a time period (say, last week or recent 10 days). Meanwhile, as keywords tagged on the trajectory points carry rich information such as activities involved and users' personal experiences, it is natural to choose the trajectories based on query keywords.

Motivated by the above facts, in this paper we study the problem of **S**patial **K**eyword **R**ange search on **T**rajectories ($SKRT$) which retrieves meaningful trajectories based on the spatio-temporal and keyword constraints. Specifically, the spatio-temporal constraint is captured by a spatial region and a timespan while a set of query keywords is used to express user's interests. A trajectory will be retrieved if the trajectory points satisfying spatio-temporal constraint contain all query keywords. Below is a motivating example.

Example 1. A travel recommendation system has collected a number of user trajectories as shown in Fig 1. Suppose a tourist wants to plan a trip in a nearby region (dotted circle with Fig 1) to enjoy wonderful local *flower* and *pizza*. Undoubtedly, it is beneficial to the tourist if the system can provide some relevant trajectory data in the last one month. In the example, trajectory R_3 is not the appropriate candidate since there is no keyword *pizza* in the query range although it does contain pizza. Similarly, R_4 does not satisfy keyword constraint because *pizza* is not covered. Therefore, only R_1 and R_2 are retrieved for this tourist.

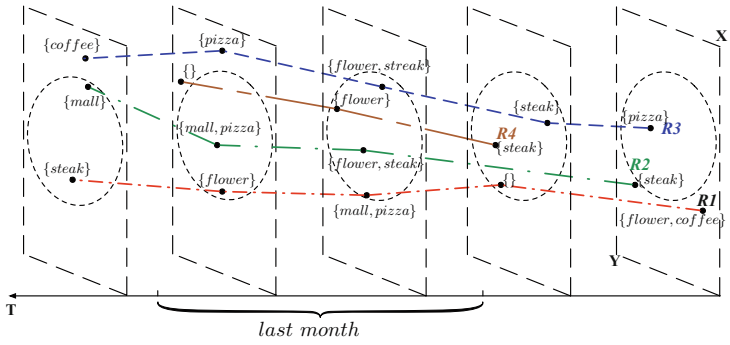


Fig. 1. Motivation Example

Challenges. In many applications, the number of trajectories might be massive, calling for efficient indexing techniques to facilitate trajectory range search proposed in this paper. The key challenge is how to effectively combine the spatio, temporal and keywords features of the objects such that a large number of non-promising trajectories can be pruned. As observed by Christoforaki et al. [5], the number of keywords issued from users in real life is generally small, typically 2-5, and hence higher priority is given to keyword feature. In this paper, we adopted ***inverted index*** technique and hence the query processing follows a keyword-first-pruning strategy. Only a few relevant trajectories containing query keywords will be loaded during query processing. Another advantage of inverted index is that the query with order-sensitive keywords can be naturally supported by choosing inverted lists in order. Regarding each keyword (i.e., inverted list), we also need effective spatial-temporal index structure to organize related trajectory points. As we observe that the spatial and temporal distributions of the real-life trajectories might be highly skewed for each keyword, we use *octree* [14] to organized trajectory points in each inverted list. Note that octree is a three-dimensional analog of quadtree [8], which is self-adaptive to the spatial and temporal distributions of the trajectory points. As the nodes of octrees can be encoded based on *morton code* [15], thus two nodes with higher spatio-temporal proximity are likely to be assigned to the same page in the secondary storage. As a result, the number of I/Os during query processing could be reduced due to principle of locality. Furthermore, to expedite the process of spatio-temporal search, *signature* technique is utilized to prune non-promising trajectories without loading trajectory points resident on the disk.

Contribution. Our main contribution can be summarized as follows:

- This is the first work to investigate the problem of spatial keyword range search where both spatio-temporal and keyword constraints are considered.
- We proposed a novel structure, namely IOC-Tree, to effectively organize trajectories with keywords.
- We also proposed an efficient algorithm to process spatial keyword range search on trajectories.
- Comprehensive experiments on real-life datasets demonstrate the efficiency of our techniques.

Organization. The remainder of the paper is organized as follows: Section 2 gives a formal problem definition and the related work is also reported. Section 3 presents the IOC-Tree structure. Efficient spatial keyword range search algorithm is proposed in Section 4, followed by experimental evaluation in Section 5. We conclude the paper in Section 6.

2 Preliminary

In this section, we first provide a formal definition of problem we study in this paper, then give a brief review of related work. Table 1 summarizes the notations used throughout the paper.

Table 1. List of Notations

Notations	Explanation
Tr	a trajectory with keywords
$SubTr(i,j)$	a sub-trajectory of Tr
$\psi(Q.R)$	the diameter of query range
$\mathbb{V}$	keyword vocabulary
w	a keyword in $\mathbb{V}$
h	maximal depth of IOC-Tree
ψ	split threshold of a node in IOC-Tree
m	number of query keywords

2.1 Problem Description

In this paper, a *trajectory* Tr is represented as a time-ordered sequence of location points with keywords: $\{(t_1,p_1,\phi_1),\ (t_2,p_2,\phi_2),...,(t_n,p_n,\phi_n)\}$, where t_i is the timestamp, p_i is the location comprised of *latitude* and *longitude*, ϕ_i is the set of keywords.

Definition 1. *A* ***sub-trajectory*** $\{(t_i,p_i,\phi_i),\ (t_{i+1},p_{i+1},\phi_{i+1}),...,(t_j,p_j,\phi_j)\}$ *where* $1 \le i \le j \le n$*, is a part of a trajectory. We denote above sub-trajectory as* $SubTr(i,j)$*.*

A sub-trajectory is a consecutive part of a trajectory and it can have only one point. The concept of sub-trajectory is given because one trajectory may enter and leave a particular area multiple times.

