Inverted Linear Quadtree: Efficient Top K Spatial Keyword Search

Chengyuan Zhang[†], Ying Zhang[†], Wenjie Zhang[†], Xuemin Lin[†] [‡]

[†] *The University Of New South Wales, Australia*

{zhangc, yingz, zhangw, lxue}@cse.unsw.edu.au

[‡] *East China Normal University, China*

***Abstract*—With advances in geo-positioning technologies and geo-location services, there are a rapidly growing amount of *spatio-textual* objects collected in many applications such as location based services and social networks, in which an object is described by its spatial location and a set of keywords (terms). Consequently, the study of spatial keyword search which explores both location and textual description of the objects has attracted great attention from the commercial organizations and research communities. In the paper, we study the problem of top k spatial keyword search (TOPK-SK), which is fundamental in the spatial keyword queries. Given a set of *spatio-textual* objects, a query location and a set of query keywords, the top k spatial keyword search retrieves the closest k objects each of which contains all keywords in the query. Based on the inverted index and the linear quadtree, we propose a novel index structure, called inverted linear quadtree (IL-Quadtree), which is carefully designed to exploit both spatial and keyword based pruning techniques to effectively reduce the search space. An efficient algorithm is then developed to tackle top k spatial keyword search. In addition, we show that the IL-Quadtree technique can also be applied to improve the performance of other spatial keyword queries such as the direction-aware top k spatial keyword search and the *spatio-textual* ranking query. Comprehensive experiments on real and synthetic data clearly demonstrate the efficiency of our methods.**

I. Introduction

With the increasing pervasiveness of the geo-positioning technologies and geo-location services, there are an enormous amount of *spatio-textual* objects available in many applications. For instance, in the local search service, online business directory (e.g., yellow pages) provides the location information as well as short descriptions of the businesses (e.g., hotels, restaurants). In the GPS navigation system, a POI (point of interest) is a geographically anchored pushpin that someone may find useful or interesting, which is usually annotated with texture information (e.g., descriptions and users' reviews). Moreover, in many social network services (e.g., Facebook, Flickr), a huge number of geo-tagged photographs are accumulated everyday, which can be geo-tagged by users, GPS-enabled smartphones or cameras with a built-in GPS receiver (e.g., Panasonic Lumix DMC-TZ10). These uploaded photographs are usually associated with multiple text labels. As a result, in recent years various spatial keyword query models and techniques have emerged such that users can effectively exploit both spatial and textual information of these *spatio-textual* objects.

In the paper, we investigate the problem of conducting top k spatial keyword search (TOPK-SK) [1], [2]; that is, given a set of *spatio-textual* objects, a query location q and a set of keywords, we aim to retrieve the k closest objects each of which contains all keywords in the query. The top k spatial keyword search is fundamental in spatial keyword queries and has a wide spectrum of applications. Below are two motivating examples.

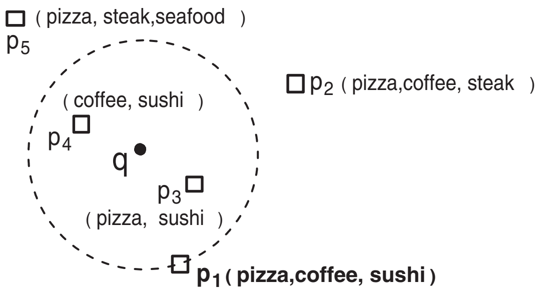

p5 (pizza, steak,seafood)
p2 (pizza,coffee, steak)
(coffee, sushi)
p4
q
p3
(pizza, sushi)
p1 (pizza,coffee, sushi)

Fig. 1. Online Yellow Pages Example

In Fig. 1, suppose there are a set of businesses whose locations (represented by squares) and service lists (a set of keywords) are registered in the online yellow pages of a local search service provider. When a GPS-enabled smartphone user wants to find a nearby restaurant to have a piece of *pizza* and a cup of *coffee*, she may send the local search server two keywords, *coffee* and *pizza*. Based on the user's current location (e.g., the point q in Fig. 1) derived from the smartphone and the two query keywords, business p_1 is returned by the server. Note that although businesses p_3 and p_4 are closer to q than p_1, they do not satisfy the keyword constraint.

In another example, suppose a traveler wants to take some photos of nearby mountains at the sunset around a tourist attraction. Based on her location and the scene she wants to capture (*mountain* and *sunset*), a TOPK-SK query on the geo-tagged photographs uploaded by other visitors can provide some useful suggestions.

Motivation. As stressed in [1], [2], the performance of the spatial keyword query is poor if objects are separately organized by textual index and spatial index. Therefore, the main challenge is how to effectively combine the spatial index and textual index such that a large number of non-promising objects can be effectively eliminated from the candidate set based on both the spatial proximity and keyword constraint.

To facilitate the TOPK-SK query, two index structures are proposed in the literature: inverted R-tree [1] and information retrieve R-tree (IR^2-tree) [2]. For each keyword t, the inverted R-tree technique builds an R-tree for objects in which t

appears, and hence objects which do not contain any query keyword can be immediately eliminated in the TOPK-SK query. The inverted R-tree based technique is efficient when the number of keywords is small. However, the performance of the inverted R-tree based technique significantly degrades when the number of keywords in the query grows because it does not exploit the **AND** semantics in the query, i.e., prune an object if it does not contain **all** query keywords. The IR^2-tree technique [2] alleviates this issue by utilizing the *signature* technique; that is, we can decide if we should further explore a node based on a small summary of the keywords appearing in the objects contained by the node. Nevertheless, according to our analysis in Section III-A, the fact that all *spatio-textual* objects are organized in a single R-tree may seriously impair the performance of the search algorithms.

Our empirical study shows that other existing spatial keyword indexing techniques (e.g., IR-tree [3], [4], KR*-tree [5] and WIR-tree [6]), which are proposed for slightly different spatial keyword search problems, cannot efficiently support the problem of top k spatial keyword search either. That is because, same as IR^2-tree, all objects are organized in a single R-tree in these techniques.

In the paper, we propose the inverted linear quadtree (IL-Quadtree) indexing technique which naturally combines the advantages of inverted R-tree and IR^2-tree techniques. Specifically, for each keyword we build a linear quadtree for the related objects so that the objects which do not contain any query keyword can be immediately excluded from computation. Besides facilitating spatial search, the quadtree for each keyword can also serve as the *signature* of the objects in the sense that we can effectively prune a group of objects based on the **AND** semantics. Moreover, as the space filling curve technique is employed by the linear quadtree, the objects are clustered on the disk based on their spatial proximity, which enhances the I/O efficiency of our search algorithm.

Contributions. Our contributions can be summarized as follows.

- To facilitate the top k spatial keyword search (TOPK-SK), we propose a novel indexing structure called IL-Quadtree to effectively organize *spatio-textual* objects.
- Based on IL-Quadtree, we develop an efficient top k spatial keyword search algorithm.
- We show that IL-Quadtree can also efficiently support the direction-aware top k spatial keyword search and spatial keyword ranking query.
- Comprehensive experiments demonstrate the efficiency of our methods.

Roadmap. The rest of the paper is organized as follows. Section II formally defines the problem of top k spatial keyword search, followed by the introduction of the related work. Section III presents the IL-Quadtree structure. Efficient top k keyword search algorithm is proposed in Section IV and possible extensions of the IL-Quadtree index are discussed in Section V. Experimental results are reported in Section VI. Section VII concludes the paper.

II. Preliminary

In this section, we first formally define the problem of top k spatial keyword search, and then introduce the related work. Table I below summarizes the mathematical notations used throughout this paper.

Notation	Definition
o (q)	a *spatio-textual* object (query)
$o.loc(q.loc)$	location of the object o (query q)
$o.\mathcal{T}$	a set of keywords used to describe o
$q.\mathcal{T}$	a set of query keywords for query q
$\mathcal{V}$, v	vocabulary, size of the vocabulary
t	a keyword (term) in $\mathcal{V}$
l	the number of query keywords in $q.\mathcal{T}$
n	the number of *spatio-textual* objects
m	the average number of keywords in each object
n_o	the number of objects within the search region
p_s	the average *surviving probability* of the objects within the search region
$\delta(p_1, p_2)$	the distance between points p_1 and p_2
$\delta_{min}(R, p_2)$	the minimal distance between the region R and point p_2
e	a node of an R-tree or quadtree
$seq(e)$	the split sequence of a quadtree node
w	maximal depth of IL-Quadtree
w'	minimal depth of the *black* leaf node in IL-Quadtree
c	split threshold for a node in IL-Quadtree

TABLE I
THE SUMMARY OF NOTATIONS.

A. Problem Definition

In the paper, a *spatio-textual* object o is described by a spatial point in a two dimensional space and a set of keywords (terms) from the vocabulary $\mathcal{V}$, denoted by $o.loc$ and $o.\mathcal{T}$ respectively. A top k spatial keyword query, denoted by q, consists of a natural number k, a query location and a set of query keywords. The problem of top k spatial keyword search (TOPK-SK) is formally defined as follows.

