Inverted Linear Quadtree: Efficient Top K Spatial Keyword Search

Chengyuan Zhang, Ying Zhang, Wenjie Zhang, and Xuemin Lin, *Fellow, IEEE*

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 two fundamental problems in the spatial keyword queries: top k spatial keyword search (TOPK-SK), and batch top k spatial keyword search (BTOPK-SK). Given a set of *spatio-textual* objects, a query location and a set of query keywords, the TOPK-SK retrieves the closest k objects each of which contains all keywords in the query. BTOPK-SK is the batch processing of sets of TOPK-SK queries. 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. To further enhance the filtering capability of the signature of linear quadtree, we propose a partition based method. In addition, to deal with BTOPK-SK, we design a new computing paradigm which partition the queries into groups based on both spatial proximity and the textual relevance between queries. We show that the IL-Quadtree technique can also efficiently support BTOPK-SK. Comprehensive experiments on real and synthetic data clearly demonstrate the efficiency of our methods.

Index Terms—Spatial, keyword, batch

1 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.

- *C. Zhang and W. Zhang are with the University of New South Wales, Australia. E-mail: {zhangc, zhangw}@cse.unsw.edu.au.*
- *Y. Zhang is with the University of Technology, Sydney, Australia and the University of New South Wales, Australia. E-mail: yingz@cse.unsw.edu.au.*
- *X. Lin is with East China Normal University, China, and the University of New South Wales, Australia. E-mail: lxue@cse.unsw.edu.au.*

Manuscript received 12 Apr. 2015; revised 20 Dec. 2015; accepted 30 Jan. 2016. Date of publication 15 Feb. 2016; date of current version 1 June 2016. Recommended for acceptance by F. Afrati.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2016.2530060

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.

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 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 many real applications, the query workload may vary from time to time, and the system may encounter a burst of queries (e.g., queries invoked by a particular event). In this scenario, the system throughout is poor if a large number of queries are processed one by one. Motivated by this, a large body of existing work (e.g., [3], [4]) have been devoted to investigate how to improve the system throughout with the batch query processing techniques such that a large number of queries in the queue can be processed with a reasonable delay. Meanwhile, nowadays, a large volume of spatial keyword queries may be generated in a short time period. For instance, a large number of queries may be imposed to seek nearby restaurants during the dinner and lunch time. An important local event may incur a large number of relevant

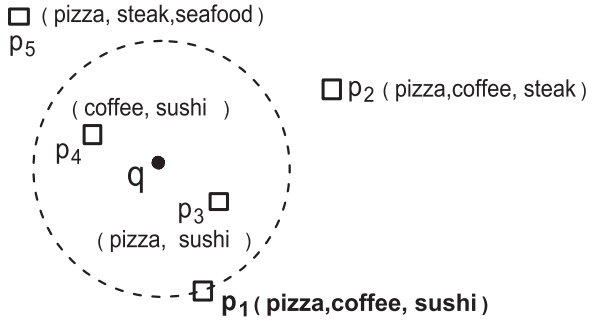


Fig. 1. Online yellow pages example.

queries. As shown in [4], a large number of fake spatial keyword searches may be issued in order to protect the users privacy. This may lead to dramatic degrade of the system throughout if queries are processed individually. To alleviate this issue, we also investigate the problem of batch spatial keyword query (BTOPK-SK) which aims to efficiently support a large number of spatial keyword queries at the same time.

1.1 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. In the paper, we propose the inverted linear quadtree (IL-Quadtree) indexing technique which naturally combines the spatial and textual features of the objects. 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. We further enhance the pruning capability of the signature technique by partitioning objects into groups.

The key challenge of the BTOPK-SK problem is the computation sharing of the spatial keyword queries. In this paper, we develop effective partition approach to group queries such that the queries in the same groups can share computation costs because they are likely to have common search pathes. Based on the inverted linear quadtree (IL-Quadtree) and the groups of the queries, we devise efficient batch processing algorithm to support BTOPK-SK queries.

1.2 Contributions

The principle contributions of this paper are 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.
- To further improve the efficiency of IL-Quadtree, we propose a novel partition based method to enhance the filtering capability of the signature technique.
- Efficient batch spatial keyword query processing techniques are developed to improve the throughout of the system when there are lots of spatial keyword queries.
- Comprehensive experiments demonstrate the effectiveness and efficiency of our techniques.

TABLE 1
The Summary of Notations

Notation	Definition
o (q)	a *spatio-textual* object (query)
$\mathcal{Q}$	a *spatio-textual* batch 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
$\mathcal{Q}.mbr$	minimum bounding rectangle of batch query $\mathcal{Q}$
$\mathcal{Q}.\mathcal{T}$	a set of query keywords for batch query $\mathcal{Q}$
$\mathcal{Q}.\mu$	is the smallest keyword set size of a subquery in $\mathcal{Q}$
$\mathcal{V}$, 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
h	the number of queries
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

1.3 Roadmap

The rest of the paper is organized as follows. Section 2 formally defines the problem of top k spatial keyword search, followed by the introduction of the related work. Section 3 presents the IL-Quadtree structure. Section 4 proposes an efficient top k keyword search algorithm. Section 5 introduces the advanced signature techniques based on object partitions. Section 6 presents efficient query processing techniques for batch spatial keyword queries. Experimental results are reported in Section 7. Section 8 concludes the paper.

2 Preliminary

In this section, we first formally define the problem of top k spatial keyword search, and then introduce the related work. Table 1 below summarizes the mathematical notations used throughout this paper.

2.1 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.

2.1.1 TOPK-SK

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 $o.loc$ ($q.loc$).

In this paper, a batch top k spatial keyword query, denoted by $\mathcal{Q}$, consists of a set of top k spatial keyword queries. We use $\mathcal{Q}.mbr$ to denote the minimum bounding rectangle of the locations of the queries in $\mathcal{Q}$; $\mathcal{Q}.\mathcal{T}$ represents the union of the keyword sets of the queries in $\mathcal{Q}$; and $\mathcal{Q}.\mu$ is the smallest keyword set size of the queries in $\mathcal{Q}$.

2.1.2 BTOPK-SK

Given a set $\mathcal{O}$ of *spatio-textual* objects, and a batch spatial keyword query (BTOPK-SK) $\mathcal{Q}$, we aim to find the top k results for each individual top k spatial keyword query in $\mathcal{Q}$.

2.2 Related Work

In this section, we first present the existing techniques for the problem of TOPK-SK query as well as some other variants of top k spatial keyword search. Then other spatial keyword related queries are introduced.

Considering the indexing scheme used in existing works, we classify the indexes into two categories, namely *Keyword First Index* and *Spatial First Index*.

2.2.1 Keyword First Index

Keyword First Index first employs keyword index to extract the related inverted indexes, then exploits spatial index for spatial filtering. To facilitate the spatial keyword search, it is 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 query, we can simultaneously apply the incremental nearest neighbor search [5] on the related spatial indexes until k objects satisfying the keyword constraint are retrieved. Such indexes include inverted R-tree [1], SFC-QUAD [6], and S2I [7].

The inverted R-tree is proposed in [1] to organize objects for each keyword. For each distinct keyword $t \in \mathcal{V}$, a separate R-tree is built for the objects in which t appears.

SFC-QUAD, which combines the space filling curve and inverted file, is proposed by Christoforaki et al. in [6]. Based on the frequency of corresponding keywords, Rocha-Junior et al. [7] proposed an index called S2I that combines R-tree/block and inverted file.

2.2.2 Spatial First Index

Intuitively, the *Keyword First Index* is efficient when there is only one query keyword since we only need to issue a k nearest neighbor search on the corresponding spatial index. Nevertheless, the performance of the *Keyword First Index* 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. In order to alleviate the dilemma of *Keyword First Index*, *Spatial First Index* is proposed. *Spatial First Index* first utilizes spatial index for spatial pruning, then employs keyword index to grab the corresponding inverted index. Such indexes comprise IR^2-tree [2], KR*-tree [8], IR-tree and its variants [9], [10], WIBR-tree [4], and SKI [11].

Felipe et al. [2] propose an index structure called IR^2-tree, which integrates signature file into each node of the R-tree. In [8], Hariharan et al. propose the KR*-tree structure to process the region based spatial keyword search. Cong et al. [9], [10] and Li et al. [12] independently proposed an IR-tree index, which primarily indexes the data using an R-tree, but creates an inverted file for each node of the tree. To distinguish them, we refer IR-tree from Li et al. as the IRLi-tree. The difference between the IR-tree and the IRLi-tree is that the IR-tree stores the inverted files for each node separately while the IRLi-tree stores one integrated inverted file for all the nodes. Some optimized variants of the IR-tree,such as DIR-tree and CDIR-tree, are introduced in [9], [10]. Cary et al. [11] proposed SKI, which combines R-tree and inverted files. The parent node of a leaf node, which is called a super node, is associated with a bitmap inverted file.

WIBR-tree, which is also a variant of IR-tree, is proposed in [4] to efficiently support a batch of TOPK-SK queries. The structure of WIBR-tree is the same as that of IR-tree. The main difference is that the construction of WIBR-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 ith 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 WIBR-tree is constructed based on these groups.

2.2.3 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 [7], [9], [10], [12], [13] is proposed to rank objects based on a scoring function which considers the distance as well as the textual relevance.

In [14], Li et al. study the problem of top k spatial keyword search in which the direction constraint is considered. Other variants include location-aware type ahead search [15], top k spatial keyword search on road network [16], [17], etc.