Definition 2. ***Spatial Keyword Range search on Trajectories*** *(SKRT) Q consists of a spatial* ***region*** *R, a* ***timespan*** *$T = [t_s,t_e]$ and a set Φ of* ***keywords*** *$(= \{k_1,k_2,...,k_m\})$. We call a trajectory Tr* ***satisfies*** *query Q if we could find sub-trajectories of Tr, $SubTr(i_1,j_1)$, $SubTr(i_2,j_2)$, ..., $SubTr(i_t,j_t)$, which locate within region during query timespan $[t_s,t_e]$, and collectively contain query keywords,* i.e.*, $\Phi \subseteq (\cup_{x=i_1}^{j_1}\phi_x) \cup (\cup_{x=i_2}^{j_2}\phi_x)... \cup (\cup_{x=i_t}^{j_t}\phi_x)$.*

Essentially, $SKRT$ consists of three constraints: spatial constraint R, temporal constraint T, and keyword constraint Φ. A trajectory will be retrieved if the trajectory points within the spatio-temporal range (i.e., satisfying both spatial and temporal constraints) cover all the query keywords. Following is an example of $SKRT$ based on trajectories in Fig 2.

Example 2. In Fig 2, there are four trajectories R_1, R_2, R_3, R_4 with 17 points where p_{ij} represents the point in R_i with its timestamp t_j. Notice that in practical applications, points of trajectories are usually not collected at the same timestamp. Assume an $SKRT$ Q is given as follows: $Q.R$ is the space within dotted circle depicted in Fig 2, $Q.T$ is $[t_s,t_e]$ where $t_1 < t_s < t_2$, $t_3 < t_e < t_4$, and $Q.\Phi = \{a,c\}$.

Fig 2 shows that trajectory R_1 and R_2 are the results returned by $SKRT$, for R_3 is eliminated due to spatio-temporal constraint and R_4 are eliminated due to keyword constraint.

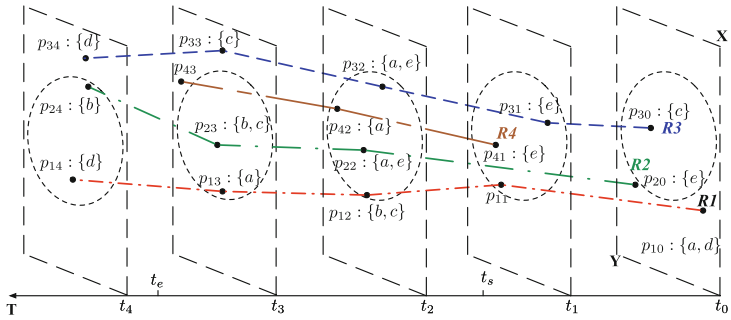


Fig. 2. Example of $SKRT$

Problem Statement. Given a database D of trajectories and an $SKRT$ query Q, we aim to retrieve all of the trajectories which ***satisfy*** the query Q from D.

2.2 Related Work

To the best of our knowledge, there is no existing work studying the problem of $SKRT$ proposed in this paper. Below, we introduce two categories of closely related existing work.

Spatial Keyword Search. Due to huge amounts of spatio-textual objects collected from location-based services, Spatial Keyword (SK) search has been widely studied. One of the most important queries is the top-k spatial keyword search, which aims to find k objects which have the most spatial proximity and contain all the query keywords. Many efficient index structures have been proposed such as inverted R-tree [24] and IR^2-tree [7]. In addition to top-k spatial keyword search, many interesting query variants are proposed such as direction-aware spatial keyword search (DESKS) [9] and collective spatial keyword search (CoSKQ) [11]. Nevertheless, the temporal information is not considered in the above work. Recently, some recent work on SK search also consider the temporal constraint. In [13], Magdy et al. proposed a system called *Mercury* for real-time support of top-k spatio-temporal queries on microblogs, which allow users to browse recent microblogs near their locations. In [17], Skovsgaard et al. proposed an index structure that extends existing techniques for counting frequent items in summaries and a scalable query processing algorithm to identify top-k terms seen in the microblog posts in a user-specified spatio-temporal range. However, these techniques are especially designed for spatio-textual objects which are inherently different from the trajectory data.

TkSK [6] and ATSQ [23] are the two most relevant work to our problem. The TkSK proposed by Cong et al. [6] is comprised of a user location and a keyword set, and returns k trajectories whose text descriptions covering the keyword set with the shortest match distance. The authors developed a hybrid index called B^{ck}*-tree* to deal with text relevance and location proximity between the query

and trajectories. The ATSQ studied by Zheng et al. [23] finds k distinct trajectories have the smallest minimum match distance with respect to query locations with their query activity. A hybrid grid index called GAT was proposed to prune the search space by location proximity and activity containment simultaneously. As shown in our empirical study, although we can extend the above techniques to support temporal pruning by further organize the trajectory points with B+ tree according to their timestamps, the performance is not satisfactory.

Historical Spatio-Temporal Trajectory Indexing. There has been considerable related work on storing and querying historical spatio-temporal trajectories. Prior work proposed TB-tree [16] to solve the problem of range query over spatio-temporal trajectories. The main idea of TB-tree indexing method is to bundle segments from the same trajectory into leaf nodes of the R-tree. The MV3R-tree [20] is a hybrid structure that uses a multi-version R-tree (MVR-tree) for time-stamp queries and a small 3D R-tree for time-interval queries. This structure has been proved to outperform other historical trajectory index structures.

SETI [4] is a grid-based index which partitions the spatial space into grids and then index temporal information within each spatial partition based on R*-tree. PIST [3] is also grid-based which focuses on indexing trajectory points. It utilizes a quad-tree like data structure to partition points into a variable-sized grid according to the density of data. Since in our problems, trajectories have extra textual data than traditional ones, structures mentioned above can't be straightforwardly extended to solve our problem.

3 Inverted Octree

In this section, we introduce a new indexing structure, namely Inverted Octree (IOC-Tree), to effectively organize the trajectories with textual information. Section 3.1 provides the overview of our IOC-Tree structure. Section 3.2 introduces the detailed data structure, followed by the index maintenance algorithms in Section 3.3.