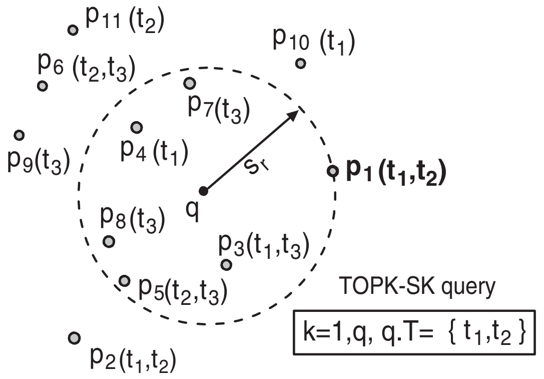


Fig. 2. Example of TOPK-SK query

Problem Statement. Given a set $\mathcal{O}$ of *spatio-textual* objects, a query object q where $q.loc$ is the query location and $q.\mathcal{T}$ is a set of query keywords, we aim to find the closest k objects each of which contains **all** of the query keywords. We assume ties are broken arbitrarily in the paper.

In the paper hereafter, whenever there is no ambiguity, "*spatio-textual* object" is abbreviated to "object" and o (q) is used to represent its location $o.loc$ ($q.loc$).

Example 1: In Fig. 2, there are a set of *spatio-textual* objects $\{p_1, \ldots, p_{11}\}$. A user located at the point q wants to find the nearest object (i.e.,$k = 1$) which contains keywords t_1 and t_2 (i.e., $q.\mathcal{T} = \{t_1, t_2\}$). Fig. 2 shows that the object retrieved by the TOPK-SK query is p_1. Objects p_3, p_4, p_7, p_8 and p_5 are eliminated due to the keyword constraint.

B. Related Work

In this subsection, we first present two existing techniques, inverted R-tree and IR^2-tree, for the problem of TOPK-SK query as well as some other top k queries dealing with both spatial and texture information. Then other types of spatial keyword queries are introduced. We also present details of some related techniques which can be adopted to the TOPK-SK query.

Inverted R-tree

The inverted index technique has been extensively applied in various information retrieval applications. For each keyword (term) t, a set of objects (e.g., files and documents) containing t are kept in a list. Given the fact that the number of keywords input by the user is usually small in practice, the inverted index technique can efficiently support keyword search since only a few lists need to be loaded and the target objects can be identified by merging the lists.

To efficiently facilitate the spatial keyword search, it is fairly natural to employ the spatial index techniques to organize the objects for each keyword, instead of keeping them in a list. Then, for a given TOPK-SK query, we can simultaneously apply the incremental nearest neighbor search [7] on the related spatial indices until k objects satisfying the keyword constraint are retrieved. The inverted grid index is proposed in [8], [9], [10] to organize objects for each keyword. Considering that the grid index does not scale well towards the location skewness of the objects, [1] proposes the inverted R-tree technique. For each keyword $t \in \mathcal{V}$, an R-tree is constructed for the objects in which t appears.

Following is an example of the inverted R-tree.

Example 2: Given a set of objects as shown in Fig. 2 where $\mathcal{V} = \{t_1, t_2, t_3\}$, Fig. 3 shows that three R-trees R_1, R_2 and R_3 are constructed based on the related objects for t_1, t_2 and t_3 respectively. For the TOPK-SK query in Fig. 2, the objects *accessed* in inverted R-tree based technique are p_3, p_4, p_5 and p_1 in order. Here, an object is *accessed* if it is within the search region and is loaded in the computation. Note that although the objects p_7 and p_8 are also within the search region, none of them is *accessed* as they are not contained in R_1 or R_2.

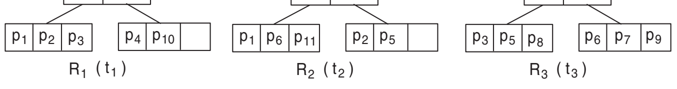


Fig. 3. Example of inverted R-tree

IR^2-tree

Intuitively, the inverted R-tree is efficient when there is only one query keyword since we only need to issue a k nearest neighbor search on the corresponding R-tree. Nevertheless, the performance of the inverted R-tree significantly degrades against the number of keywords l in the query. This is because the search region of the TOPK-SK query will enlarge against the number of keywords, i.e., an object is less likely to contain all query keywords in $\mathcal{T}$ when l increases, and all objects which contain at least one keyword and are within the search region will be visited.

Motivated by this, Felipe *et al.* propose the information retrieval R-tree (IR^2-tree) structure [2] which can alleviate above issue by utilizing the *signature* technique. Suppose objects are organized by an R-tree according to their spatial locations, a *signature* (i.e., a fixed-length bitmap) is put to each node of R-tree, which summarizes the set of keywords contained by its descendent data entries (objects). More specifically, suppose each keyword is mapped to a particular position of the signature, e.g., by applying hash function on the keyword, the i-th bit of the signature of a node is set to one if any of the keywords contained by the objects in the node is mapped to i. Clearly, the *signature* of an intermediate IR^2-tree node is the bitwise-OR of its child entries' *signatures*. During the query processing, we do not need to further explore an IR^2-tree node if its *signature* confirms that none of the objects in the node contains all query keywords, and hence the I/O cost can be saved.

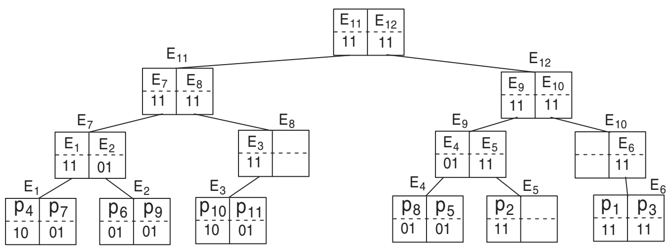


Fig. 4. Example of IR^2-tree

Example 3: Regarding the objects in Fig. 2, assume that each *signature* (bitmap) has two bits, and keywords t_1, t_2 and t_3 are mapped to the first, the second and the second positions of the bitmap respectively, Fig. 4 illustrates the IR^2-tree entries and their corresponding *signatures*. For the TOPK-SK query in Fig. 2, although there are totally 6 objects within the search region, we only access 4 objects p_3, p_4, p_7 and p_1 in order because the entry E_4 can be eliminated based on its *signature*, and hence the accesses of p_5 and p_8 can be avoided. Note that, although none of the objects in E_1 (p_4 and p_7) satisfies the keywords constraint, E_1 is loaded due to the *false positive* incurred.

Variants of the top k spatial keyword search.

Besides the top k spatial keyword search, there are many important variants in the literature with different focus. Instead of applying the keyword constraint, the spatial keyword ranking query [3], [4], [11] is proposed to rank objects based on a scoring function which considers the distance to the query location as well as the textual relevance to the query keywords. In [3], [4], the information R-tree (IR-Tree) structure is proposed to support the spatial keyword ranking query, which is similar to the IR^2-tree technique. Based on the inverted R-tree, the spatial inverted Index (S2I) structure is proposed in [11], where an aggregate R-tree is employed to organize objects for each keyword.

In [12], Li *et al.* study the problem of top k spatial keyword search in which the direction constraint is considered. Effective direction-aware index structure is proposed to prune the search space. To facilitate the joint processing of multiple TOPK-SK queries, the WIR-tree is proposed in [6] following the framework of IR-tree. Other variants include location-aware type ahead search [13], top k spatial keyword search on road network [14], etc.

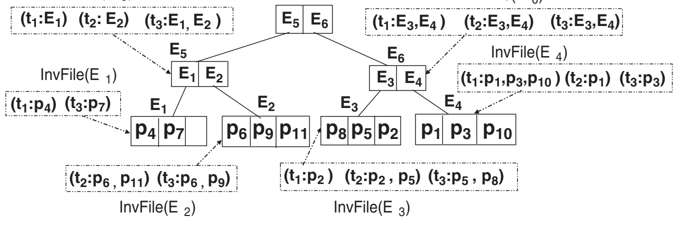
Other Spatial Keyword related Queries.

Besides the top k query, other types of spatial keyword queries have also attracted much attention in the literature. Given a search region, the spatial keyword search is studied in [8], [9], [5], [10] which aims to efficiently identify the objects which satisfy both spatial and keyword constraints; that is, each retrieved object falls in the search region and contains all query keywords. Specifically, the inverted grid index technique is employed in [8], [9], [10] where a grid index is used to organize the related objects for each keyword. The KR*-tree is presented in [5]. The inverted R-tree [1] can also support the region based spatial keyword search. In [15] and [16], the problem of the spatial keyword ranking query within a search region is studied, where various *tf-idf* based ranking functions are proposed. In [17], Li *et al.* investigate the problem of spatial approximate string search. To explore the spatial relationship among the candidate objects, Zhang *et al.* [18] study the problem of m-closest keyword search which retrieves spatially closest objects matching m keywords. Chao *et al.* [19] investigate the problem of k collective keyword search. Very recently, assuming the spatial extent of the objects are regions, Fan *el al.* study the problem of *spatio-textual* similarity search in [20].