3 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 3.1 we describe the shortcomings of the existing indexing approaches. To address

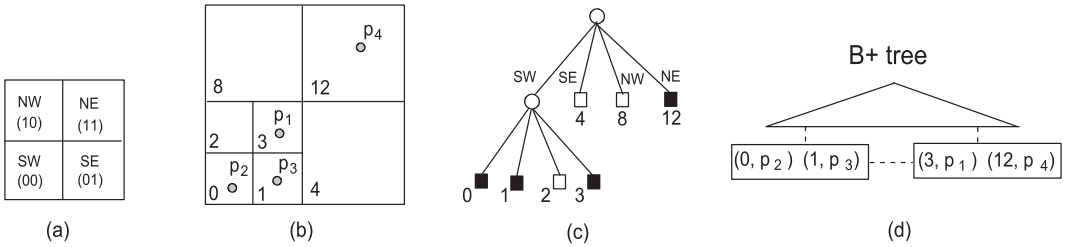


Fig. 2. Linear quadtree example.

these shortcomings, Section 3.2 proposes the inverted linear quadtree based index structure.

3.1 Motivation

To deal with the TOPK-SK problem, two great challenges faced by current works: i) how to effectively reduce the number of objects with the search region. ii) how to effectively decrease the average *surviving probability* of these objects (i.e., an object within the search region is expected to be loaded). Motivated by these, we should develop a new index structure has following properties. *First*, the index structure should fall 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 eliminated. *Second*, the new index structure should be adaptive to the distribution of the objects for each keyword. *Third*, we need to exploit the *AND* semantic, i.e., pruning a group of objects which do not satisfy the keyword constraint.

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.[1] Clearly, the new structure proposed satisfies the above-mentioned three important criteria of the spatial keyword indexing method.

3.2 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 [18] 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 [19], Hilbert code and gray code [20]).

In the paper, we encode the quadtree nodes based on the Morton code [19] (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.

1. In Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2016.2530060, the performance of different node fanouts demonstrates the efficacy of quadtree.

Now we describe how to derive the Morton code of a node based on its split sequence in two-dimensional space. As shown in Fig. 2a, 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.[2] Then the code can be derived by concatenating the split codes in each subdivision. For example, Figs. 2b and 2c 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. 2c, 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 $1, 100$ where the last two binary digits are padding. In Figs. 2b and 2c, 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. 2d 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.

3.2.1 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, at most 2 bits are kept for each node of the quadtree, which are set to 11 for *black* leaf nodes, 10 for non-leaf nodes, and 0 otherwise. Obviously, a node in LQ_i is empty (i.e., it does not contain any object with keyword t_i) if the bit is set to 0. Given two keywords t_1 and t_2, and a set of points, Fig. 3 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.

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

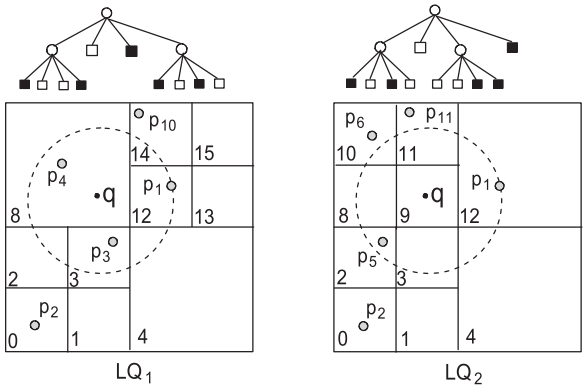


Fig. 3. Example of IL-Quadtree.

3.2.2 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.

For a given keyword, we only build quad-tree and B-tree when its corresponding objects cannot be fit into one disk page; otherwise, only one disk page (a block) was used to store the objects of this keyword.

4 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-Quadtrees. Same as other inverted index based approaches, we also conduct incremental nearest neighbor search on the IL-Quadtrees. 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 1 below shows the motivation of our algorithm.