3.1 Overview

As discussed in Section 1, we follow the keyword-first-pruning strategy because we observe that the query keyword usually has the lowest selectivity (i.e., highest pruning capability) compared with spatial and temporal factors. Therefore, we adopted ***inverted index*** technique such that only the trajectory points associated with at least one query keyword will be involved in the range search. For each keyword in the vocabulary, a corresponding octree is built to organize the relevant trajectory points. The spatio-temporal space (i.e., 3-dimensional space) is recursively divided into cells (nodes) in an adaptive way, and trajectory points are kept on the leaf nodes of the octree. Moreover, we apply the *3D morton code* technique to encode the leaf nodes of the octree, and the non-empty leaf nodes are organized by an auxiliary disk-based one dimensional structure (e.g., B+ tree) where the morton code is the key of the nodes. In this way, trajectory

points with high spatio-temporal proximity are likely to be resident in the same page on the disk. We also employ the *signature* technique to keep trajectory identification information at high level node so that some non-promising trajectories can be pruned earlier without invoke extra I/O costs. For the purpose of verification, we also keep the exact trajectory information for each non-empty leaf node of octrees.

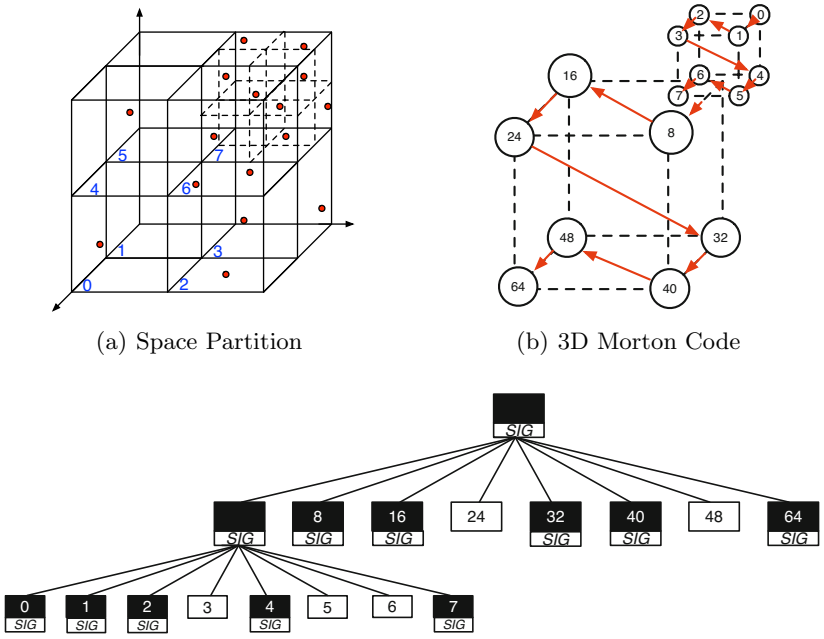
3.2 IOC-Tree Structure

In our IOC-Tree, we have an octree OC_w for each keyword $w \in \mathbb{V}$ [1]. In octree, each non-leaf nodes have eight children, and a simple illustration of octrees is depicted in Fig 3(a). Construction of an octree starts from treating the whole spatio-temporal three-dimensional space as a root node, and then recursively divides space into eight subspaces as children nodes if the node has sufficient objects, *i.e.*, spatio-temporal points. As shown in Fig 3(a), the space is first partitioned into 8 nodes, and only one of them is further divided. In this way, the octree can effectively handle the skewness of the trajectory points.

Next, we will explain how to generate morton code for each leaf node of octree. In [15], a 3-dimensional space will be recursively divided into 8^{h-1} cells where h is the depth of the partition, and the morton code of each cell is assigned based on its visiting order. Nevertheless, in this paper, the octree is constructed in an adaptive way and hence the leaf nodes may resident on different levels. Thus, the morton code of each leaf node is assigned as follows. Let h denote the maximal height of the octree, we assume the space is evenly partitioned into 8^{h-1} *virtual cells* and the morton code of each virtual cell is immediate. Then the morton code of a leaf node v corresponds to the minimal code of virtual cells covered by v. Fig 3(b) illustrates an example where the maximal depth of the octree is 3. Each circle represents a leaf node of the octree built from Fig 3(a) and each of them is assigned a morton code according to our approach. The morton code of Node #24 is 24 as it is the smallest codes in the virtual cells it contains. We remark that we do not need to materialize the virtual cells in our implementation, the morton code can be assigned based on its split sequence. Details are omitted due to space limitation.

In order to efficiently prune non-promising nodes when searching octrees, for each node v, we also maintain a *signature* to summarize the identifications of a set of trajectories that go through the corresponding spatio-temporal region of v. In particular, a signature of a node v is a bit vector, and each trajectory ID will be mapped onto one bit by a hash function. Then its i-th bit will be set 1 if there *exists* a trajectory point within in v (i.e., point satisfying spatio-temporal constraint regarding v) and its ID is mapped to the i-th position. Otherwise, the i-th bit is set to 0. As shown in Section 4.1, the non-promising trajectories may

[1] Note that as the frequencies of the keywords follow the *power-law* distribution in real-life, and there is a large portion of low frequency keywords. In our implementation, we simply keep trajectory points of low frequency keyword in one disk page, instead of building the related octree.



(a) Space Partition (b) 3D Morton Code

(c) IOC-Tree Structure

Fig. 3. A simple example of IOC-Tree

be pruned at high level of octree with the help of node signatures, and hence significantly reduce the I/O costs. Note that although the number of trajectory points is very huge, the number of trajectories is usually one or two orders of magnitude smaller, which makes the storage of *signature* feasible.