Details of the IR-tree, KR*-tree and WIR-tree

Although the above-mentioned IR-tree, KR*-tree and WIR-tree techniques are designed to tackle slightly different problems, they can be easily adopted to support top k spatial keyword search. Below, we provide some detailed introduction for these techniques, which will be evaluated in our empirical study.

IR-tree. In [3], [4], the information R-tree (IR-tree) structure is proposed to support the spatial keyword ranking query, which is similar to the IR^2-tree technique. The main difference is that, instead of the *signature*, an inverted file is maintained for each node which maintains the keywords and the related information for objects contained by the node. Some optimizations are applied to further improve the query performance by taking advantage of the tree construction methods.

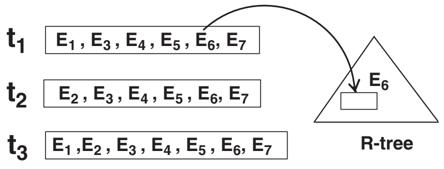


InvFile(E 5)
(t1:E1) (t2: E2) (t3:E1, E2)
E7
E5 E6
InvFile(E 6)
(t1:E3,E4) (t2:E3,E4) (t3:E3,E4)
E5
E1 E2
E6
E3 E4
InvFile(E 4)
(t1:p1,p3,p10) (t2:p1) (t3:p3)
InvFile(E 1)
(t1:p4) (t3:p7)
E1
E2
E3
E4
p4 p7
p6 p9 p11
p8 p5 p2
p1 p3 p10
(t2:p6 , p11) (t3:p6 , p9)
InvFile(E 2)
(t1:p2) (t2:p2 , p5) (t3:p5 , p8)
InvFile(E 3)

Fig. 5. Example of IR-tree

Example 4: Suppose the objects in Fig. 2 are organized by the IR-tree as shown in Fig. 5 [1]. Same as the Example 3, there are 6 objects in the search region. Nevertheless, we only access 4 objects p_3, p_8, p_5 and p_1 in order because the entry E_1 can be eliminated based on inverted file $InvFile(E_1)$.

WIR-tree. WIR-tree is proposed in [6] to efficiently support a batch of TOPK-SK queries. The structure of WIR-tree tree is the same as the IR-tree where all objects are organized by one augmented R-tree. The main difference is that the construction of WIR-tree takes advantage of the term frequencies of the keywords to facilitate the joint TOPK-SK queries. More specifically, the objects will be recursively partitioned into 2^h groups for given h most frequent terms $t_1, \ldots, t_h$. In the i-th iteration, objects in a group g will be divided into two groups g_1 and g_2 where objects in g_1 contain t_i and objects in g_2 do not. Then WIR-tree is constructed based on these groups.



t1 E1 , E3 , E4 , E5 , E6, E7
t2 E2 , E3 , E4 , E5 , E6, E7
t3 E1 ,E2 , E3 , E4 , E5 , E6, E7
E6
R-tree

Fig. 6. Example of KR*-tree

KR*-tree. In [5], Hariharan *et al.* propose the KR*-tree structure to process the region based spatial keyword search. KR*-tree is similar to IR-tree in the sense that objects are first organized by one R-tree and then inverted index techniques are applied. As shown in Fig. 6, for each keyword t, an R-tree entry e is kept in the list if t appears in any objects contained by e. Regarding the example in Fig. 2, KR*-tree has the same R-tree structure as the IR-tree (Fig. 5). In Fig. 6, the entry lists of keywords t_1, t_2 and t_3 are illustrated. Same as Example 3, p_4 and p_7 can be eliminated because entry E_1 does not appear in the entry list of t_2.

III. IL-QUADTREE

In this section, we introduce a new indexing mechanism called inverted linear quadtree (IL-Quadtree) for the top k spatial keyword search. In Section III-A we describe the shortcomings of the existing indexing approaches. To address these shortcomings, Section III-B proposes the inverted linear quadtree based index structure.

A. *Motivation*

In this subsection, we first conduct performance analysis based on some assumptions to intuitively demonstrate the advantages and disadvantages of the existing techniques. Then we present some important properties for an indexing structure supporting TOPK-SK query.

Since the incremental nearest neighbor search is applied in the existing works, they have the same search region; that is, the radius of the search region centered at q (e.g., s_r in Fig. 2) grows until finding k objects satisfying keywords constraint. Recall that we say an object is *accessed* if it is within the search region and is loaded in the main memory.

[1]We do not show the term frequency related aggregate information in IR-tree since we do not need these information for TOPK-SK query.

In the sequel, we evaluate the performance of inverted R-tree and IR^2-tree by estimating the number of objects *accessed* during the search, denoted by s_n, because the cost model derived is simple and can provide some useful insights. Let n_o and p_s denote the number of objects within the search region and the average *surviving probability* of these objects (i.e., an object within the search region is expected to be loaded with probability p_s), respectively. Clearly, we have $s_n = n_o \times p_s$. Below, we will analyze the performance of two techniques in the aspects of n_o and p_s values.

Assume n objects are uniformly distributed in two dimensional space $[0,1]^2$, and there are m keywords for each object which are randomly picked up (with replacement) from the vocabulary $\mathcal{V}$ with size v. For a given TOPK-SK query with l keywords, the probability that an object contains all query keywords is $(\frac{m}{v})^l$. Therefore, the number of objects in the search region (n_o) is expected to be $k \times (\frac{v}{m})^l$. Let s_r denote the radius of the search region, with the same rationale in [7] we have

$$s_r = \sqrt{\frac{k \times (\frac{v}{m})^l}{\pi \times n}} \quad (1)$$

Inverted R-tree. According to the uniform assumption, for each keyword $t_i \in \mathcal{V}$, there are $\frac{n \times m}{v}$ objects in its corresponding R-tree R_i. Since only the objects containing at least one keyword in $q.\mathcal{T}$ are accessed, it is immediate that

$$s_n = n_o = l \times \pi \times s_r^2 \times \frac{n \times m}{v} \quad (2)$$

$$= l \times k \times (\frac{v}{m})^{l-1} \quad (3)$$

As shown in Equation 3, we have $s_n = k$ when $l = 1$, i.e., there is only one query keyword, which is the optimal solution in terms of the number of objects accessed. However, s_n grows exponentially against the number of keywords l in the TOPK-SK query because the number of objects within the search region (n_o) grows quickly but none of them can be eliminated (i.e., $p_s = 1$). Therefore, the inverted R-tree based technique performs poor when l increases.

IR^2-tree. In [2], an object will be loaded in the main memory if all of its descendent entries satisfy the conjunction condition. Here, we only need to consider the *signatures* for the leaf nodes because the *signatures* on higher level do not enhance the filtering capability although they may speed up the *filtering* process. Given the s_r in Equation 1, the expected number of objects within the search region, denoted by n_o, equals $k \times (\frac{v}{m})^l$. Let n_p, n_e and n_L denote the page size, the entry size and the length of each *signature* (bitmap), the capability of each IR^2-tree node, denoted by n_c, is $\lceil \frac{n_p}{n_e + n_L} \rceil$. Now, we assume objects are organized by $\frac{n_o}{n_c}$ groups (i.e., leaf nodes). For each group, let p denote the *surviving probability* of a group, i.e., the probability that the *signature* satisfies the keyword constraint. In [2], a group survives the test if the l corresponding bits are hit in the *signature* (bitmap)[2]. Given a group of n_c objects and each object is described by m keywords, a bit of the *signature* is set with probability $\frac{n_c \times m}{n_L}$. Then, we have $p = (\frac{n_c \times m}{n_L})^l$. Consequently, the expected number of objects visited (s_n) is

$$s_n = \frac{n_o}{n_c} \times p \times n_c = n_o \times p \quad (4)$$

$$= k \times (\frac{v}{m})^l \times (\frac{m \times n_c}{n_L})^l \quad (5)$$

[2]For the simplicity of the cost model, we assume l keywords are mapped to different bits.

As shown in Equation 5, the probability p ($p_s = p$) decreases exponentially regarding l. Therefore, compared to the inverted R-tree method (Equation 3), Equation 5 implies that the IR^2-tree technique might be less sensitive to the number of query keywords l if the costs incurred by the enlarged search region can be compensated by the enhanced pruning capability.

Despite its advantage, IR^2-tree technique is less efficient than inverted R-tree based one when the number of keyword is small. As shown in Equation 3 and Equation 5, we have n_o equals $l \times k \times (\frac{v}{m})^{l-1}$ and $k \times (\frac{v}{m})^l$ for inverted R-tree and IR^2-tree techniques respectively. This implies that much more objects are involved in the search region in IR^2-tree based methods since $\frac{m \times l}{v} \ll 1$ in practice. Therefore, as confirmed in our initial empirical study, inverted R-tree always outperforms IR^2-tree when l is small although a significant amount of objects can be pruned. Moreover, as a single tree is constructed for all objects, IR^2-tree technique cannot properly capture the distribution of objects regarding each individual keyword. In other existing techniques such as IR-tree, WIR-tree and KR*-tree, the objects are also organized by a single augmented R-tree. This implies that they inevitably suffer from the same problems.