Example 1. Given the TOPK-SK query q, where $q.\mathcal{T} = \{t_1, t_2\}$ 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. The objects, which not appear in LQ_1 or LQ_2, are immediately eliminated. 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 e 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 kth closest distance in $\mathcal{R}$. Lines 13-19 first calculate the distance between the object o and q if the counter of o reaches l, i.e., o contains all query keywords, and then update $\mathcal{R}$ and λ if the distance between the object o and q is smaller than λ. Otherwise, we terminate the algorithm. When the popped node e is a non-leaf node (Line 20), 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 (Lines 21-23). 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: R : TOPK-SK query result.
1  R = ∅; H = ∅; λ = ∞;
2  for each LQ_i where i ∈ I(q.T) do
3    Push root node of LQ_i into H;
4  while H ≠ ∅ do
5    e ← the quadtree node popped from H;
6    if e is a black leaf node
7      Suppose e is from LQ_i;
8      for each LQ_j where j ≠ i and j ∈ I(q.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 δ(o, q);
16           if δ(o, q) > λ then
17             Break;
18           else
19             Update λ and R;
20   else if e is a non-leaf node then
21     for each child node e' of e do
22       if e is not a white leaf node and δ_min(e', q) ≤ λ then
23         Push e' into H;
24 return R
```

Remark 1. In the implementation, we randomly select one of the query keywords and put the root of corresponding quad-tree into the heap. Then, when a node is popped from the heap, we use the signatures of corresponding nodes of other quad-trees to decide whether the children of current node should be further explored.

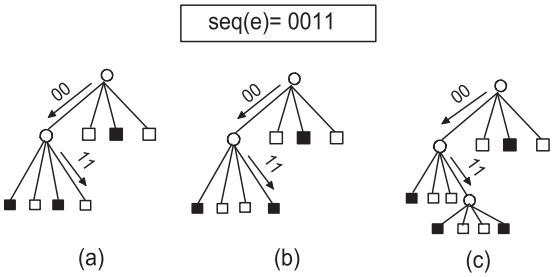


Fig. 4. *Signature* check.

Remark 2. If the query keyword is stored in block (page), we first load the block (page) into memory. Then, we extract the relevant objects from the block (page), and dynamically build the corresponding quad-tree by these objects.

4.1 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. 4, there are three cases for a given split sequence $seq(e)$ and quadtree LQ_j: (i) encounter a *white* node (Fig. 4a), (ii) encounter a *black* node (Fig. 4b) and (iii) end up at a non-leaf node (Fig. 4c). 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. Second, 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.

5 Enhancement of the Signature Technique

In this section, we enhance the effectiveness of the signature technique by partitioning the objects into several groups. We first introduce the motivation of the method, then a heuristic based method is proposed.

5.1 Motivation

In Section 3, for a cell c and a term t, we use a signature (1 or 0) to indicate if there exists an object o which resides in c and contains term t. We do not need to explore a cell c if the sigature is 0 for any query keyword due to the AND semantics of our spatial keyword search. Intuitively, in addition to the spatial and keyword features, we may further improve the filtering capacity if the objects are partitioned into η groups; that is, for the cell c and the term t, we use η signatures to summarize the relevant objects.

As shown in Fig. 5a, given a region r, suppose we have five objects $o_1(t_1, t_3)$, $o_2(t_2, t_3)$, $o_3(t_1)$, $o_4(t_4)$, and o_5 (t_1, t_4) in this region r and the vocabulary $\mathcal{V} = \{t_1, t_2, t_3, t_4, t_5\}$. The signatures of five keywords are shown in Fig. 5b where $I(r, t_1) = 1$, $I(r, t_2) = 1$, $I(r, t_3) = 1$, $I(r, t_4) = 1$, and

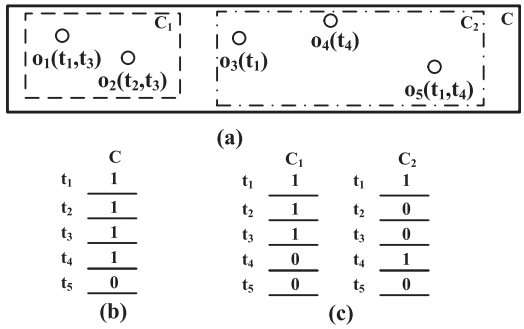


Fig. 5. Example of object partition.

$I(r, t_5) = 0$ respectively. Given a query q with $q.T = \{t_2, t_4\}$, all objects will be loaded if r is accessed in Algorithm 1 since $I(r, t_2) = 1$ and $I(r, t_4) = 1$. However, none of the objects contains both t_2 and t_4. We say this is a *false hit* if a region passes the signature test of a query but does not return any object satisfying the keyword constraint. Similarly, a query q with $q.T = \{t_3, t_4\}$ may result in a *false hit* as well. Note that it is a *true hit* regarding $q.T = \{t_1, t_3\}$ since the object o_1 contains both t_1 and t_3. And $q.T = \{t_1, t_5\}$ is also not a *false hit* since it fails the signature test. Intuitively, we can classify the objects in the region r into two clusters c_1 and c_2 as shown in Fig. 5a, and the signature of r can be refined as shown in Fig. 5c. Then we can avoid loading objects resulted from the false hit when $q.T = \{t_2, t_4\}$ since it fails the signature tests for both c_1 and c_2. In this paper, the filtering capacity of the signature techniques can be naturally evaluated by the number of false hits.

In this paper, we assume objects are partitioned into η groups. In Fig. 5a, we have $C = \{c_1, c_2\}$ with $\eta = 2$. There are $\binom{2}{n}$ possible partitions if there are n objects in region r. We aim to identify the partition with the minimal false hit cost. Specifically, for a given TOPK-SK query q, $\xi(q, r)$ denotes the *false hit cost* of r (i.e., the number of objects loaded due to the *false hits* of the region r). Similarly, we can define the *false hit cost* of a partition C, denoted by $\xi(q, C)$, where

$$\xi(q, C) = \sum_{c' \in C} \xi(q, c'). \tag{1}$$

Note that $\xi(q, r) = 0$ if i) r is not visited, ii) r fails the signature test, or iii) it is a true hit (i.e., find an object satisfying the keyword constraint). Regarding the example in Fig. 5, assume that $\mathcal{Q} = \{q_1, q_2, q_3\}$ where $q_1.T = \{t_1, t_3\}$, $q_2.T = \{t_2, t_4\}$ and $q_3.T = \{t_1, t_2\}$, we have $\xi(q_1, r) = 0$, $\xi(q_2, r) = 5$, and $\xi(q_3, r) = 5$. Similarly we have $\xi(q_1, C) = \xi(q_1, c_1) + \xi(q_1, c_2) = 0$, $\xi(q_2, C) = \xi(q_2, c_1) + \xi(q_2, c_2) = 0$, and $\xi(q_3, C) = \xi(q_3, c_1) + \xi(q_3, c_2) = 2 + 0 = 2$. Let $\xi(\mathcal{Q}, C)$ denotes the *false hit cost* of the partition C and $Pr(q)$ denotes the probability that a query $q \in \mathcal{Q}$ is issued, we have

$$\xi(\mathcal{Q}, C) = \sum_{q \in Q} \xi(q, C) \times Pr(q). \tag{2}$$

5.2 Partition Algorithm

Assume there are n objects in the dataset, we aim to partition these objects into η groups/clusters so that the number of false hits can be significantly reduced. As it is infeasible

to enumerate all possible partitions to achieve the minimal false hit costs in Equation (2), we resort to some heuristics approaches in this paper.

According to the definition of *false hit*, the partition of two objects which is far from each other will not affect the false hit costs even if they share the same terms. Consequently, we assume the space is partitioned into a set of cells by the quadtree, and the objects within each individual cell will be partitioned into η clusters.

Algorithm 2 illustrates the details of our greedy algorithm.[3] Assume there are three global variables η, γ, and $\mathcal{C}$, where η is the number of clusters,[4] γ is the threshold of term frequency,[5] and $\mathcal{C}$ denotes the η clusters that the objects should be partitioned. The greedy algorithm includes two stages: (I) Lines 2-6 depicts the preparation before partitioning the objects into η clusters. (II) Lines 7-20 describes how the algorithm partitions the input objects into η clusters with the help of false hit pairs.

In stage I, Line 2 extracts the terms whose local frequency are larger than γ from the input objects and adds them into high frequency set $hfSet$. With the assistance of $hfSet$, the algorithm enumerates all the unique term pairs from $hfSet$ in Line 3. For each object in input objects, Lines 4 first extracts the high frequency terms included in $hfSet$, then enumerates the unique term pairs on these high frequency terms and inserts these pairs into hit pair set $hpSet$. Next, to gain the *false* hit pairs, the algorithm removes the *true* hit pairs from $fpSet$ in Line 5. Then, for each query in query log, if the terms of the query are fully covered by the terms of input objects, we add the query to influence set Q for further processing in Line 6, otherwise we skip this query.