Fig 3(c) demonstrates the structure of an octree built from Fig 3(a), where nodes that contain trajectory points (i.e., non-empty nodes) is set *black* and the rest nodes (empty nodes) are set *white*. Each leaf node is labeled by its morton code, and a signature is maintained to summarize the trajectory IDs within the node. In addition to the octree structure, we also keep the trajectory points in each black leaf node. Each trajectory point is recorded as a tuple $(pID, tID, lat, lng, time)$, where pID is the point ID, tID is corresponding trajectory ID, and $(lat, lng, time)$ is the spatio-temporal value of this point. The non-empty leaf nodes from octrees will be kept on the disk by one dimensional index structure (e.g., B+ tree), which are ordered by their morton codes. However, *signature* only keeps a rough description for trajectories and will only be helpful for the pruning non-promising nodes. Therefore, for each leaf nodes of octrees, the exact information to explain the trajectories that they contain (i.e., trajectory IDs) need to be kept on the disk respectively. These information will also be organized by a B+ tree with the morton code of the corresponding leaf node as the key.

3.3 IOC-Tree Maintenance

The insertion of a trajectory includes two steps. Firstly for every point p from a trajectory Tr, it will be assigned into the corresponding octrees based on the keywords it contains. A leaf node of the octrees will be split if it contains more than ψ points and does not reach the maximal depth h. Meanwhile, for every octree node along the inserting path of p, the bit of its *signatures* mapped by Tr will be set to 1. As to deletion of a particular trajectory, we should remove all its points from their corresponding octrees along with some possible merges of the nodes. A bit in signature of a node from octrees will be reset correspondingly. Moreover, for leaf nodes in both cases of insertion and deletion, the related exact trajectory information on the disk also need to be updated.

Table 2. Distribution of Trajectory Points

Node#	0	1	2	3	4	5	6	7
Points	p_{11}	p_{10}	p_{30}	p_{20}	p_{23}	p_{12}	p_{22}	p_{42}
		p_{31}	p_{41}		p_{24}	p_{13}	p_{32}	p_{43}
					p_{33}	p_{14}	p_{34}	

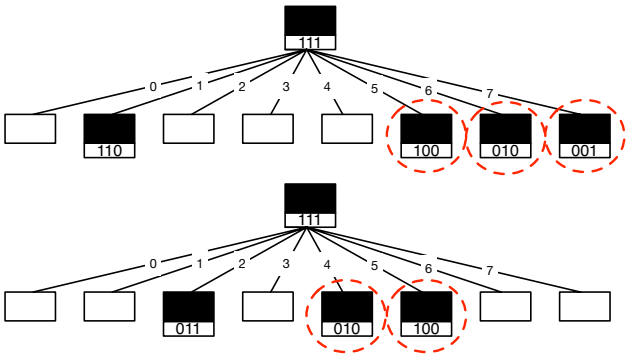


Fig. 4. IOC-trees' Construction

Example 3. For the sake of brevity, we build 2-level octrees regarding the case in Fig 2, where 17 trajectory points are distributed in different subspaces as shown in Table 2. Assume that each *signature* has three bits, and trajectory R_1 is mapped to the first position, R_2 and R_3 are mapped to the second position, R_4 is mapped to the third position. Fig 4 demonstrates the inverted octrees OC_a and OC_c where a node is set black if it contains points and white otherwise. The *signatures* are only kept for black nodes. As a matter of fact, white nodes are not reserved in our implementation. As an illustration of *signature*, for *node*#1 in OC_a, its *signature* is 110 because only trajectories R_1 (p_{10}) and R_3 (p_{31}) go through the corresponding region. Notice the exact trajectory information for each non-empty leaf nodes are stored on the disk and they are not shown in Fig 4.

4 Algorithm for Query Processing

In this section, we present efficient algorithms for processing $SKRT$ and one of its variant query with the assumption that trajectory data are organized by IOC-Tree. Section 4.1 will give a specific description of algorithm for $SKRT$. As an extension, a variant query of $SKRT$ called $SKRTO$ and its brief processing algorithm will be introduced in Section 4.2.

4.1 Algorithm for Processing *SKRT*

In our algorithm, octree nodes are divided into three types with regard to spatio-temporal query range: one locate outside the range, one locate totally in the range which are called *fully-covered* nodes, and the other intersect with range which are called *partially-covered* nodes. we employ a set L to keep octree nodes that are being processed. Moreover, sets of candidate nodes CN_i^f and CN_i^p are employed to record *fully-covered* and *partially-covered* candidates from related octree OC_i respectively. $\mathcal{TR}(CN)$ denotes a set of trajectories that the nodes in CN contains. Besides, CT^f contains candidate trajectories that appear in the *fully-covered* nodes for all the query keywords, while CT^p contains the rest candidate trajectories based on sets of candidate nodes.

Algorithm 1. Algorithm Outline For $SKRT$

Input: Q: an $SKRT$ query with three constraints ($Q.R$, $Q.T$, $Q.\delta$), OC: inverted octree, D: trajectory database
Output: $\mathcal{A}$: set of trajectories from D satisfy Q

1 $\mathcal{A} \leftarrow \emptyset$; $L \leftarrow \emptyset$; $CT^f \leftarrow \emptyset$; $CT^p \leftarrow \emptyset$;
2 **foreach** $k_i \in Q.\delta$ **do**
3 put $root(OC_i)$ into L;
4 $CN_i^f \leftarrow \emptyset$; $CN_i^p \leftarrow \emptyset$;
5 ***Prune***(Q, L);
6 $CT^f \leftarrow \cap_{i=1}^{m} \mathcal{TR}(CN_i^f)$; $CT^p \leftarrow \cup_{i=1}^{m} \mathcal{TR}(CN_i^p) + \cup_{i=1}^{m} \mathcal{TR}(CN_i^f)$;
7 $CT^p \leftarrow CT^p - CT^f$;
8 ***Verification***$(Q, \mathcal{A})$;
9 return $\mathcal{A}$;

The basic outline of algorithm for processing $SKRT$ is illustrated in Algorithm 1. The main idea is to prune as many trajectories as possible based on the spatio, temporal and textual information by using IOC-Tree structure. In Line 3, the root nodes of related octrees are put into the set L. Sets of different types of candidate nodes for each query keywords are initialized in Line 4. The next step is to explore the nodes in L to prune nodes that does not satisfy spatio-temporal constraint (Line 5). After that, we determine the different sets of candidate trajectories based on the exact trajectory information (Line 6-7). Finally, we validate each candidate trajectory and put right ones into the result set $\mathcal{A}$ (Line 8).