Above observations motivate us to develop a new index structure so that we may have a small number of objects within the search region (i.e., small n_o) as well as the ability to prune objects within the search region (i.e., small p_s). **Firstly**, it is desirable to develop a new index structure which falls in the category of inverted index, i.e., related objects are organized by a spatial index for each keyword, so that the objects which do not contain any query keyword can be immediately eliminated. **Secondly**, the new index structure should be adaptive to the distribution of the objects for each keyword. **Thirdly**, we need to exploit the **AND** semantic, i.e., pruning a group of objects which do not satisfy the keyword constraint.

Unlike IR^2-tree which applies the *signature* technique on the keyword, we can build the *signatures* based on the space partition for each keyword so that a group of objects can be eliminated based on their *signature*. Given a region R in the space, let s_i be an indicator where s_i is set *true* if there is at least an object containing keyword t_i, and s_i is set *false* otherwise. Obviously, we do not need to explore any object within R if any of the s_i is set *false* and $t_i \in q.\mathcal{T}$ since none of these objects satisfies the conjunction condition. Here, s_i is the *signature* of the region R regarding keyword t_i, which can be represented by one bit.

We can enforce the inverted R-tree technique to exploit the **AND** semantics by issuing a range query on other related R-tree. However, this cannot pay-off the high checking cost.

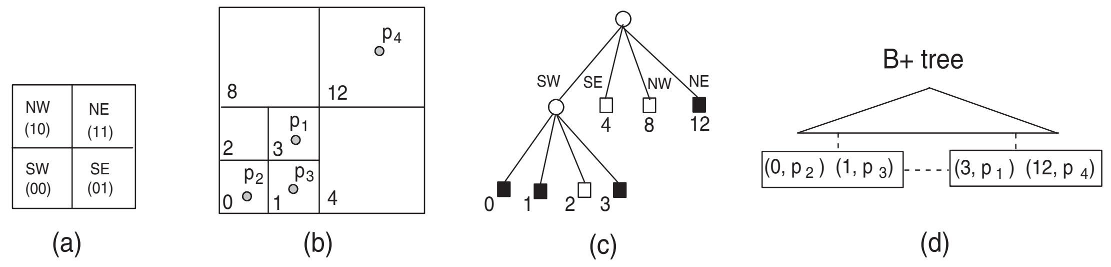


Fig. 7. Linear Quadtree Example

On the other hand, the inverted grid index technique [8], [9], [10] can be naturally extended to utilize the *signature* technique. More specifically, as the space is partitioned into cells by the grid index, for each keyword $t_i \in \mathcal{V}$ we can use one bit as the *signature* of a cell. Our empirical study shows this can significantly improve the performance. However, the grid index technique does not scale well towards the skewness of the objects, which is common in many real applications.

In the paper, we adopt the linear quadtree structure because the quadtree is more flexible in the sense that the index is adaptive to the distribution of the objects and we may prune the objects at high levels of the quadtree. Clearly, the new structure proposed satisfies the above-mentioned three important criteria of the spatial keyword indexing method.

B. *IL-Quadtree Structure*

A quadtree is a space partitioning tree data structure in which a d-dimensional space is recursively subdivided into 2^d regions. Due to its simplicity and regularity, the quadtree technique has been widely applied in many applications. As an efficient implementation of the disk-based quadtree, the linear quadtree [21] is proposed to keep the non-empty leaf node of the quadtree in an auxiliary disk-based one dimensional structure (e.g., B^+ tree), where each node can be encoded by the space filling curve techniques (e.g., Morton code [22], Hilbert code and gray code [23]).

In the paper, we encode the quadtree nodes based on the Morton code [22] (a.k.a. Z-order) because the Morton code of a node is encoded based on its *split sequence*, i.e., the path of the node in the quadtree, and the code of a particular node (region) in the space is unique. This is essential because multiple quadtrees with different shapes are used in the paper.

Now we describe how to derive the Morton code of a node based on its split sequence in 2-dimensional space. As shown in Fig. 7(a), assuming that quadtrees resulting from a split are numbered in the order SW, SE, NW and NE, which are represented by 00, 01, 10 and 11 respectively [3]. Then the code can be derived by concatenating the split codes in each subdivision. For example, Fig. 7(b) and Fig. 7(c) show the space partition and the corresponding tree structure of a simple quadtree for a given set of points $\{p_1, \ldots, p_4\}$ where leaf nodes are labelled by their Morton codes. As shown in Fig. 7(c), in the paper, we use a circle and a square to denote the non-leaf node and leaf node respectively. Moreover, a leaf node is set *black* if it is not empty, i.e., it contains at least one point. Otherwise, it is a *white* leaf node. Suppose the maximal depth of the quadtree is 2, the split sequence of the node 1 is "SW, SE" and hence its code is represented by 0001. For the node at higher level, we use 0 to pad the remaining binary digits. For instance, the split sequence of the node 12 is "NE" and its code is represented by 1100 where the last two binary digits are padding. In Fig. 7(b) and (c), the codes of the leaf nodes are labelled by the corresponding integer number of their split sequences (i.e, Morton code). Note that in the paper the quadtree structure is kept as the space partition based *signature* of the objects, and hence the level of a node in the quadtree is available during the query processing. Consequently, we can come up with the correct node (region) based on the code and the level information.

For the linear quadtree, we only keep the *black* leaf nodes on the disk by one dimensional index structure (e.g., B^+ tree), which are ordered by their Morton codes. Fig. 7(d) shows an ordering results of these *black* leaf nodes as well as the objects resident on them, where the node codes are represented as integer numbers.

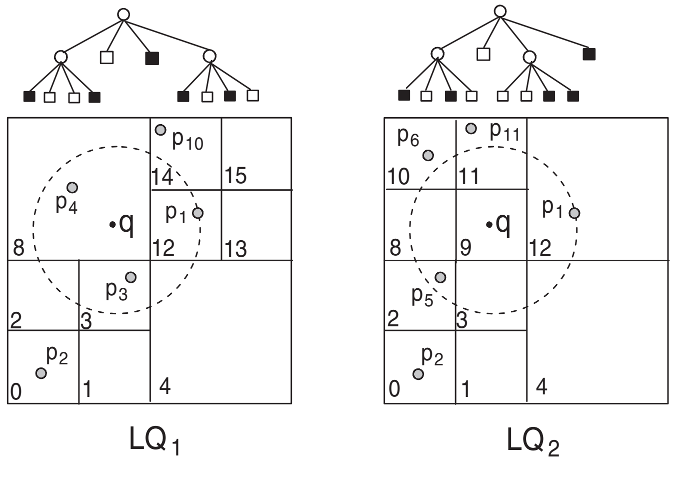


Fig. 8. Example of IL-Quadtree

IL-Quadtree. In the paper, for each keyword $t_i \in \mathcal{V}$ we build a linear quadtree, denoted by LQ_i, for the objects which contain the keyword t_i. Besides the *black* leaf nodes, we also explicitly keep the quadtree structure, which serves as the *signature* of the objects in LQ_i, which can be easily fit into the main memory. More specifically, a bit is kept for each node of the quadtree, which is set to 1 for *black* leaf nodes and non-leaf nodes and 0 otherwise. Obviously, a node

[3]Note that SW is the abbreviation of South-West. Similar definition goes to SE, NW and NE.

in LQ_i is empty (i.e., it does not contain any object with keyword t_i) if the bit is set to 0. Regarding the objects in Example 1 (Fig. 2), Fig. 8 illustrates the linear quadtrees LQ_1 and LQ_2 constructed for keywords t_1 and t_2 respectively. Recall that the leaf nodes are labeled by their Morton codes.

Index Maintenance. For an incoming new object o, it will be inserted into the corresponding linear quadtrees based on its textual information. Particularly, a leaf node of the quadtree is split if it contains more than c objects and it does not reach the maximal depth w which is the pre-determined maximal partition level. As to the deletion, an object o will be removed from its corresponding linear quadtrees. Meanwhile, some of the cells may be merged due to the deletion. For the effectiveness of the *signatures*, we enforce that all objects are pushed to the *black* leaf node below the level w' (minimal partition level) because a *black* leaf node at high level may impair the *pruning* capability.

IV. IL-Quadtree based TOPK-SK query

In this section, we introduce an efficient TOPK-SK query algorithm assuming that objects are organized by an IL-Quadtree. Same as other inverted index based approaches, we also conduct incremental nearest neighbor search on the IL-Quadtree. The main difference is that we can make use of the space partition based *signatures* (i.e., quadtree structures) to eliminate non-promising objects. Example 5 below shows the motivation of our algorithm.