In stage II, based on the input objects, for each pair in $fpList$, the algorithm first assigns the objects into two clusters by the terms of selected pair in Lines 8-12. Then, the false hit cost values between these two clusters and influence set were computed respectively based on Equation (2), and the summation of these two values compares with the current minimum false hit cost min in Line 13. If the sum value is smaller than min, this term pair is considered as a possible term pair, the value of min is replaced by the sum value, and the two candidate lists are superseded by the two clusters in Line 14. Otherwise it cannot contribute to current partition. After the partition has been decided, the selected pair is removed from $fpList$ and the the value of depth $\mathcal{D}$ increases one in Line 15. Next, if η is still large than $2^{\mathcal{D}}$, the two candidate lists require further partition in Lines 17-18. Otherwise we update $\mathcal{C}$ by these two candidate lists in Line 20.

5.3 Time Complexity

Suppose there are n_p pairs in $fpList$, n_o objects within the cell, and each object at most contains t_o unique terms. Hence, based on a selected pair, the cost of partitioning the objects into two clusters is n_o. Assume there are n_q queries in the query log, each query at most contains t_q unique terms. Given a query, the cost of computing the false hit cost between the query and this partition is at most $t_q \times t_o$. The cost of computing the false hit cost between the query log and this partition is at most $n_q \times t_q \times t_o$. Thus, the cost of each loop is $n_p \times (n_o + n_q \times t_q \times t_o)$. Because the objects need to be clustered into η clusters and the objects are partitioned into two clusters in each loop, the computation time of Algorithm 2 is $n_p \times (n_o + n_q \times t_q \times t_o) \times log_2(\eta)$ in the worse case.

Algorithm 2. Partition($\mathcal{O}$, $\mathcal{Q}$, $\mathcal{D}$)

Input: $\mathcal{O}$: the spatial textual objects, $\mathcal{Q}$: the query log
$\mathcal{D}$: the depth of recursion

1 $hfSet = \emptyset$; $hpSet = \emptyset$; $fpSet = \emptyset$; $fpList = \emptyset$;
2 $hfSet \leftarrow$ extracts the terms whose local frequency larger than γ;
3 $fpSet \leftarrow$ enumerates all unique term pairs from $hfSet$;
4 $hpSet \leftarrow$ from objects in $\mathcal{O}$ enumerate the unique term pairs;
5 $fpList \leftarrow fpSet \setminus hpSet$;
6 $Q \leftarrow$ extracts the query whose terms covered by $\mathcal{O}$ from $\mathcal{Q}\emptyset$
7 $cList_1 = \emptyset$; $cList_2 = \emptyset$; $min = \inf$; $p_{sel} = \emptyset$;
8 **for** each pair $p_i \in fpList$ **do**
9 assume t_j is the j-th term from p_i; $tList_1 = \emptyset$; $tList_2 = \emptyset$;
10 $tList_1 \leftarrow$ the objects in $\mathcal{O}$ contained t_1;
11 $tList_2 \leftarrow$ the objects in $\mathcal{O}$ contained t_2;
12 the other objects assign to the list that has higher similarity;
13 **if** $\xi(Q, \mathcal{C}_{tList_1}) + \xi(Q, \mathcal{C}_{tList_2}) \leq min$ **then**
14 update min, $cList1$, $cList2$, and p_{sel};
15 $fpList \leftarrow fpList \setminus p_{sel}$; $\mathcal{D} = \mathcal{D} + 1$;
16 **if** $\eta > 2^{\mathcal{D}}$ **then**
17 $\mathcal{C} \leftarrow \mathcal{C} \bigcup$ Partition($cList_1$, $fpList$, Q, $\eta/2$);
18 $\mathcal{C} \leftarrow \mathcal{C} \bigcup$ Partition($cList_2$, $fpList$, Q, $\eta/2$);
19 **else**
20 update $\mathcal{C}$ by $cList_1$, and $cList_2$;

5.4 Query Processing

In this paper, the IL-Quadtree with advanced signature techniques is called Enhanced Inverted Linear Quad-tree (EIL-Quadtree for short). The corresponding query processing algorithm is the same as Algorithm 1, except the signature test procedure (Line 9), in which η signatures will be involved in the test instead of one. In particular, the search will stop on a node if its η signature tests fail.

6 Batch Top k Spatial Keyword Search

In this section, we investigate how to effectively process the batch top-k spatial keyword queries. Section 6.1 introduces the challenges and motivations, and Section 6.2 develops an algorithm to efficiently partition the queries to groups. Section 6.3 presents an efficient algorithm to support batch search for IL-Quadtree based on the aggregation of the signatures from relevant keywords.

6.1 Motivation

Intuitively, the system throughout may be deteriorated if we separately process each individual query in the BTOPK-SK, especially when the volume of the queries is large. Therefore, it is desirable to partition the queries into groups so that the queries in the same group may share computation and hence significantly reduce the overall

3. In Appendix B, available online, we explore and evaluate our greedy algorithm and possible clustering methods.

4. The value of η sets to 2^n.

5. In Appendix C, available online, we investigate the impact of parameters γ.

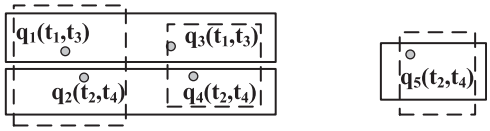


Fig. 6. Example of query partition.

costs. Moreover, novel query processing techniques are necessary to cope with groups of queries based on the IL-Quadtree proposed in this paper.

6.1.1 Effective Query Partition

Query partition plays an important role in batch query processing because it can significantly reduce the processing time by grouping similar queries so that the CPU and I/O costs can be shared between queries in the same group. This strategy has been widely adopted in many research papers such as [4]. However, this method only considers spatial information, and not explicitly considers textual composition. Thus, we take both spatial feature and textual information into consideration to further improve the effectiveness of query partition. Below is a motivating example.

Consider the example of (Fig. 6) that shows five queries $q_1, \ldots, q_5$ in a two-dimensional space. We aim to partition these queries into three groups to enhance the search performance. In [4], the queries will be partitioned to three groups that are surrounded by dotted lines. However, intuitively it is less effective compared with another alternative which are enclosed by solid lines. Because the ignorance of keyword similarities may miss the opportunity to share computation from keywords perspective. Thus, not only the spatial locations, but also the textual information of the queries should be considered while partitioning the queries to different groups.

6.1.2 Efficient Batch Query Processing

After dividing the batch query into a set of groups, it is essential to devise new query processing techniques based on IL-Quadtree structure proposed in this paper. The key is the computation sharing strategies. We carefully maintain the objects loaded during the query processing so that the location information of these objects can be used to facilitate the computation. More specifically, assume there are a set of objects loaded from cell c of LQ_t where c is a black leaf node of LQ_t. If there are some candidate queries on c, we can further push down these objects for LQ_t, i.e., generate refined signatures for child cells of c on the fly, to facilitate the signature tests of these queries.

6.2 Query Partition

In the following, we first introduce how to effectively calculate the spatial proximity and the textual relevance between a query and a query group. Then an efficient group algorithm, based on spatio-textual ranking score, is presented.

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 query group $\mathcal{Q}$, denoted by $f_s(q, \mathcal{Q})$, is defined as $\frac{\delta(q.loc, \mathcal{Q}.mbr)}{\delta_{max}}$.*

Same as [9], we adopt the language model based function to measure the textual relevance of the group regarding the query q, which is defined as follows.

Definition 2 (Textual relevance $f_t(q, \mathcal{Q})$). *Let $w_{t,\mathcal{Q}}$ denote the weight of the keyword t regarding query group $\mathcal{Q}$, and*

$$w_{t,\mathcal{Q}} = (1-\xi)\frac{tf_{t,\mathcal{Q}}}{|\mathcal{Q}.\mathcal{T}|} + \xi\frac{tf_{t,D}}{|D|}, \quad (3)$$

where $tf_{t,\mathcal{Q}}$ and $tf_{t,D}$ are the **term frequency** *of t in $\mathcal{Q}.\mathcal{T}$ and D respectively. Here, D represents the text information of all queries in input query set and ξ is a smoothing parameter. Then the textual relevance between q and $\mathcal{Q}$ is defined as follows:*

$$f_t(q, \mathcal{Q}) = 1 - \frac{\prod_{t \in q.\mathcal{T}} w_{t,\mathcal{Q}}}{maxP}, \quad (4)$$

where $maxP$ is used for normalization.

Based on the spatial proximity and textual relevance between the query and the query group, the `spatio-textual ranking score` of a query group $\mathcal{Q}$ regarding the query q can be defined as follows.

Definition 3 (spatio-textual ranking score $f(q, \mathcal{Q})$). *Let α be the parameter used to balance the importance of the spatial proximity and textual relevancy, we have $f(q, \mathcal{Q}) = \alpha \times f_s(q, \mathcal{Q}) + (1-\alpha) \times f_t(q, \mathcal{Q})$.[6] Note that the queries with* ***small score values*** *are preferred (i.e., ranked higher).*

6.2.1 Grouping Algorithm