We proceed to give more details of procedures invoked in Algorithm 1.

Procedure *Prune* prunes non-promising nodes based on the inverted octree and deals with nodes level by level among the related octrees. In each level, nodes that don't satisfy spatio-temporal constraint will be firstly pruned by *STRange-Filter*. During the process of *STRangeFilter*, we only explore *black* nodes (i.e., non-empty nodes) and keep a one-bit flag for each node that satisfy spatio-temporal constraint to indicate a *fully-covered* or *partially-covered* node. Line 3-7 perform a *signature* test to prune nodes that definitely share no trajectories with nodes from other octrees. Among the nodes that survive from the *signature* test, leaf nodes will be inserted in the corresponding sets of candidate nodes (Line 8-13), and the non-*white* children nodes of non-leaf nodes will be put into L (Line 14-17). The last step is to retrieve candidate trajectories from the disk for each query keyword based on different types of candidate nodes (Line 19-20).

Procedure *Verification* aims at further validating candidate trajectories and inserting trajectories that pass all the tests into result set $\mathcal{A}$. Firstly, we believe all the trajectories in CT^f are appropriate ones which can be easily proved (Line 1). For each trajectory Tr in CT^p, a set Ω of query keywords that Tr doesn't have within *fully-covered* nodes are identified (Line 2-4). Then according to each keyword k_j in Ω, only if Tr appears in $\mathcal{TR}(CN_j^p)$ will it be loaded from the corresponding cell on the disk to verify spatio-temoral constraint (Line 5-7).

Example 4. Consider the $SKRT$ problem given in Fig 2. Since the query keyword set $Q.\phi$ contains two keywords a and c, only OC_a and OC_c which we have built in Fig 4 need to be explored. The nodes marked with dotted circle signify the ones within query spatio-temporal range in both trees. *node*#7 in OC_a is pruned because it does not pass the *signature* test. According to containment relationship with query range, after procedure *Prune*, we have $CN_a^f = CN_c^f = \{node\#5\}$, $CN_a^p = \{node\#6\}$ and $CN_c^p = \{node\#4\}$. Table 2 shows that only trajectory R_1 goes through *node*#5, and trajectory R_2 and R_3 go through *node*#4 and *node*#6. Therefore, $\mathcal{TR}(CN_a^f) = \mathcal{TR}(CN_c^f) = \{R_1\}$ and $\mathcal{TR}(CN_a^p) = \mathcal{TR}(CN_c^p) = \{R_2, R_3\}$. After intersecting $\mathcal{TR}(CN_a^f)$ and $\mathcal{TR}(CN_c^f)$, we get the first qualified trajectory $R1$. Nevertheless, the trajectories in $CT^p = \{R_2, R_3\}$ obtained from $\mathcal{TR}(CN_a^p)$ and $\mathcal{TR}(CN_c^p)$ still have to be verified. To do that, we need to load *node*#4 and *node*#6 from the disk, where we find that R_2 has points p_{22} and p_{23} that contain a and c in the query range respectively, while R_3 doesn't. This implies that R_2 is another qualified trajectory while R_3 is not. Finally we get the answer set $\mathcal{A} = \{R_1, R_2\}$.

4.2 Extension for Query with Order-Sensitive Keywords

As mentioned before, one advantage of inverted octree is that it can also support the query with order-sensitive keywords ($SKRTO$). The definition of $SKRTO$ has similar constraints with $SKRT$, except that query keywords from $Q.\Phi$ should be satisfied by a trajectory in chronological order.

In the sequel, we denote the earliest timestamp of a node v as $v.t_{start}$ and the latest timestamp as $v.t_{end}$. Due to space limits, we just highlight two important

Procedure Prune(Q, L)

while $L \neq \emptyset$ **do**
  STRangeFilter($Q.R$, $Q.T$, L);
  foreach $k_i \in Q.\delta$ **do**
    SIG_i = bitwise-OR of signatures of nodes $v \in L$ from OC_i;
  foreach *node* $v \in L$ *from* OC_i **do**
    foreach SIG_j *where* $j \neq i$ **do**
      SignatureCheck(v, SIG_j);
  foreach *node* $v \in L$ *that survive from the signature test* **do**
    Suppose v comes from OC_j;
    if v *is a fully covered leaf node* **then**
      add v into CN_j^f;
    else if v *is a partially covered leaf node* **then**
      add v into CN_j^p;
    else if v *is non-leaf node* **then**
      foreach *child node* v' *of* v **do**
        if v' *is not a white node* **then**
          put v' into L;
    delete v from L;
foreach $k_i \in Q.\delta$ **do**
  determine $\mathcal{TR}(CN_i^f)$ and $\mathcal{TR}(CN_i^p)$;

Procedure Verification(Q, $\mathcal{A}$)