Example 5: Given the TOPK-SK query as shown in Example 1 (Fig. 2) where the query point is q, $q.\mathcal{T} = \{t_1, t_2\}$ and $k = 1$, the search region of IL-Quadtree based technique is the same as the search region of the inverted R-tree and IR^2-tree techniques. Only objects p_4 and p_1 will be *accessed* in our example. Objects p_7 and p_8 are immediately eliminated because they do not appear in LQ_1 or LQ_2. Moreover, we do not need to explore cell 3 in LQ_1 because cell 3 in LQ_2 is empty, and hence p_3 is eliminated. Similarly, p_5 is not *accessed* as well. Note that we access p_4 because cell 8 in LQ_1 overlaps cell 10 in LQ_2, and both of them are *black* leaf nodes.

Algorithm 1 illustrates the details of the IL-Quadtree based TOPK-SK query. A min heap $\mathcal{H}$ is employed to keep the quadtree nodes where the key of a node is its minimal distance to q, denoted by $\delta_{min}(e, q)$. Let $I(q.\mathcal{T})$ denote a subset of $\{1, \ldots, v\}$ such that $i \in I(q.\mathcal{T})$ if the keyword t_i is a query keyword. In Line 3, the root nodes of the related quadtrees are pushed into the heap $\mathcal{H}$. Each node e popped in Line 5 is processed as follows. For a *black* leaf node e of the keyword t_i, if there are more than one keyword in $q.\mathcal{T}$ (i.e., $l > 1$) we need to decide if the objects within e should be loaded by testing the *signatures* of the quadtrees of other query keywords (Line 9). More specifically, based on the Morton code (i.e., split sequence) of the node e, we can quickly find out if e is overlapped with another *black* leaf node of LQ_j. Following the split sequence of e, we claim that none of the objects satisfies the keyword constraint if a *white* leaf node in LQ_j is encountered. If the node e passes these $l-1$ tests (Line 10), Line 11 loads the objects contained by e. We keep a counter for each object o, denoted by o_{hit} and Line 13 increases it by one. A set $\mathcal{R}$ is employed to keep the k closest objects seen so far, which satisfy the keyword constraint, and the distance threshold λ represents the k-th closest distance in $\mathcal{R}$. Lines 13-16 calculate the distance between the object o and q and update $\mathcal{R}$ and λ if the counter of o reaches l, i.e., o contains all query keywords. When the popped node e is a non-leaf node (Line 17), a child node e' of e will be pushed to $\mathcal{H}$ if it is not a *white* leaf node and the minimal distance between e' and q, denoted by $\delta_{min}(e', q)$, is not larger than λ (Line 18-20). The algorithm terminates when $\mathcal{H}$ is empty and the results are kept in $\mathcal{R}$.

Algorithm 1: **TOPK-SK** ($\mathcal{Q}$, q, k)

```
Input  : $LQ$ : the IL-Quadtree, $q$ : the spatio-textual query
         $k$ : number of objects returned,
Output : $\mathcal{R}$ : TOPK-SK query result.
1  $\mathcal{R} := \emptyset$; $\mathcal{H} = \emptyset$; $\lambda = \infty$ ;
2  for each $LQ_i$ where $i \in I(q.\mathcal{T})$ do
3      Push root node of $R$ into $H$ ;
4  while $\mathcal{H} \neq \emptyset$ do
5      $e \leftarrow$ the quadtree node popped from $H$;
6      if $e$ is a black leaf node then
7          Suppose $e$ is from $LQ_i$;
8          for each $LQ_j$ where $j \neq i$ and $j \in I(q.\mathcal{T})$ do
9              CheckSignature($e$, $LQ_j$) ;
10         if $e$ passed the signature test then
11             Load objects contained by $e$;
12             for each object $o$ contained by $e$ do
13                 $o_{hit} := o_{hit} + 1$ ;
14                 if $o_{hit} = l$ then
15                     Compute $\delta(o, q)$ ;
16                     Update $\lambda$ and $\mathcal{R}$ ;
17     else if $e$ is a non-leaf node then
18         for each child node $e'$ of $e$ do
19             if $e$ is
               not a white leaf node and $\delta_{min}(e', q) \leq \lambda$ then
20                 Push $e'$ into $H$;
21 return $\mathcal{R}$
```

Correctness. We first show the correctness of the *signature* check. For any object o in a quadtree LQ_j, o must reside on a *black* leaf node. Therefore, given a *black* leaf node e in LQ_i, it is immediate that we do not need to explore objects in e if there is another quadtree LQ_j where $j \in I(q.\mathcal{T})$ and none of the *black* leaf nodes of LQ_j overlaps e. According to the construction of the quadtree, if e overlaps a node e' in LQ_j, it will imply that $seq(e) \subseteq seq(e')$ or $seq(e') \subseteq seq(e)$ where $seq(e)$ is the split sequence of the node e. As shown in Fig. 9, there are three cases for a given split sequence $seq(e)$ and quadtree LQ_j: (i) encounter a *white* node (Fig. 9(a)), (ii) encounter a *black* node (Fig. 9(b)) and (iii) end up at a non-leaf node (Fig. 9(c)). Clearly, the occurrence of the case (i) implies that we do not need to explore e since none of the *black* leaf nodes in LQ_j overlaps the node e. Otherwise, cases (ii) and (iii) indicate that there exists a *black* leaf node in LQ_j which overlaps e. Therefore, the correctness of our *signature* follows. Secondly, we show that it is safe to prune a node e' if $\delta_{min}(e', q) > \lambda$. This is immediate because the nodes of the quadtrees are accessed in non-decreasing order according to their minimal distances to q.

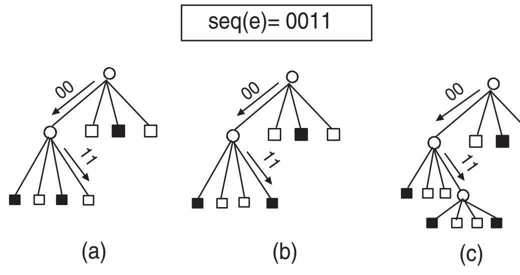


Fig. 9. *signature* Check

Time Complexity. Suppose there are n_o objects within the search region of the algorithm, the heap size is at most $l \times n_o$ and hence the heap maintenance cost is $\log(l \times n_o)$ for each node in the worst case where none of the objects is pruned (Line 20). The *signature* checking cost is $(l-1) \times w$ for each node at Line 9 and hence the total checking cost is $n_o \times (l-1) \times w$ in the worst case, where w is the maximal depth of the quadtree. As to each object o, it takes $O(1)$ time to find and increase its counter (o_{hit}) supposing that *accessed* objects are kept in a hash table. Therefore, in the worse case, the computational time of Algorithm 1 is $O(l \times n_o \times (\log(l \times n_o) + l \times w))$. In the empirical study, the performance of the algorithm is highly efficient as a large number of objects and quadtree nodes are eliminated by making use of distance based pruning and *signature* based pruning techniques.

V. Support Other Spatial Keyword Queries

IL-Quadtree can also support other types of spatial keyword queries. In the paper, we focus on the direction-aware top k spatial keyword search and the spatial keyword ranking queries.

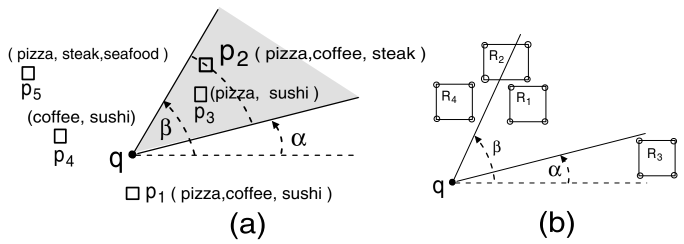


Fig. 10. Direction-aware top k spatial keyword search

A. Direction-aware top k spatial keyword search

As discussed in [12], in some applications users may be interested in the direction-aware top k spatial keyword search; that is, besides the keyword constraint, the candidate objects must fall in a particular direction regarding the query q. As shown in Fig. 10(a), suppose query keywords are `coffee` and `pizza` and query direction constraint is described by [α, β] (i.e., the shaded area), the object p_2 is the nearest object which satisfies both keyword and direction constraint. Note that p_1 is not a candidate object because of the direction constraint.

Let $angle(o, q)$ denote the angle of the object o regarding the query q, the problem of direction-aware top k spatial keyword search is formally defined as follows.

Problem Definition. Given a set $\mathcal{O}$ of objects, a query q with direction constraint [α, β], we aim to retrieve k closest objects $\{o\}$ such that each object o satisfies both keyword and direction constraints; that is, $o.\mathcal{T}$ constains all keywords in $q.\mathcal{T}$ and $\alpha \leq angle(o, q) \leq \beta$.

Algorithm 1 in Section IV can be easily extended to support the direction-aware top k search by verifying direction constraint for the objects and quadtree nodes at Line 12 and 19 of Algorithm 1 respectively. It is trivial to check the direction constraint for an object. We show how to efficiently check the direction constraint for a rectangular region (quadtree node). Particularly, we say a region R *fully satisfies* the direction constraint if all points within R meet the direction constraint. And R does not *satisfy* the direction constraint if none of the points in R meets the direction constraint. Otherwise, R *partially satisfies* the direction constraint. As shown in Fig. 10(b), we have R_1 *fully satisfies* the direction constraint, R_2 *partially satisfies* the direction constraint, and R_3 and R_4 do not satisfy the direction constraint.