Based on the definition of *spatio-textual* ranking score, Algorithm 3 illustrates the details of the Grouping algorithm. In (Lines 1-2), the z-order values are first calculated based on the spatial locations of the queries. Next, based on these z-order values, k points are uniformly selected from these queries as the seeding center points of the groups in Line 3. Then, the remaining queries will be assigned to the most similar groups based on their *spatio-textual* ranking score value in (Lines 5-14) following the k means clustering computation paradigm. Fig. 6 illustrates an example of query partition where three groups are enclosed by solid lines and the queries in the same group are similar on both spatial and textual perspectives.

6.3 IL-Quadtree Based BTOPK-SK Algorithm

In this section, we first introduce the concept of aggregate node and then present a batch search algorithm base on the IL-Quadtree.

6.3.1 Virtual Quadtree

The linear quadtree for each individual term is a regular space partition based spatial index. Therefore, for a given virtual quadtree node (i.e., a space partition), we can aggregate the node information (i.e., signature and objects) from related quadtrees based on terms involved. Each virtual node consists of three components: 1) the identifies of the corresponding quatrees nodes involved in the computation;

6. In Appendix D, available online, we investigate the impact of parameters α.

2) flags to indicate whether the corresponding quadtree nodes are leaf nodes; 3) object list which records the objects which have been loaded from the related linear quadtrees, if current node is a leaf node in that linear quadtree. During the batch query processing, for each query group a virtual quadtree is built on the fly by dynamically aggregating relevant quadtree nodes information.

6.3.2 Algorithm

Algorithm 4 illustrates the details of the batch top k spatial keyword query processsing. Specifically, the algorithm first initializes a root node e in Line 2. Then, for each query $q_i \in Q$, a maximum heap $\mathcal{R}_i$ employed to maintain the top-k objects of q_i in Lines 4-5. The value of $q_i.\lambda$, which denotes the maximum distance in $\mathcal{R}_i$ is initialized to infinity in 6. The identities of queries in the query group were added to the query list of the root node in Line 7. Next, for all terms in the query group, the root node e aggregates the root nodes from corresponding linear quadtrees in Lines 8-11. The root node e is pushed into the minimum heap $\mathcal{H}$, and λ_{max} is initialized to infinity in Line 12. Then, we execute the while loop (Lines 13-31) until the top-k results of each query in query group are ultimately reported in Line 32.

In each iteration, the top entry e with minimum score is popped from $\mathcal{H}$. More specifically, two stages included in each iteration. (I) for each dequeued entry e, we detect whether it can contribute a closer and relevant result for some queries in (Lines 13-19). (II) we tackle the child entries of entry e that passes detection in (Lines 20-31). In stage I, we first check whether the entry e's minimum value e_{min} is larger than the λ_{max}, which denotes the maximum distance in $\mathcal{R}$. If e_{min} is larger, the algorithm breaks in Line 16. Otherwise, set $\mathcal{Q}'$, which contains the queries that have the possibility to obtain closer and relevant results from the subentries of e, is computed. Then, we check whether $\mathcal{Q}'$ is empty or not. If it is empty, the while loop continues, otherwise the iteration continues.

Algorithm 3. Grouping ($\mathcal{Q}$, k)

Input: $\mathcal{Q}$: a set of *spatio-textual* queries
k : the number of group assigned,
Output: $\mathcal{G}$: a set of queries assigned to k groups.
1 **for** each query $q_i \in \mathcal{Q}$ **do**
2 $q_i.val \leftarrow$ calculate z-order value for q_i;
3 $\mathcal{G} \leftarrow$ sample k points uniformly based on queries' z-order values;
4 $isMove \leftarrow$ true;
5 **while** $isMove$ **do**
6 **for** each query $q_i \in \mathcal{Q}$ **do**
7 $score \leftarrow \infty$, $sel \leftarrow 0$;
8 **for** each cluster $g_j \in \mathcal{G}$ **do**
9 **if** $f(q_j, g_i) < q_{score}$ **then**
10 $score \leftarrow f(q_j, g_i)$; $sel \leftarrow j$;
11 $\mathcal{G}_{sel} \leftarrow \mathcal{G}_{sel} \bigcup q_i$;
12 recalculate the k centroids;
13 **if** the k centroids no longer move **then**
14 $isMove \leftarrow false$;
15 **return** $\mathcal{G}$

If entry e passes all tests, we enter stage II. For each term t in $e.\mathcal{T}$, we progressively check whether the object list of selected term of e is empty in Lines 20-22. If the object list is empty we jump to the next term, otherwise the objects in the object list allocated to e's child entries by their location information. Next, for each child entries e' of e, we compute whether the minimum distance from the entry e' to the remain queries in the set $\mathcal{Q}'$ is smaller than maximum distance $\mathcal{Q}'.\lambda$ and whether the conjunct keywords between the set $\mathcal{Q}'$ and entry e' is larger than the $\mathcal{Q}.\mu$. If both constraints satisfied, similar to stage I, we further compute a set $\mathcal{Q}^*$ which consists of a set of queries that share sufficient common keywords and close to entry e' in Line 25. Otherwise, we hop to next child entry. Then, if $\mathcal{Q}^*$ is not empty and the level of current entry e' is smaller than the minimal depth w', we update the child entry e' by $\mathcal{Q}^*$ in Line 30 and push e' into heap Line 31. Otherwise, we further process the node e' before it is pushed into the heap.

Algorithm 4. BTOPK-SK ($\mathcal{Q}$, k)

Input: k : number of objects returned, $\mathcal{Q}$: the query group,
Output: $\mathcal{R}$: top-k results of each query in $\mathcal{Q}$
1 $\mathcal{R} := \emptyset$; $\mathcal{H} = \emptyset$ $\lambda_{max} = \infty$ $\mathcal{Q}.\mathcal{T} = \emptyset$;
2 $e \leftarrow$ new a root node;
3 **for** each subquery $q_i \in \mathcal{Q}$ **do**
4 $\mathcal{R}_i \leftarrow$ new max heap;
5 initialize $\mathcal{R}_i$ with k null object with distance ∞;
6 let $q_i.\lambda$ be the maximum distance in $\mathcal{R}_i$; $q_i.\lambda \leftarrow \infty$;
7 $e.\mathcal{Q} \leftarrow e.\mathcal{Q} \cup q_i$;
8 **for** each LQ_t where $t \in I(q_i.\mathcal{T})$ **do**
9 **if** $t \notin e.\mathcal{T}$ **then**
10 $e.\mathcal{T} \leftarrow e.\mathcal{T} \cup t$;
11 update e by the root node of LQ_t;
12 Push e into $\mathcal{H}$, $\lambda_{max} \leftarrow \infty$;
13 **while** $\mathcal{H} \neq \emptyset$ **do**
14 $e \leftarrow$ the node popped from $\mathcal{H}\emptyset$
15 **if** $e_{min} \geq \lambda_{max}$ **then**
16 break;
17 $\mathcal{Q}' \leftarrow \{q_i \in e.\mathcal{Q} | \delta_{min}(q_i, e) \leq q_i.\lambda \wedge q_i.\mathcal{T} \in e.\mathcal{T}\}$;
18 **if** $\mathcal{Q}'$ is $\emptyset$ **then**
19 continue the while-loop;
20 **for** each $t \in e.\mathcal{T}$ **do**;
21 **if** the object list of e' of t is not empty **then**
22 split the objects into child entries e';
23 **for** each child entries e' of e **do**
24 **if** $|\mathcal{Q}'.\mathcal{T} \cap e'.\mathcal{T}| \geq \mathcal{Q}.\mu$ and $\delta_{min}(\mathcal{Q}', e') < \mathcal{Q}'.\lambda$ **then**
25 $\mathcal{Q}^* \leftarrow \{q_i \in \mathcal{Q}' | \delta_{min}(q_i, e') \leq q_i.\lambda \wedge q_i.\mathcal{T} \in e'.\mathcal{T}\}$;
26 **if** $\mathcal{Q}^*$ is not $\emptyset$ then
27 **if** $e'.level \geq w'$ **then**
28 Node-process($\mathcal{Q}^*$,e',$\mathcal{R}$);
29 **else**
30 update e' by $\mathcal{Q}^*$;
31 Push e' into $\mathcal{H}$;
32 **return** $\mathcal{R}$

Algorithm 5 illustrates the details of the node processing. More specifically, for each linear quadtree in the selected query, we first check whether current entry e' is the leaf entry of current linear quadtree in Line 4. If it is not a leaf node of current linear quadtree, we move to the next linear quadtree. Otherwise, we increase the counter value by 1. If the corresponding object list is empty, we load the objects from the corresponding linear quadtree and update the object list of e'

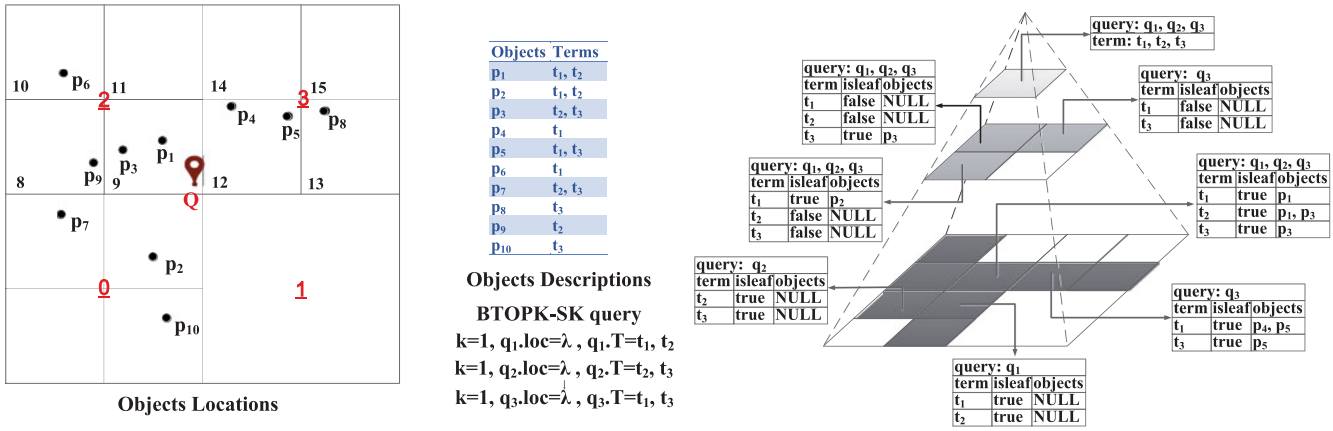


Fig. 7. BTOPK-SK data and virtual quadtree example.

in Line 7. Next, if the counter does not equal the terms number of the selected query, we update the current entry e' by selected query in Line 15. Otherwise, we first intersect the object lists from different terms of selected query in Line 9. Then, for each object in the intersect list, we calculate the distance between the object o and selected query in Line 11, and update $\mathcal{R}_i$ and λ_{max} in Lines 12-13 if the distance is smaller than the maximum distance of selected query.