$\mathcal{A} \leftarrow CT^f$;
foreach *trajectory* $Tr \in CT^p$ **do**
  find out a keyword set $\Psi = \{k_i | Tr \in \mathcal{TR}(CN_i^f)\}$;
  $\Omega \leftarrow Q.\Phi - \Psi$;
  foreach $\mathcal{TR}(CN_j^p)$ ***where*** $k_j \in \Omega$ **do**
    if $Tr \in \mathcal{TR}(CN)_j^p$ ***and LoadAndJudge****(Tr, $\mathcal{TR}(CN_j^p)$)* **then**
      $\mathcal{A} \leftarrow \mathcal{A} \cup Tr$;

techniques to modify Algorithm 1 to answer $SKRTO$. One is a pruning technique during the process of procedure *Prune*. In each level, we will visit nodes of related octrees in order of query keywords. If there is a node v from OC_i whose latest timestamp is not larger than all the earliest timestamp of qualified nodes in OC_j $(j < i)$, then this node can be pruned safely. The other aspect is how to deal with trajectories in CT^f in procedure *Verification* because they may not satisfy order-sensitive keyword constraint. Instead, we should find a node sequence $v_1, v_2, ..., v_m$ such that $v_i \in CN_i^f$ $(i \leftarrow 1\ to\ m)$ and $v_{j-1}.t_{end} \leq v_j.t_{start}$ $(j \leftarrow 2\ to\ m)$. After obtaining such a cell sequence, we can guarantee trajectories

Table 3. Dataset Statistics

	LA	**NY**	**TW**
#trajectory	31,553	49,022	214,834
#location	215,614	206,416	1,287,315
#tag	3,175,597	3,068,401	28,645,905
#distinct-tag	100,843	89,665	1,852,141

Table 4. Experimental Settings

	$\|\boldsymbol{Q.\phi}\|$	$\|\boldsymbol{Q.T}\|$ (month)	$\boldsymbol{\delta(Q.R)}$(km)
LA,NYC	2, 3, 4, 5	1, 2, 3, 4, 5	10, 20, 30, 40, 50
TW	2, 3, 4 ,5	0.5, 1, 1.5, 2, 2.5	5, 10, 15, 20, 25

from intersection of trajectory set contained by v_i $(i \leftarrow 1\ to\ m)$ satisfy $SKRTO$ query. In this way, a considerable number of node access can be avoided.

5 Experiments

We conduct comprehensive empirical experiments in this section to evaluate CPU and I/O performance between our proposed algorithm and two baselines for both $SKRT$ and $SKRTO$ query.

5.1 Experimental Setup & Datasets

All the algorithms including baselines are implemented in C++ and the experiments are performed on a machine with Intel i5 CPU (3.10GHz) and 8GB main memory, running Windows 7. The raw datasets are all stored in binary files with page size 4096 bytes. Notice that the number of I/Os is considered as the number of accessed pages in different algorithms.

Three real trajectory datasets are used, two of which are from check-in records of Foursquare within areas of Los Angeles (LA) and New York (NY) [1], the third one is from geo-tagged tweets (TW) [10] collected from May 2012 to August 2013. For all three datasets, records belonging to the same user are ordered chronologically to form a trajectory of this user. The frequent and meaningful words are collected from the plain text in each record. Table 3 summaries important statistics of three datasets.

For different datasets, different set-ups are designed as shown in Table 4. The default settings of the parameter are underlined. For each experimental set, we generate 50 queries and report the average running time and accessed pages. We randomly pick up several trajectories from datasets and generate queries by selectively choosing keywords and setting reasonable query range and timespan. For the inverted octree in all the experiments, the maximal depth h is set to 5 and the split threshold ψ is set to 80.

5.2 Baselines

Two baselines extended from techniques from [6] and [23] respectively are proposed for comparison and validation of our proposal algorithm. Both baselines need extra order examination when dealing with $SKRTO$ query.

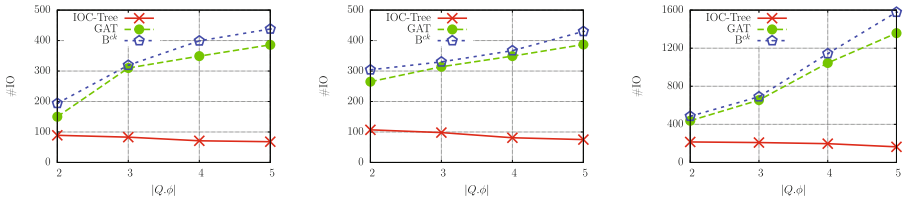
B^{ck}-tree The original B^{ck}-tree [6] is designed to solve spatial keyword problem on trajectories. It divides the spatial region into quad cells, and builds a B+ tree based on the cell division. Three elements including *wordID*, *cellID*, *posting list*, are contained by leaf entries of the B+ tree, among which the first two are the keys and *posting list* is a sequence of trajectories that go through cell *cellID* and contain word *wordID*. To incorporate temporal information, some modification are made on the posting lists, *i.e.*, trajectories are sorted by the timestamp of its point which locates in the corresponding cell and contains the corresponding word. In general, **B^{ck}-tree** firstly prunes trajectories by spatio-textual constraint, and then by temporal constraint.

GAT The second baseline is a natural extension of Grid Index for Activity Trajectories (GAT)[23]. We only divide a grid when necessary in a way similar to how we divide spatio-temporal space. For each grid, we construct a B+ tree to index trajectory points based on their timestamps and then build an inverted index of points for every keyword in this grid to record textual information. Generally, **GAT** adopts a strategy by pruning trajectory on spatio, temporal and textual constraint in sequence.

5.3 Performance Evaluation

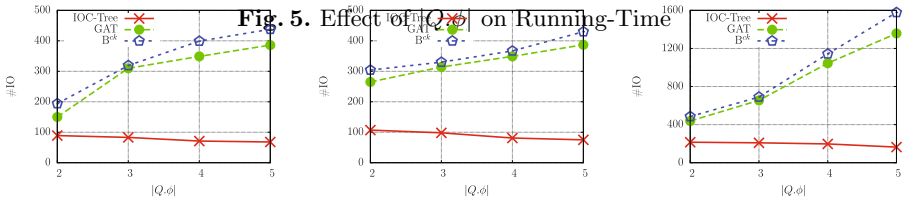
Varying Number of Query Keywords $|Q.\phi|$. In the first set of experiments, we vary $|Q.\phi|$, the number of query keywords, to compare CPU and IO cost on $SKRT$ query among three algorithms. Fig 5 shows that as $|Q.\phi|$ gets larger, the running time of IOC-Tree doesn't grow as fast as other two baselines on all three datasets. Especially in the case of dataset TW, IOC-Tree spends much less time than GAT and B^{ck}-tree when $|Q.\phi|$ becomes larger. The reason can be revealed when we carefully check Fig 6, which shows IO cost result. While the number of accessed pages by GAT and B^{ck} become larger as $|Q.\phi|$ increases, that of IOC-Tree decreases conversely. In the case of TW, IOC-Tree outperforms two baselines for nearly one order of magnitude when $|Q.\phi|$ is 5. This is due to that given a fixed spatio-temporal search space, more nodes of the octree can be pruned when more query keywords are involved during process of octrees' exploration and *signature* test. Therefore, the whole running time can be greatly reduced although more keywords may incur more examination on trajectories.