Clearly, we can prune a quadtree node if it does not satisfy the direction constraint. The qudatree node e and all of its descent nodes satisfy the direction constraint if e *fully satisfies* the direction constraint. We need to further check the direction constraint for its child nodes or objects contained if the node *partially satisfies* the direction constraint.

B. Spatial Keyword based Ranking

In some applications such as location based web documents search, the textual information of an object may be described by a bag of keywords, and it is desirable to evaluate the textual relevance between an object and the query based on some ranking functions in the field of information retrieval. In this subsection, we first introduce how to rank objects based on both the spatial proximity and the textual relevance. Then an efficient ranking algorithm is presented based on a slight modification of IL-Quadtree.

Definition 1: **Spatial proximity** $f_s(q, o)$. Let δ_{max} denote the maximal distance in the space, the spatial relevance between the query q and the object o, denoted by $f_s(q, o)$, is defined as $\frac{\delta(q.loc, o.loc)}{\delta_{max}}$.

Same as [3], we adopt the language model based function to measure the textual relevance of the objects regarding the query q, which is defined as follows.

Definition 2: **Textual relevance** $f_t(q, o)$. Let $w_{t,o}$ denote the weight of the keyword t regarding object o, and

$$w_{t,o} = (1-\xi)\frac{tf_{t,o}}{|o.\mathcal{T}|} + \xi\frac{tf_{t,D}}{|D|}, \quad (6)$$

where $tf_{t,o}$ and $tf_{t,D}$ are the `term frequency` of t in $o.\mathcal{T}$ and D respectively. Here, D represents the text information of all objects in $\mathcal{O}$ and ξ is a smoothing parameter. Then the textual relevance between q and o is defined as follows.

$$f_t(q, o) = 1 - \frac{\prod_{t \in q.\mathcal{T}} w_{t,o}}{maxP} \quad (7)$$

where $maxP$ is used for normalization.

Based on the spatial proximity and textual relevance between the query and the object, the `spatio-textual ranking score` of an object o regarding the query q can be defined as follows.

Definition 3: ***spatio-textual* ranking score** $f(q, o)$**.** Let α be the parameter used to balance the importance of the spatial proximity and textual relevancy, we have $f(q, o) = \alpha \times f_s(q, o) + (1 - \alpha) \times f_t(q, o)$. Note that the objects with **small score values** are preferred (i,e., ranked higher).

Problem Definition. Given a query q and a set $\mathcal{O}$ of objects, we aim to find the top k *spatio-textual* objects with smallest *spatio-textual* ranking score values regarding Q.

Aggregate IL-Quadtree. To efficiently support the spatial keyword ranking query, we slightly argument the IL-Quadtree by hierarchically maintaining the keyword weight information in the IL-Quadtree. For each quadtree node, instead of using one bit to indicate whether there is any object within the node, we keep the maximal keywords weight among the objects within the node, which is named aggregate IL-Quadtree in the paper.

Based on the definition of the *spatio-textual* ranking score, the following lemma is immediate, which implies that we may come up with the lower bound of the ranking score value for an object due to the monotonic property of the scoring function.

Lemma 1: Given two objects o and p, we say o is *dominated* by p regarding a query q if $\delta(o, q) \geq \delta(p, q)$ and $w_{t,o} \leq w_{t,p}$ for every keyword $t \in q.\mathcal{T}$, we have $f(o, q) \geq f(p, q)$.

The following Theorem is immediate based on Lemma 1, which implies that we may avoid accessing objects in a particular region based on the aggregated term weight information.

Theorem 1: Given a region R in the space, for each keyword $t_i \in q.\mathcal{T}$, we use $f_i(R)$ to denote the maximal term weight of the objects within the region. Suppose we have an object p where $\delta(p, q) = \delta_{min}(R, q)$ and $w_{t_i,p} = f_i(R)$ for $i \in I(q.\mathcal{T})$, then for any object o within R, $f(p, q) \leq f(o, q)$.

Spatial Keyword Ranking Algorithm. We suppose objects are organized by an aggregate IL-Quadtree, denoted by aLQ. Let Q be a *virtual* quadtree and a node e in Q is a leaf node regarding the query q if for each keyword $t_i \in q.\mathcal{T}$ there is a *black* leaf node e' in aLQ_i such that $seq(e) \subseteq seq(e')$; that is, for a leaf node e in Q, none of the corresponding nodes (i.e., nodes with the same split sequence as e) is a non-leaf node.

Algorithm 2 illustrates the details of the aggregate IL-Quadtree based spatial keyword ranking approach where Theorem 1 is used to prune non-promising objects and the *best-first* paradigm is adopted. Similar to the *signature* check in Algorithm 1, we can quickly identify the maximal term weights for a node e, i.e., $f_i(e)$ for any $i \in I(q.\mathcal{T})$, based on the split sequence of e. Then, a lower bound of the ranking score value for all objects within e, denoted by $f(e,\ q)$, can be derived based on Theorem 1 at Line 11. Meanwhile, $f(e,\ q)$ serves as the key of a node e in the min heap $\mathcal{H}$ (Line 12). Similarly, the rank score of an object regarding q is its key in $\mathcal{H}$ (Line 19). The algorithm terminates when k best objects are visited (Line 7).

Algorithm 2: **SK-RANKING** (aLQ, k, q)

```
Input  : $aLQ$ : the aggregate IL-Quadtree,
         $k$ : number of objects returned, $q$ : the query
Output : $\mathcal{R}$ : $k$ objects with highest scores
1  $\mathcal{R} := \emptyset$; $\mathcal{H} = \emptyset$;
2  Push root node of the virtual quadtree $Q$ into $H$ ;
3  while $\mathcal{H} \neq \emptyset$ do
4      $e \leftarrow$ the tuple popped from $\mathcal{H}$ ;
5      if $e$ is an object then
6          $\mathcal{R} := \mathcal{R} \cup e$ ;
7          Terminate the Loop if $|\mathcal{R}| = k$ ;
8      else
9          if $e$ is not a leaf node then
10             for each child entry $e'$ do
11                 Compute $f(e',\ q)$;
12                 Push $e'$ into $\mathcal{H}$ ;
13         else
14             $\mathcal{C} := \emptyset$ ;
15             for each quadtree $aLQ_i$ where $i \in I(q.\mathcal{T})$ do
16                 $e' \leftarrow$ the black
                   leaf node in $aLQ_i$ with $seq(e') \subseteq seq(e)$ ;
17                 $\mathcal{C} := \mathcal{C} \cup$ objects in $e'$;
18             for each object $o \in \mathcal{C}$ do
19                 Compute $f(o,\ q)$ and push it to $\mathcal{H}$ ;
20 return $\mathcal{R}$
```

VI. Performance Evaluation

We present results of a comprehensive performance study to evaluate the efficiency and scalability of the proposed techniques in the paper. Especially, the following algorithms are evaluated for the TOPK-SK query.

- **ILQ** IL-Quadtree based TOPK-SK algorithm proposed in Section IV.
- **IVR** The inverted R-tree based TOPK-SK algorithm [1].
- **MIR**^{2} The enhanced IR^2-tree technique [2] based TOPK-SK algorithm.
- **KR** TOPK-SK algorithm developed based on the KR*-tree [5].
- **WIR** WIR-tree technique proposed in [6].
- **IR** ,**CM-CDIR** IR-tree is proposed in [3] and CM-CDIR [4] is one of its optimized variants.

Note that WIR technique [6] can immediately support TOPK-SK query by issuing one query each time. Other indexing techniques (KR, IR and CM-CDIR) can be easily adopted by applying the incremental nearest neighbor search and conducting keyword constraint verification against each tree node visited.

Datasets. Performance of various algorithms are evaluated on both real and synthetic datasets. The following four datasets are employed in the experiments. Real spatial dataset **GN** is obtained from the US Board on Geographic Names[4] in which each object is associated with a geographic location and a short text description. **GN** serves as the default dataset in the experiments. We also generate datasets **Tigers** and **Cars** by obtaining

[4]http://geonames.usgs.gov

the spatial locations from corresponding spatial datasets from Rtree-Portal [5] and randomly geo-tagging these objects with user-generated textual content from 20 Newsgroups [6]. To investigate the scalability of the algorithms, we also include the synthetic dataset **SYN**. A number of objects are randomly chosen from *Cars* dataset, and their corresponding keywords are obtained from a vocabulary whose term frequencies follow the *zipf* distribution where the parameter z varies from 0.9 to 1.3 with default value 1.1. By default, the number of objects (n), the vocabulary size (v) and the number of keywords per object (m) are set to 1 million, 100 thousands and 15 respectively. Table II summaries the important statistics of four datasets.