Algorithm 5. Node-Process($\mathcal{Q}^*$, e', $\mathcal{R}$)

Input: $\mathcal{Q}^*$: the *spatio-textual* queries satisfied term constraint, e' : current node, $\mathcal{R}$: top-k results of each subquery for $\mathcal{Q}$
Output: $\mathcal{R}$: top-k results of each subquery for $\mathcal{Q}$

1 **for each** $q_i \in \mathcal{Q}^*$ **do**
2 $hit \leftarrow 0$;
3 **for each** $t \in q.\mathcal{T}$ **do**
4 **if** e' is a leaf node of LQ_t **then**
5 $hit := hit + 1$;
6 **if** the object list of e' of LQ_t is empty **then**
7 load the objects from LQ_t and update e';
8 **if** $hit = |q.\mathcal{T}|$ **then**
9 $objs \leftarrow$ intersects the objects of related terms in e';
10 **for each** $o \in objs$ **do**
11 **if** $\delta_{min}(q, o) < q.\lambda$ **then**
12 update $\mathcal{R}_i$ by $(o, \delta_{min}(q, o))$;
13 update λ_{max};
14 **else**
15 update e' by q_i;
16 **return** $\mathcal{R}$

Example 2. Consider a query group $\mathcal{Q} = \{q_1, q_2, q_3\}$ in (Fig. 7), where $q_1 = \langle \lambda, \{t_1, t_2\} \rangle$, $q_2 = \langle \lambda, \{t_2, t_3\} \rangle$, $q_3 = \langle \lambda, \{t_1, t_3\} \rangle$, $w' = 1$, $w = 2$, and we want to find the top-1 object. Hence, $\mathcal{Q}.\mathcal{T} = t_1, t_2, t_3$, $\mathcal{Q}_\mu = 2$, and all subqueries have the same location λ.

The algorithm maintains a heap $\mathcal{H}$ of size 1 for each query to continuously keep the current result of each query. First, after creating the root node e of the virtual quadtree, we update e by the query group and the root information from related linear quadtrees as what is displayed in (Fig. 7). Then BTOPK-SK first visits the root node e, and checks its children details. Since $|C_{1,0}.t| = 3$ ($\geq \mathcal{Q}.\mu$; $C_{1,0}$ contains t_1, t_2, t_3), $|C_{1,2}.t| = 3$ ($\geq \mathcal{Q}.\mu$; $C_{1,2}$ contains t_1, t_2, t_3), $|C_{1,3}.t| = 2$ ($\geq \mathcal{Q}.\mu$; $C_{1,3}$ contains t_1, t_3), these three cells are updated by influential queries and inserted into the minimum heap according to values, i.e., the minimum distances between current cell and the influential queries. Meanwhile, because the flag of t_3 of cell $C_{1,2}$ and the flag of t_1 of cell $C_{1,3}$ have been set *true* and the corresponding object lists are empty. Thus, We should load the objects of $C_{1,2}$ from LQ_3 and objects of $C_{1,3}$ from LQ_1 respectively and insert them into the corresponding object list as what is showed in (Fig. 7).

Since the value of $C_{1,2}$ is smaller than that of $C_{1,0}$ and $C_{1,3}$, $C_{1,2}$ is dequeued first. Because the object list of t_3 in current cell is not empty, we split objects in the object list into its child cells. Specifically, object p_3 assigned to t_3 of cell $C_{2,9}$, and the corresponding flag is set *true*. Since $|C_{2,9}.t| = 3$ ($\geq \mathcal{Q}.\mu$; $C_{2,9}$ contains t_1, t_2, t_3), $|C_{2,8}.t| = 1$ ($\leq \mathcal{Q}.\mu$; $C_{2,8}$ contains t_1), and $|C_{2,10}.t| = 1$ ($\leq \mathcal{Q}.\mu$; $C_{2,10}$ contains t_2), $C_{2,9}$ is inserted into the minimum heap, while $C_{2,8}$ and $C_{2,10}$ are pruned. Then, there are three cells in the heap: $|C_{2,9}|$, $|C_{1,3}|$, $|C_{1,0}|$. $C_{2,9}$ is dequeued because of the smallest distance. Owing to all terms in $C_{2,9}$ arrives leaf node, for each qualified query in current cell, we open the corresponding nodes in the related IL-Quadtrees, load the objects, and update the cell. Then the top-1 candidate for query q_1 is found, i.e., p_1; and the top-1 candidate for query q_2 is also found, i.e., p_3.

Next, $|C_{1,3}|$ is dequeued. Since only the term number of $|C_{2,12}|$ is larger than $\mathcal{Q}.\mu$, we insert it into the heap. Then $|C_{2,12}|$ is dequeued. Since all terms in $|C_{2,12}|$ also arrive leaf level, we load the objects from corresponding nodes of the related IL-Quadtrees similarly, and update current entry. Then the top-1 candidate for query q_3 is found, i.e., p_5. Then, $|C_{1,0}|$ is dequeued. Similar as $C_{1,2}$, we split the objects in the current cell to the corresponding terms of its child cells, i.e., the object p_2 allocated to t_2 of $C_{2,3}$. Because $|C_{2,3}.t| = 2$ ($\geq \mathcal{Q}.\mu$; $C_{2,3}$ contains t_1, t_2) and $dist(q_1, C_{2,3}) \leq dist(q_1, p_1)$, so $|C_{2,3}.t|$ inserts into the heap. However, $C_{2,2}$, which corresponds to q_2, is pruned, because $dist(q_2, C_{2,2}) \geq dist(q_2, p_3)$. We continue the dequeue and enqueue process, until the queue is empty or the top-1 result of all queries not updated.

7 Performance Evaluation

7.1 Experiment Setup

We present results of a comprehensive performance study to evaluate the efficiency and scalability of the proposed

TABLE 2
Dataset Details

Property	GN	EURO	WEB	TW
# of objects (millions)	2.2	0.18	2.24	13.35
vocabulary size (thousands)	208	68	2,898	6,890
avg. # of keywords per obj.	6.75	7	429	10.05

techniques in the paper. Especially, the following algorithms are evaluated for the TOPK-SK query.[7]

- *ILQ*. IL-Quadtree based TOPK-SK algorithm proposed in Section 4.
- *IVR*. The inverted R-tree based TOPK-SK algorithm [1].
- *IR*^{2}. The enhanced IR^2-tree technique [2] based TOPK-SK algorithm.
- *KR**. TOPK-SK algorithm developed based on the KR*-tree [8].
- *WIBR*. The WIBR-tree technique is proposed in [4].
- *CDIR*. The CDIR-tree technique is proposed in [9].
- *S2I*. The S2I[8] technique is proposed in [7].
- *SFCQ*. The SFCQ[9] technique is proposed in [6].
- *EILQ*. Enhanced IL-Quadtree by object partition technique proposed in Section 5.

To evaluate the efficiency of batch algorithm, the following algorithms are evaluated for the BTOPK-SK query.

- *I-ILQ*. The IL-Quadtree based TOPK-SK algorithm proposed in Section 4.
- *I-WIBR*. The WIBR-tree[10] technique is proposed in [4].
- *SG-WIBR*. The group WIBR-tree technique considers spatial partition proposed in [4].
- *QG-WIBR*. The group WIBR-tree technique combines the technique proposed in Section 6.2.
- *SG-ILQ*. The group IL-Quadtree technique considers spatial partition and proposed in Section 6.1.
- *QG-ILQ*. The group IL-Quadtree technique combines the technique proposed in Section 6.2.
- *QG-EILQ*. The group EILQ technique combines the technique proposed in Section 6.2.

7.1.1 Datasets

Performances 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[11] in which each object is associated with a geographic location and a short text description. *GN* serves as the default dataset in the experiments. *EURO*[12] is a real dataset that contains points of interest (e.g., park, hospital, supermarket) in Europe. The dataset *WEB* is a synthetic dataset generated from two real datasets, namely California[13] and WEBSPAM-UK2007.[14] We assign each document to a geographical location from the California road network. Similarly, the dataset *TW* generated from 13 million real tweets in [23] from May 2012 to August 2012. We combine the tweets without locations with a real spatial dataset that models the road network of USA.[15] In the experiments, the locations of all the dataset are scaled to the two-dimensional space $[0, 10000]^2$. Table 2 summaries the important statistics of four datasets.

7.1.2 Experiment Settings

All indices and algorithms[16] were implemented in Java. Two servers were used for evaluation, one is equipped with 2x 3.33 GHz Intel Xeon X5680 (6 Core), 48 GB RAM, and a 2TB SATA disk, and the other one is equipped with 2x 3.1 GHz Intel Xeon E5-2687W (8 Cores), 128 GB RAM, and a 1TB SATA disk. For ensuring a comparable evaluation results, the same server is used for conducting experiments on the same dataset. The Java Virtual Machine Heap is set to 8 GB. All index structures are disk resident, and the page size is fixed to 8,192 bytes. For the page size of R-tree,[17] we follow the setting in [22]. We evaluates both the response time and the number of simulated I/O.

7.1.3 Index Settings

For a fair comparison, we tune the important parameters of the algorithms for their best performances.

For CDIR-tree, we vary the parameter β from 0 to 1 and find that the it performs the best on all datasets at $\beta = 0.9$. The other parameter is the number of clusters k. According to the experimental results, we set k to 20, 10, 30, and 40 on GN, $EURO$, WEB, and TW, respectively.

For the IR^2-tree, the signature length l_s affects performance. Hence, we empirically find that the IR2-tree performs the best at l_s = 10,000 on GN, l_s = 7,000 on $EURO$, at l_s = 12,000 on WEB, and at l_s = 20,000 on TW.