Varying Query Timespan $|Q.T|$. Then we proceed to investigate the effect of $|Q.T|$, the query timespan, on the performance of algorithms by plotting the time cost of $SKRT$ and $SKRTO$ on different datasets in Fig 7 and Fig 8. Apparently, longer query timespan means larger spatio-temporal search space, which results in longer running time. Generally, $SKRTO$ query requires more runtime cost because of the high computation of verification on order-sensitive keywords.



(a) $SKRT$ on LA (b) $SKRT$ on NYC (c) $SKRT$ on TW

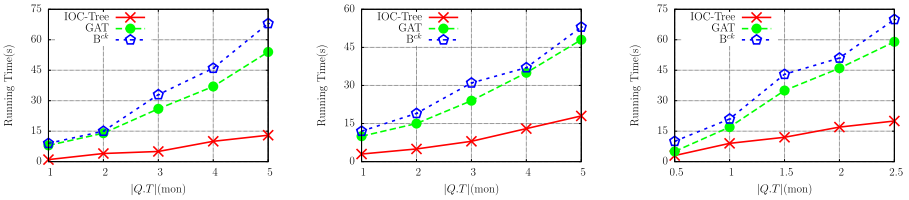
Fig. 5. Effect of $|Q.\phi|$ on Running-Time



(a) $SKRT$ on LA (b) $SKRT$ on NYC (c) $SKRT$ on TW

Fig. 6. Effect of $|Q.\phi|$ on $\#IO$

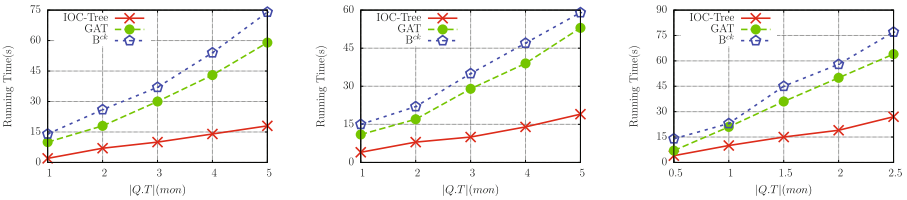
However, IOC-Tree maintains a relatively slow growth of time cost and outperforms two baselines under all cases which is benefited by fast nodes' pruning and morton code's utilization. In the case of dataset TW, the gap between IOC-Tree and two baselines becomes even larger. The reason is that trajectories of TW contain more tags averagely, and IOC-Tree adopts a keyword-first-pruning strategy; as the query timespan gets larger, GAT and B^{ck}-tree have much more candidates to verify while IOC-Tree has pruned a lot in the first place.



(a) $SKRT$ on LA (b) $SKRT$ on NYC (c) $SKRT$ on TW

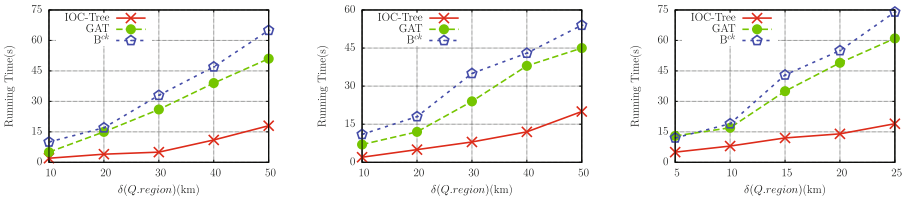
Fig. 7. Effect of $|Q.T|$

Varying Diameter of Query Range $\delta(Q.R)$. Finally we study the effect of $\delta(Q.R)$, the diameter of query spatial range, on the running time of $SKRT$ and $SKRTO$ among three algorithms. The experimental results are demonstrated in Fig 9 and Fig 10. Similar to varying $|Q.T|$, larger diameter of query spatial range naturally involves larger search space. Therefore, more cells and trajectories will be identified as candidates. As expected, the performance of all algorithms



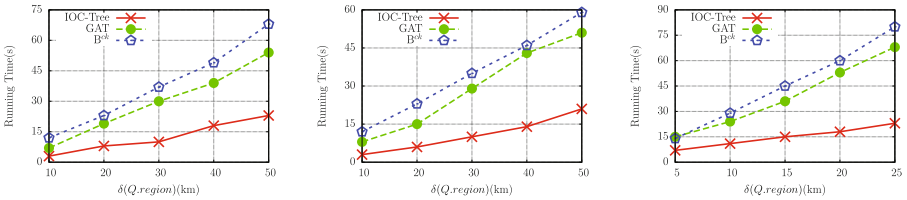
(a) $SKRTO$ on LA (b) $SKRTO$ on NYC (c) $SKRTO$ on TW

Fig. 8. Effect of $|Q.T|$



(a) $SKRT$ on LA (b) $SKRT$ on NYC (c) $SKRT$ on TW

Fig. 9. Effect of $\delta(Q.region)$



(a) $SKRTO$ on LA (b) $SKRTO$ on NYC (c) $SKRTO$ on TW

Fig. 10. Effect of $\delta(Q.region)$

degrades regarding the increase of $\delta(Q.R)$. Between two baselines, GAT performs better than B^{ck}-tree since given a fixed spatial space, GAT can quickly locate the quad cells and explore the corresponding B+ tree. However, IOC-Tree still has superior performance among all the algorithms. This is mainly because it takes fully advantage of spatio-temporal proximity and thus a considerable IO cost can be saved.