Property	Tigers	SYN	GN	Cars
number of objects (millions)	0.56	1	2.2	2.25
vocabulary size (thousands)	78	100	208	81
avg. number of keywords per obj.	14.878	15	6.75	26

TABLE II
DATASET DETAILS

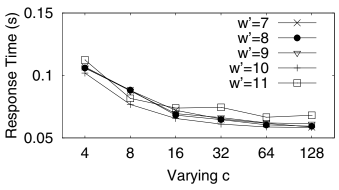


Fig. 11. Tuning IL-Quadtree

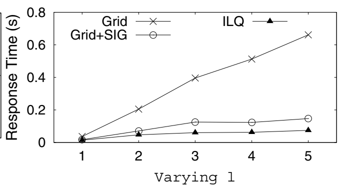


Fig. 12. Comparison with Grid

Workload. A workload for the TOPK-SK query consists of 200 queries, and the average query response time and the average number of disk accessed are employed to evaluate the performance of the algorithms. The query locations are randomly selected from the underlying dataset. On the other hand, the likelihood of a keyword t being chosen as query keyword is $\frac{freq(t)}{\sum_{t_i \in \mathcal{V}} freq(t_i)}$ where $freq(t)$ is the term frequency of t in the dataset. The number of query keywords (l) varies from 1 to 5, and the number of results (k) grows from 10 to 50. By default, l and k are set to **3** and **10** respectively.

All algorithms in the experiments are implemented in Java. The source codes of IR and CM-CDIR [4] are public available, and other algorithms are implemented based on the data structures (e.g., R-tree and B^+ tree) in that source code package. Experiments are run on a PC with Intel Xeon 2.40GHz dual CPU and 4G memory running Debian Linux. All index structures are disk resident, and the page size is fixed to 4096 bytes. We assume it takes $2ms$ for each disk access. For a fair comparison, we tune the important parameters of the competitor algorithms for their best performance. Particularly, for MIR^2 the bitmap length at the leaf node is set to 4160, 5120, 5120 and 7680 bits for *Tigers*, *SYN*, *GN* and *Cars* respectively. The parameter β and the number of clusters are set to 0.1 and 20 for CM-CDIR.

[5]http://www.rtreeportal.org
[6]http://people.csail.mit.edu/jrennie/20Newsgroups

A. Evaluating TOPK-SK query

Tuning IL-Quadtree. In all experiments, the maximal partition level of IL-Quadtree (w) is set to 16. In Fig 11, we tune the performance of ILQ Algorithm by varying minimal depth of the *black* leaf node (w') and the split threshold (c). It is reported that different values of w' and c have no significant impact on the performance when $w' < 11$ and $c > 16$ in the settings. In the experiments, we set $w' = 8$ and $c = 64$ for all datasets.

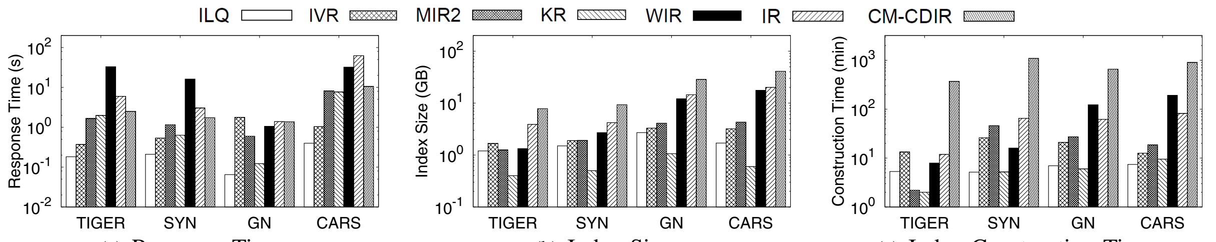
Comparison with Inverted Grid Index. In Fig. 12, Algorithm Grid is the implementation of inverted index technique in [10], which does not utilize any *signature* technique. As expected, the performance of Grid degrades significantly with the increase of the number of query keywords (l) since there is no *pruning* technique for objects within the search region. As discussed in Section III-A, Algorithm Grid+SIG is the extension of Grid by utilizing the space partition based *signature* technique. It significantly enhances the performance of Grid. Nevertheless, its performance is dominated by IL-Quadtree because Grid+SIG cannot capture the objects distribution for each keyword.

Evaluation on different datasets. We investigate the query response time, index construction time and index size of 7 algorithms against four datasets *Tigers*, *SYN*, *GN* and *Cars*, where other parameters are set to default values. In Fig. 13(a), ILQ demonstrates superior performance in comparison with other algorithms. Fig. 13(b) depicts the index sizes occupied by the algorithms. Particularly, KR has the smallest index size among the algorithms since only the R-tree entry identifications are kept for each keyword as shown in Fig. 6. On the other hand, CM-CDIR has the largest index size due to the extra clusters information kept. The construction time of the index structures is depicted in Fig. 13(c). In the rest of this subsection, we exclude IR and CM-CDIR from the performance comparison as both of them are dominated by MIR^2 and KR under all settings.

Effect of the number of query keywords (l). We evaluate the effect of the number of query keywords in Fig. 14. Recall that the performance of the algorithms can be measured by $n_o \times p_s$ in the cost analysis of Section III-A, where n_o is the number of the related objects covered by the search region and p_s is the average *surviving probability* of these n_o objects. Clearly, the growth of l will enlarge the search region, and hence leads to a larger n_o. On the other hand, the pruning capability of the algorithms will increase due to the tighter keyword constraint, i.e., smaller p_s.

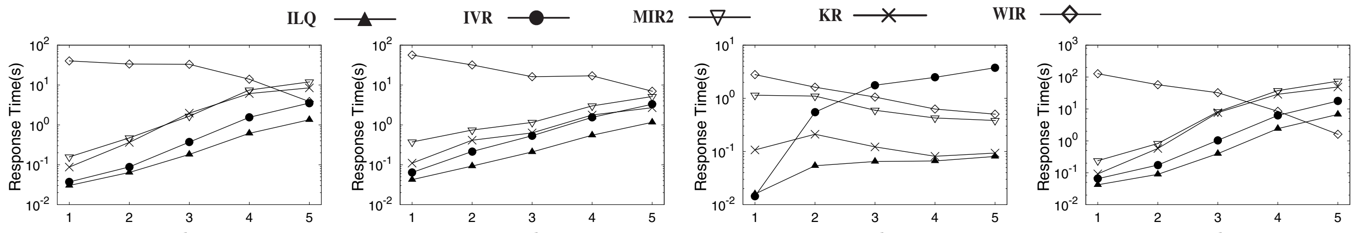
Based on Fig. 14, we can make the following observations.

- **ILQ** and **IVR.** As only the objects containing at least one query keyword may contribute to n_o in ILQ and IVR, they have the best performance when $l = 1$ under all settings. As expected, the performance of IVR degrades dramatically when l grows due to the lack of ability to prune objects. It is reported that the performance of ILQ is much more scalable by making use of the space partition based *signature* technique.



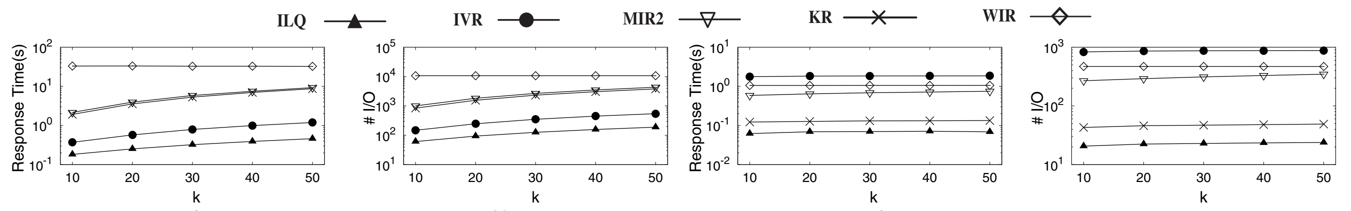
(a) Response Time (b) Index Size (c) Index Construction Time

Fig. 13. Performance over Various Datasets



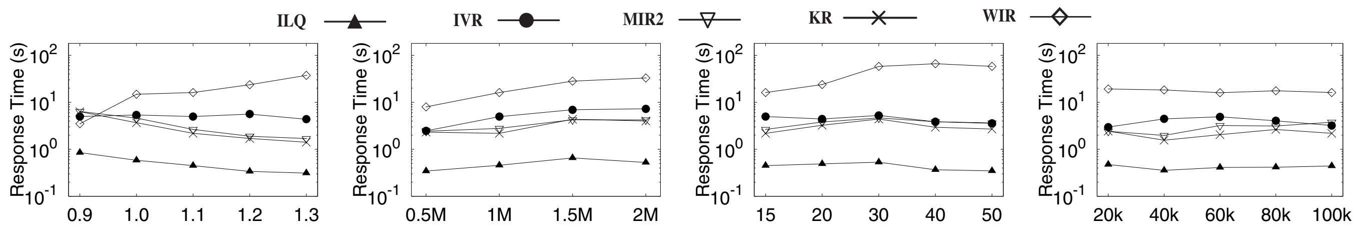
(a) Varying l on *Tigers* (b) Varying l on *SYN* (c) Varying l on *GN* (d) Varying l on *Cars*