Before constructing the WIBR-tree, we need to iteratively partition all objects in a dataset into two groups using top w frequent keywords. According to our experimental result, it is of little help in query performance when w exceeds 300, 200, 600, and 1,500 on GN, $EURO$, WEB, and TW, respectively.

For ILQ, we need to set an appropriate minimal depth value w' and a suitable split threshold c for the datasets. Based on the results of additional experiments, for w', we set 8 for GN and $EURO$, 6 for WEB and TW; for c, we set 64 for GN and $EURO$, and 128 for WEB and TW.

For EILQ, the number of cluster for each cell is another factor we need to consider. We find that for EILQ, 16

7. Appendix E, available online, rewrites general boolean expressions into Disjunctive Normal Form (DNF) [21], and evaluates the existing techniques on general Boolean queries.

8. Note that S2I technique [7] can immediately support TOPK-SK query by setting the preference parameter α to 1 (just consider the spatial proximity in ranking) and adding a judgment for each object retrieved to check whether it contains all the query keywords.

9. The code of SFCQ from [22]. We extend it to process top k spatial keyword search. According to [6], we assume all locations have been loaded into memory first.

10. We only consider WIBR-tree owing to all the other algorithms and their varieties are dominated by WIBR-tree based algorithms in the experiment of [4].

11. http://geonames.usgs.gov.

12. http://www.pocketgpsworld.com.

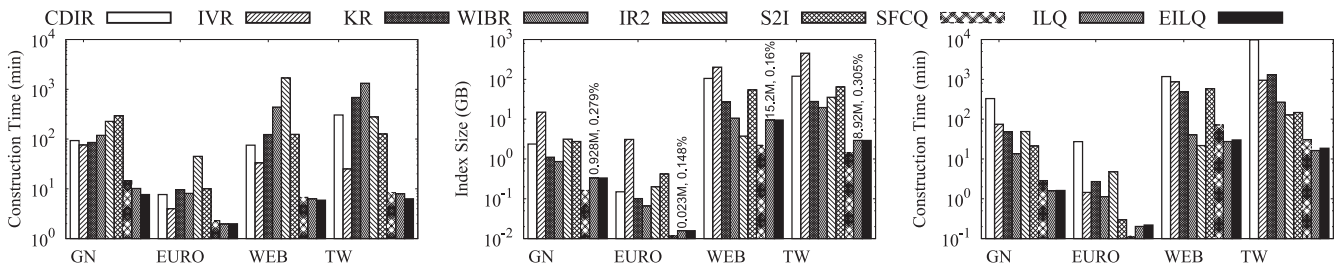
13. http://www.rtreeportal.org.

14. http://barcelona.research.yahoo.net/webspam/datasets/uk2007.

15. 5http://www.dis.uniroma1.it/challenge9/download.shtml.

16. The code of S2I is from its inventors.All the other codes from the author of [22].

17. We use 4K for IR^2, and 32K for the other R-tree based indices.



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

Fig. 8. Performance over various datasets.

clusters for *GN*, *EURO* and *WEB*, and 12 clusters for *TW* render the best performance.

7.2 Evaluating TOPK-SK Query

7.2.1 Workload

A workload for the TOPK-SK query consists of 300 queries, and the average query response time and the average number of simulated I/O are employed to evaluate the performance of the algorithms. We first randomly pick a object in the dataset and regard the location of the object as the query location. Then, we randomly select another object, and choose a specified number of keywords from the object based on the likelihood of the keyword. 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 and10 respectively.

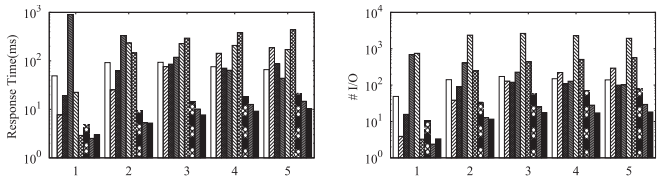
7.2.2 Evaluation on Different Datasets

We investigate the query response time, index construction time and index size of 8 algorithms against four datasets *GN*, *EURO*, *WEB* and *TW*, where other parameters are set to default values. In Fig. 8a, ILQ and EILQ demonstrate superior performance in comparison with other algorithms. Fig. 8b depicts the index sizes occupied by the algorithms. Particularly, SFCQ has the smallest index size among the algorithms, because the location information of each object is only stored once in SFCQ. On the other hand, IVR has the largest index size due to the extra duplicate information kept. The construction time of the index structures is depicted in Fig. 8c. More specifically, CDIR has the longest construction time in all datasets, owing to the complexity of the construction of DIR-tree. Withal, for datasets *GN*, *WEB* and *TW*, ILQ has the shortest construction time; for dataset *EURO*, SFCQ achieves the best performance.

7.2.3 Effect of the Number of Query Keywords (l)

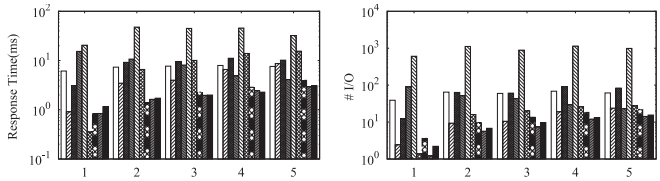
Figs. 9, 10, 11, 12 display the effect of the number query keywords on dataset *GN*, *EURO*, *WEB* and *TW*.

- *SFCQ*, *ILQ* and *EILQ*. EILQ performs the best on datasets *GN*, *WEB* and *TW* when the number of query keywords is larger than 1. Compared with EILQ, ILQ shows better performance in dataset *EURO*, because it doesn't have the supernumerary IO caused by extra clusters information and the size of dataset *EURO* is far less than that of other datasets. As expected, the performance of SFCQ, ILQ and EILQ degrades slightly when l grows due to effectiveness of the space partition based signature technique. Although SFCQ has similar response time as ILQ and EILQ, the I/O cost of SFCQ is always larger than that of ILQ and EILQ.
- *IVR* and *S2I*. The spatial-first R-tree based indexes IVR and S2I, which loosely combine the R-tree and the inverted file, perform well when l is 1, and their performances deteriorate as the number of keywords increases.
- *WIBR* and *CDIR*. The trend of WIBR is quite different to other algorithms, where the performance is significantly enhanced when the number of l grows. This is because the construction of WIBR-tree relies on the keyword partition and hence can minimize the number of average keyword size of each node. However, the performance of WIBR is disappointing on all datasets when l is small. The reason is that the construction of WIBR may impair the spatial closeness of the objects within the same node of WIBR-tree. Compared with WIBR, the search region of CDIR is smaller than that of WIBR on account of the smaller average MBR area. Thus, the performance of CDIR is much better than WIBR when l is small.
- *KR** and *IR*^{2}. It is interesting that when l grows, the performance of IR^2 degrades first and then improve. The former degradation caused by the growth of



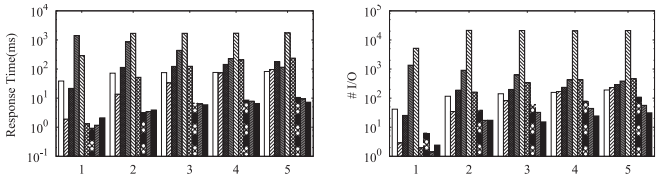
(a) Response time, *GN* (b) I/O, *GN*

Fig. 9. Varying # l on *GN*.



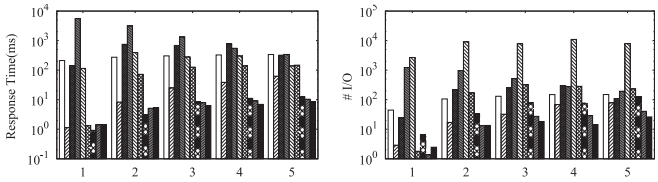
(a) Response time, *EURO* (b) I/O, *EURO*

Fig. 10. Varying # l on *EURO*.



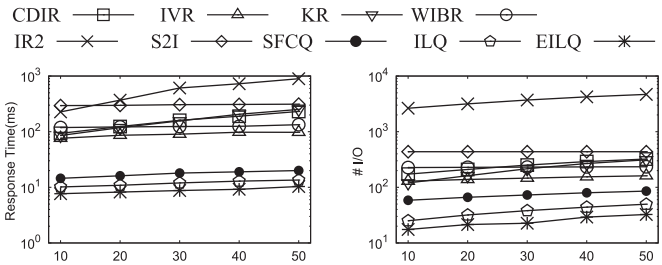
(a) Response time, *WEB* (b) I/O, *WEB*

Fig. 11. Varying # l on *WEB*.



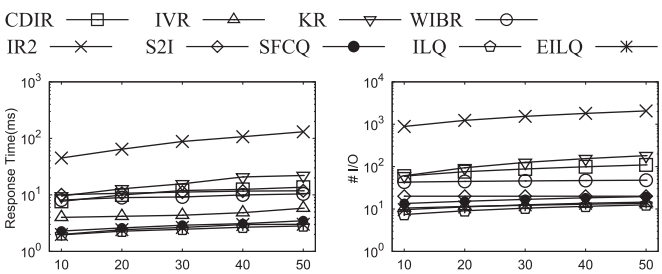
(a) Response time, *TW* (b) I/O, *TW*

Fig. 12. Varying # l on *TW*.



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

Fig. 13. Varying # k on *GN*.

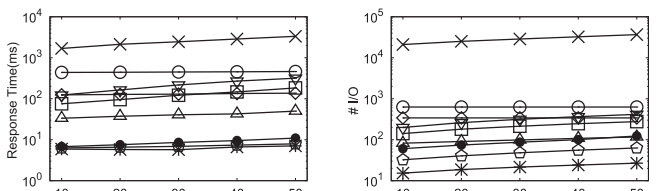


(a) Response time, *EURO* (b) I/O, *EURO*

Fig. 14. Varying # k on *EURO*.

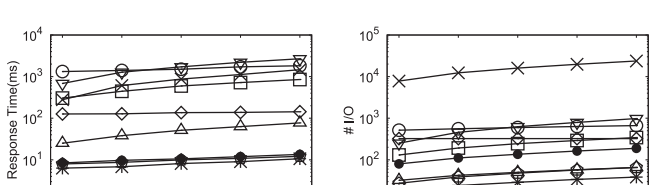
search region, the latter improvement result from the stronger pruning capability of signature. As expected, the performance of KR* deteriorates when l rises.

As we can see, all indices, except WIBR and IR^2, display an increasing tendency on all datasets in terms of both runtime and I/O cost as the number of keywords increases. For the text-first indices, which include IVR and S2I, they need to access more posting lists of words as the number of query keywords increases, and thus would need more I/Os. For the space-first and R-tree based indices, including CDIR and KR*, as the number of keyword increases, the possibility for a tree node to contain all the query keywords would be decreased, and the distance between the query results and the query point would be increased. Thus, such indices might check more nodes. Similarly, the space-first and Quadtree based indices, such as SFCQ, ILQ and EILQ, might inspect more cells as well.



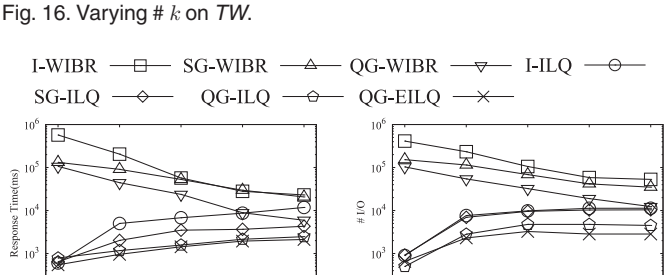
(a) Response time, *WEB* (b) I/O, *WEB*

Fig. 15. Varying # k on *WEB*.



(a) Response time, *TW* (b) I/O, *TW*

Fig. 16. Varying # k on *TW*.



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

Fig. 17. Varying # l on *GN*.

7.2.4 Effect of the Number of Results (k)

Figs. 13, 14, 15, 16 show results of this experiment in which we investigate the effect of k by varying the value k from 10 to 50. As expected, both the response time and the I/O cost of each index increase with an increasing value of k. A larger value of k leads to a larger search region in query processing, incurring more accesses to R-tree nodes or Quadtree cells.

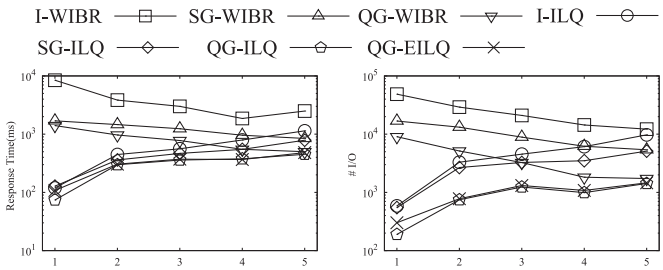
EILQ consistently performs the best on *GN*, *WEB*, and *TW*, while ILQ performs the best on *EURO*. The performance of S2I and WIBR is less affected as we increase the value of k, compared with other indices.

7.3 Evaluating BTOPK-SK Query

In this section, we evaluate the performance of batch top k spatial keyword query where the number of queries (h) varies from 500 to 2,500, where 500 is the default value.

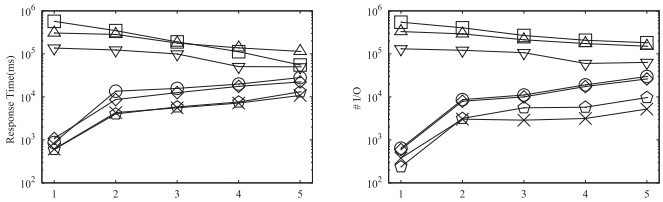
7.3.1 Effect of the Number of Query Keywords (l)

Figs. 17, 18, 19, 20 show the total I/O cost and the response time when the number of query keywords varies from 1 to 5. Since the group based algorithms process queries simultaneously, they exploit opportunities to share disk accesses and the computation among the queries. Note that such shared disk pages are only visited once. Therefore, compared with iterate based algorithms, the group based algorithms demonstrate superior performance in terms of response time and I/O cost. The