6 Conclusion

We study spatial keyword range search on trajectories, which takes spatio, tempral and textual properties of trajectories into consideration. To efficiently solve spatial keyword range search on trajectories ($SKRT$) and its variation with order-sensitive keywords ($SKRTO$), we design a novel index structure named

IOC-Tree with signature to organize trajectory data and propose an efficient algorithm for query processing. Extensive experiments on real datasets confirm the efficiency of our techniques.

Acknowledgments. The work is supported by NSFC61232006, NSFC61321064, ARC DE140100679, ARC DP130103245, ARC DP150103071 and DP150102728.

References

1. Bao, J., Zheng, Y., Mokbel, M.F.: Location-based and preference-aware recommendation using sparse geo-social networking data. In: Proceedings of the 20th International Conference on Advances in Geographic Information Systems, pp. 199–208. ACM (2012)
2. BOGORNY, V., ALVARES, L.O., Kuijpers, B., Fernandes de Macedo, J. A., MOELANS, B., and Tietbohl Palma, A.: Towards semantic trajectory knowledge discovery
3. Botea, V., Mallett, D., Nascimento, M.A., Sander, J.: Pist: an efficient and practical indexing technique for historical spatio-temporal point data. GeoInformatica **12**(2), 143–168 (2008)
4. Chakka, V.P., Everspaugh, A.C., Patel, J.M.: Indexing large trajectory data sets with seti. Ann Arbor 1001, 48109–2122 (2003)
5. Christoforaki, M., He, J., Dimopoulos, C., Markowetz, A., Suel, T.: Text vs. space: efficient geo-search query processing. In: Proceedings of the 20th ACM international conference on Information and knowledge Management, pp. 423–432. ACM (2011)
6. Cong, G., Lu, H., Ooi, B. C., Zhang, D., Zhang, M.: Efficient spatial keyword search in trajectory databases (2012). arXiv preprint arXiv:1205.2880
7. De Felipe, I., Hristidis, V., Rishe, N.: Keyword search on spatial databases. In: 2008 IEEE 24th International Conference on Data Engineering. ICDE 2008, pp. 656–665. IEEE (2008)
8. Finkel, R.A., Bentley, J.L.: Quad trees a data structure for retrieval on composite keys. Acta informatica **4**(1), 1–9 (1974)
9. Li, G., Feng, J., and Xu, J. Desks: Direction-aware spatial keyword search. In: 2012 IEEE 28th International Conference on Data Engineering (ICDE), pp. 474–485. IEEE (2012)
10. Li, G., Wang, Y., Wang, T., Feng, J.: Location-aware publish/subscribe. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 802–810. ACM (2013)
11. Long, C., Wong, R.C.-W., Wang, K., and Fu, A.W.-C.: Collective spatial keyword queries: a distance owner-driven approach. In: Proceedings of the 2013 International Conference on Management of Data, pp. 689–700. ACM (2013)
12. Luo, W., Tan, H., Chen, L., and Ni, L. M. Finding time period-based most frequent path in big trajectory data. In: Proceedings of the 2013 International Conference on Management of Data, pp. 713–724. ACM (2013)
13. Magdy, A., Mokbel, M. F., Elnikety, S., Nath, S., He, Y.: Mercury: a memory-constrained spatio-temporal real-time search on microblogs. In: 2014 IEEE 30th International Conference on Data Engineering (ICDE), pp. 172–183. IEEE (2014)

14. Meagher, D.J.: Octree encoding: a new technique for the representation, manipulation and display of arbitrary 3-d objects by computer. Electrical and Systems Engineering Department Rensseiaer Polytechnic Institute Image Processing Laboratory (1980)
15. Morton, G.M.: A computer oriented geodetic data base and a new technique in file sequencing. International Business Machines Company (1966)
16. Pfoser, D., Jensen, C.S., Theodoridis, Y., et al.: Novel approaches to the indexing of moving object trajectories. In: Proceedings of VLDB, pp. 395–406. Citeseer (2000)
17. Skovsgaard, A., Sidlauskas, D., Jensen, C.S.: Scalable top-k spatio-temporal term querying. In: 2014 IEEE 30th International Conference on Data Engineering (ICDE), pp. 148–159. IEEE (2014)
18. Song, R., Sun, W., Zheng, B., Zheng, Y.: Press: a novel framework of trajectory compression in road networks (2014). arXiv preprint arXiv:1402.1546
19. Su, H., Zheng, K., Huang, J., Jeung, H., Chen, L., Zhou, X.: Crowdplanner: a crowd-based route recommendation system. In: 2014 IEEE 30th International Conference on Data Engineering (ICDE), pp. 1144–1155. IEEE (2014)
20. Tao, Y., Papadias, D.: The mv3r-tree: A spatio-temporal access method for timestamp and interval queries
21. Wang, Y., Zheng, Y., Xue, Y.: Travel time estimation of a path using sparse trajectories. In: Proceeding of the 20th SIGKDD Conference on Knowledge Discovery and Data Mining (2014)
22. Zhang, C., Han, J., Shou, L., Lu, J., La Porta, T.: Splitter: Mining fine-grained sequential patterns in semantic trajectories. Proceedings of the VLDB Endowment **7**, 9 (2014)
23. Zheng, K., Shang, S., Yuan, N.J., Yang, Y.: Towards efficient search for activity trajectories. In: 2013 IEEE 29th International Conference on Data Engineering (ICDE), pp. 230–241. IEEE (2013)
24. Zhou, Y., Xie, X., Wang, C., Gong, Y., Ma, W.-Y.: Hybrid index structures for location-based web search. In: Proceedings of the 14th ACM International Conference on Information and Knowledge Management, pp. 155–162. ACM (2005)