Fig. 14. Effect of the the number of query keywords (l)



(a) Response time, *Tigers* (b) I/O, *Tigers* (c) Response time, *GN* (d) I/O, *GN*

Fig. 15. Effect of k



(a) Varying z (b) Varying n (c) Varying m (d) Varying v

Fig. 16. Effect of term frequency skewness (z), number of objects (n), number of keywords per object (m) and vocabulary size (v)

- **MIR^2 and KR.** It is interesting that when l grows, the performance of MIR^2 and KR degrades significantly on datasets *Tigers* (Fig. 14(a)), *SYN* (Fig. 14(b)) and *Cars* (Fig. 14(d)), while it becomes better on dataset *GN* (Fig. 14(c)). This is because the locations of many objects with high-frequency terms are closely clustered in dataset *GN*, and hence the growth of the search region is very slow regarding l. Therefore, the extra cost invoked by enlarging the search region can be easily compensated by the improved pruning capability on dataset *GN*. Recall that all objects within the search region contribute to n_o for MIR^2, KR and WIR. Therefore, the number of objects involved are likely to grow quickly regarding l, which is the case regarding datasets *Tigers*, *SYN* and *Cars*, and hence the performance of MIR^2 and KR is seriously deteriorated by the increase of l in Fig. 14(a), 14(b) and 14(d).
- **WIR.** The trend of WIR is quite different to other algorithms, where the performance is significantly enhanced when the number of query keywords grows. This is because the construction of WIR-tree relies on the keyword partition and hence can take great advantage of a large number of query keywords. However, the performance of WIR is disappointing on all datasets when l is small. The reason is that the construction of WIR may impair the spatial closeness of the objects within the same node of WIR-tree.

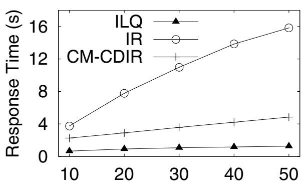
Effect of the number of results (k). Fig. 15 reports the query response time and the number of I/O accesses of the algorithms as a function of k on two datasets *Tigers* and *GN*. As expected, the performance of all algorithms degrades regarding the increase of k (i.e., the larger search region). Compared with Fig. 14, the growth of the searching cost is much slower, specially for the *GN* dataset.

Effect of the term frequency skewness (z). In Fig. 16(a), we evaluate the impact of the term frequency skewness on the algorithms, where z varies from 0.9 to 1.3. As a query keyword is selected based on the term frequencies, the search region of the algorithms shrinks when z increases, and hence the performance of the algorithms (except WIR) improves when z is large. A keyword t with extremely high term frequency is not in favor of WIR-tree construction because it is hard to evenly partition a group. This leads to the poor performance of WIR when z increases.

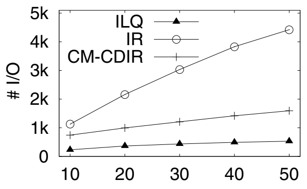
Effect of n, m and v. Fig. 16(b), Fig. 16(c) and Fig. 16(d) report the query response time of the algorithms as a function of the number of objects (n), the average number of keywords per object (m), and the vocabulary size (v). It is shown that the performance of IL-Quadtree is quite scalable against the growth of these parameters.

B. Evaluating Spatial Keyword Ranking Query

To our best knowledge, CM-CDIR [4] is the state of the art technique for the spatial keyword ranking, which is an optimized version of IR-tree by applying the clustering technique. We compare the performance of the aggregate IL-Quadtree based algorithm introduced in Section V-B with IR and CM-CDIR. Fig. 17 reports the performance of three algorithms against the growth of k where *Tigers* dataset is employed. The parameter α in Definition 3 is set to 0.3, and other parameters are set to default values. It is shown that aggregate IL-Quadtree technique outperformances the IR and CM-CDIR algorithms.



(a) Time vs k

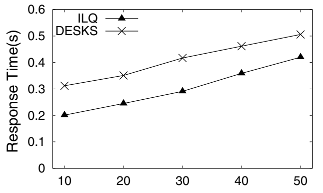


(b) I/O vs k

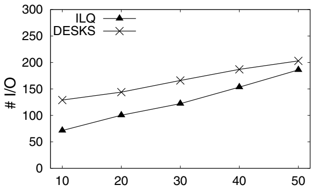
Fig. 17. Spatial Keyword Ranking Query

C. Evaluating Direction-Aware TOPK-SK Query

We compare the performance of the direction-aware TOPK-SK query algorithm introduced in Section V-A with DESKS [12], which is the state of the art technique for this problem. Fig. 18 illustrates the performance of two algorithms on *Cars* dataset where the query response time and the number of I/O accesses are reported as a function of k. The α and β values ($\alpha < \beta$) of the direction constraint are randomly chosen between $[0, \frac{\pi}{2}]$. It is reported that IL-Quadtree technique can also efficiently support the direction-aware TOPK-SK query.



(a) Time vs k



(b) I/O vs k

Fig. 18. Direction-aware Top k Spatial Keyword Search

VII. Conclusions

The problem of top k spatial keyword search is important due to the increasing amount of *spatio-textual* objects collected in a wide spectrum of applications. In the paper, we propose a novel index structure, namely IL-Quadtree, to organize the *spatio-textual* objects. An efficient algorithm is developed to support the top k spatial keyword search by taking advantage of the IL-Quadtree. Our comprehensive experiments convincingly demonstrate the efficiency of our techniques.

Acknowledgment

Ying Zhang is supported by ARC DP110104880, ARC DP130103245 and UNSW ECR grant PS27476. Wenjie Zhang is supported by ARC DP120104168 and ARC DE120102144. Xuemin Lin is supported by ARC DP0987557, ARC DP110102937, ARC DP120104168 and NSFC61021004.

References

[1] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma, "Hybrid index structures for location-based web search," in *CIKM*, 2005.
[2] I. D. Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *ICDE*, 2008.
[3] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *PVLDB*, vol. 2, no. 1, 2009.
[4] D. Wu, G. Cong, and C. S. Jensen, "A framework for efficient spatial web object retrieval," *VLDB J.*, 2012.
[5] R. Hariharan, B. Hore, C. Li, and S. Mehrotra, "Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems," in *SSDBM*, 2007.
[6] D. Wu, M. L. Yiu, G. Cong, and C. S. Jensen, "Joint top k spatial keyword query processing," *TKDE*, 2011.
[7] G. R. Hjaltason and H. Samet, "Distance browsing in spatial databases." *TODS*, vol. 24, no. 2, pp. 265–318, 1999.
[8] S. Vaid, C. B. Jones, H. Joho, and M. Sanderson, "Spatio-textual indexing for geographical search on the web," in *SSTD*, 2005.
[9] Y.-Y. Chen, T. Suel, and A. Markowetz, "Efficient query processing in geographic web search engines," in *SIGMOD Conference*, 2006.
[10] M. Christoforaki, J. He, C. Dimopoulos, A. Markowetz, and T. Suel, "Text vs. space: efficient geo-search query processing," in *CIKM*, 2011.
[11] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg, "Efficient processing of top-k spatial keyword queries," in *SSTD*, 2011.
[12] G. Li, J. Feng, and J. Xu, "Desks: Direction-aware spatial keyword search," in *ICDE*, 2012.
[13] S. B. Roy and K. Chakrabarti, "Location-aware type ahead search on spatial databases: semantics and efficiency," in *SIGMOD Conference*, 2011.
[14] J. B. Rocha-Junior and K. Nørvåg, "Top-k spatial keyword queries on road networks," in *EDBT*, 2012.
[15] A. Khodaei, C. Shahabi, and C. Li, "Hybrid indexing and seamless ranking of spatial and textual features of web documents," in *DEXA*, 2010.
[16] Z. Li, K. C. K. Lee, B. Zheng, W.-C. Lee, D. L. Lee, and X. Wang, "Ir-tree: An efficient index for geographic document search," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 4, 2011.
[17] F. Li, B. Yao, M. Tang, and M. Hadjieleftheriou, "Spatial approximate string search," *TKDE*, 2012.
[18] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa, "Keyword search in spatial databases: Towards searching by document," in *ICDE*, 2009.
[19] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi, "Collective spatial keyword querying," in *SIGMOD Conference*, 2011.
[20] J. Fan, G. Li, L. Zhou, S. Chen, and J. Hu, "Seal: Spatio-textual similarity search," *PVLDB*, vol. 5, no. 9, 2012.
[21] I. Gargantini, "An effective way to represent quadtrees," *Commun. ACM*, vol. 25, no. 12, pp. 905–910, 1982.
[22] G. M. Morton, "A computer oriented geodetic data base and a new technique in file sequencing," *Technical Report, Ottawa, Canada: IBM Ltd*, 1966.
[23] C. Faloutsos, "Multiattribute hashing using gray codes," in *SIGMOD Conference*, 1986.