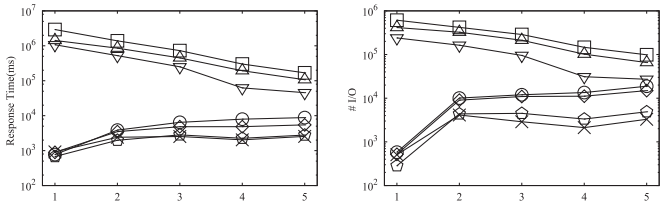
(a) Response time, *EURO* (b) I/O, *EURO*

Fig. 18. Varying # l on *EURO*.



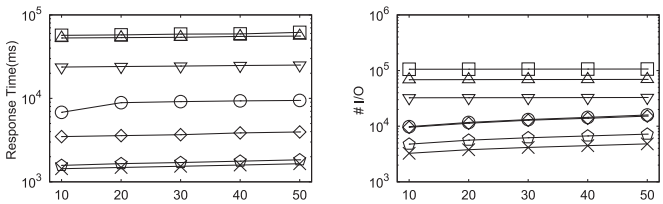
(a) Response time, *WEB* (b) I/O, *WEB*

Fig. 19. Varying # l on *WEB*.



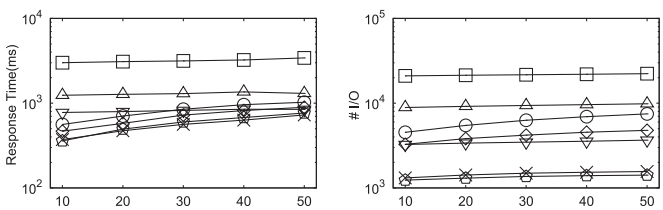
(a) Response time, *TW* (b) I/O, *TW*

Fig. 20. Varying # l on *TW*.



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

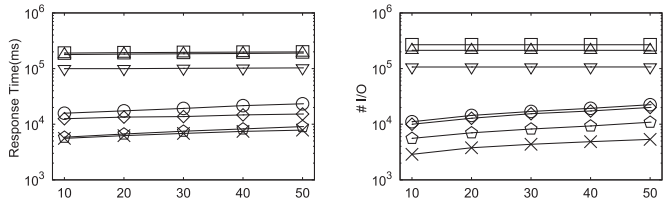
Fig. 21. Varying # k on *GN*.



(a) Response time, *EURO* (b) I/O, *EURO*

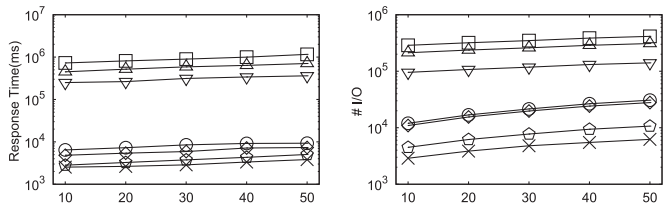
Fig. 22. Varying # k on *EURO*.

performance of QG-ILQ significantly outperforms that of SG-ILQ in all datasets due to proper query partition strategy. Similarly, the performance of QG-WIBR is also much better than that of SG-WIBR as well. QG-EILQ performs the best on datasets *GN*, *WEB* and *TW* when the number of query keywords is larger than 1. Compared with QG-EILQ, QG-ILQ demonstrates superior performance in dataset *EURO*.



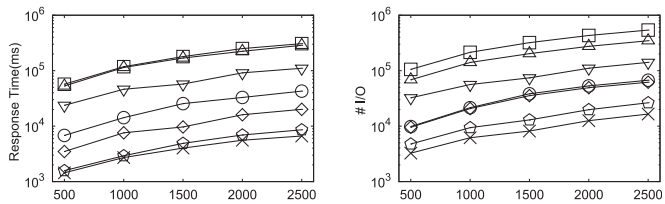
(a) Response time, *WEB* (b) I/O, *WEB*

Fig. 23. Varying # k on *WEB*.



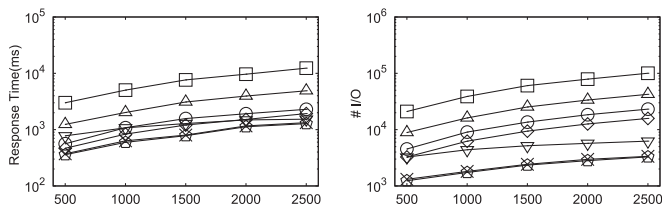
(a) Response time, *TW* (b) I/O, *TW*

Fig. 24. Varying # k on *TW*.



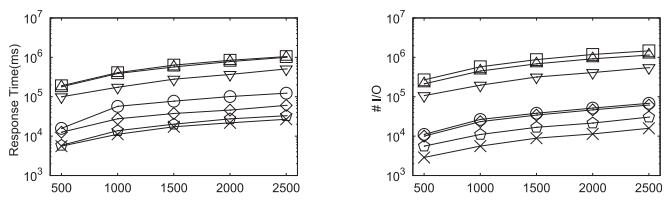
(a) Response time, *GN* (b) I/O, *GN*

Fig. 25. Varying # h on *GN*.



(a) Response time, *EURO* (b) I/O, *EURO*

Fig. 26. Varying # h on *EURO*.

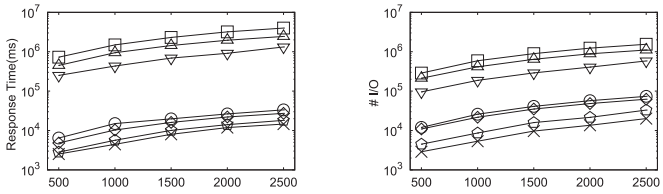


(a) Response time, *WEB* (b) I/O, *WEB*

Fig. 27. Varying # h on *WEB*.

7.3.2 Effect of the Number of Results (k)

Figs. 21, 22, 23, 24 show results of this experiment in which we investigate the effect of k by varying the value k from 10 to 50. As expected, both the response time and the I/O cost of each index increase with an increasing value of k. A larger value of k leads to a larger search region in query processing, incurring more accesses to R-tree nodes or Quadtree cells. In particular, QG-EILQ consistently has the best performance on *GN*, *WEB*, and *TW*, while QG-ILQ performs the best on *EURO*. The performance of all WIBR-tree based algorithms is less affected as we increase the value of k, compared with



(a) Response time, *TW* (b) I/O, *TW*

Fig. 28. Varying # h on *TW*.

that of IL-Quadtree based algorithms. We observe that the performance of IL-Quadtree based algorithms is at least one magnitude better than the corresponding WIBR-tree based algorithms in all datasets except *EURO*.

7.3.3 Effect of the Number of Queries (h)

Figs. 25, 26, 27, 28 depict the total I/O cost and response time on four datasets by varying the number of queries (h) from 500 to 2,500. As expected, when the number of queries increases, both the response time and the I/O cost of each index degrade since more R-tree nodes or Quadtree cells will be accessed. It is observed that the performance of IL-Quadtree based algorithms is still significantly superior to that of WIBR-tree based algorithms.

8 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. We further propose a partition based method to enhance the effectiveness of the signature of linear quadtree. To facilitate a large amount of spatial keyword queries, we propose a BTOPK-SK algorithm as well as a query group algorithm to enhance the performance of the system. Our comprehensive experiments convincingly demonstrate the efficiency of our techniques.

Acknowledgments

Ying Zhang is supported by ARC DE140100679 and DP130103245. Wenjie Zhang is supported by ARC DP150103071 and DP150102728. Xuemin Lin is supported by NSFC61232006, ARC DP150102728, and DP140103578.

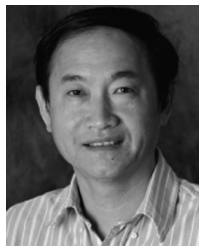
References

[1] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma, "Hybrid index structures for location-based web search," in *Proc. 14th ACM Int. Conf. Inf. Knowl. Manage.*, 2005, pp. 155–162.
[2] I. D. Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *Proc. IEEE 24th Int. Conf. Data Eng.*, 2008, pp. 656–665.
[3] S. Ding, J. Attenberg, R. A. Baeza-Yates, and T. Suel, "Batch query processing for web search engines," in *Proc. 4th Int. Conf. Web Search Web Data Mining*, Feb. 9–12, 2011, pp. 137–146.
[4] D. Wu, M. L. Yiu, G. Cong, and C. S. Jensen, "Joint top k spatial keyword query processing," *IEEE Tran. Knowl. Data Eng.*, vol. 24, no. 10, pp. 1889–1903, Oct. 2011.
[5] G. R. Hjaltason and H. Samet, "Distance browsing in spatial databases," *ACM Trans. Database Syst.*, vol. 24, no. 2, pp. 265–318, 1999.
[6] M. Christoforaki, J. He, C. Dimopoulos, A. Markowetz, and T. Suel, "Text vs. space: Efficient geo-search query processing," in *Proc. 20th ACM Int. Conf. Inf. Knowl. Manage.*, 2011, pp. 423–432.
[7] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg, "Efficient processing of top-k spatial keyword queries," in *Proc. 12th Int. Conf. Adv. Spatial Temporal Databases*, 2011, pp. 205–222.
[8] R. Hariharan, B. Hore, C. Li, and S. Mehrotra, "Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems," in *Proc. 19th Int. Conf. Sci. Statist. Database Manage.*, 2007, p. 16.
[9] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *Proc. VLDB Endowment*, vol. 2, no. 1, pp. 337–348, 2009.
[10] D. Wu, G. Cong, and C. S. Jensen, "A framework for efficient spatial web object retrieval," *VLDB J.*, vol. 21, pp. 797–822, 2012.
[11] A. Cary, O. Wolfson, and N. Rishe, "Efficient and scalable method for processing top-k spatial Boolean queries," in *Proc. Int. Conf. Sci. Statist. Database Manage.*, 2010, pp. 87–95.
[12] 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, pp. 585–599, Apr. 2011.
[13] D. Zhang, K.-L. Tan, and A. K. H. Tung, "Scalable top-k spatial keyword search," in *Proc. Int. Conf. Extending Database Technol.: Adv. Database Technol.*, 2013, pp. 359–370.
[14] G. Li, J. Feng, and J. Xu, "Desks: Direction-aware spatial keyword search," in *Proc. IEEE 28th Int. Conf. Data Eng.*, 2012, pp. 474–485.
[15] S. B. Roy and K. Chakrabarti, "Location-aware type ahead search on spatial databases: Semantics and efficiency," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 361–372.
[16] J. B. Rocha-Junior and K. Nørvåg, "Top-k spatial keyword queries on road networks," in *Proc. 15th Int. Conf. Extending Database Technol.*, 2012, pp. 168–179.
[17] C. Zhang, Y. Zhang, W. Zhang, X. Lin, M. A. Cheema, and X. Wang, "Diversified spatial keyword search on road networks," in *Proc. Int. Conf. Extending Database Technol.*, 2014, pp. 367–378.
[18] I. Gargantini, "An effective way to represent quadtrees," *Commun. ACM*, vol. 25, no. 12, pp. 905–910, 1982.
[19] G. M. Morton, "A computer oriented geodetic data base and a new technique in file sequencing," IBM Ltd, Ottawa, Canada, Tech. Rep., 1966.
[20] C. Faloutsos, "Multiattribute hashing using gray codes," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1986, pp. 227–238.
[21] H. B. Enderton, *A Mathematical Introduction to Logic*. New York, NY, USA: Academic Press, 1972.
[22] L. Chen, G. Cong, C. S. Jensen, and D. Wu, "Spatial keyword query processing: An experimental evaluation," *Proc. VLDB Endowment*, vol. 6, no. 3, pp. 217–228, 2013.
[23] G. Li, Y. Wang, T. Wang, and J. Feng, "Location-aware publish/subscribe," in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2013, pp. 802–810.

Chengyuan Zhang received the BSc degree in software engineering from SUN YAT-SEN University, China, and the MSc and PhD degrees in computer science from the University of New South Wales. He is a research assistant of computer science and engineering at The University of New South Wales (UNSW), Sydney, Australia. His research interests include query processing on database and query stream.

Ying Zhang received the BSc and MSc degrees in computer science from Peking University, and the PhD degree in computer science from the University of New South Wales. He is a senior lectuer and was an ARC DECRA research fellow from 2014 to 2016 at QCIS, the University of Technology, Sydney (UTS). His research interests include query processing on data stream, uncertain data and graphs. He was an Australian Research Council Australian Postdoctoral Fellowship (ARC APD) holder from 2010 to 2013.

Wenjie Zhang received the PhD degree in computer science and engineering in 2010 from the University of New South Wales. She is currently a lecturer and ARC DECRA (Australian Research Council Discovery Early Career Researcher Award) fellow in the School of Computer Science and Engineering, the University of New South Wales, Australia. Since 2008, she has published more than 20 papers in SIGMOD, VLDB, ICDE, *ACM Transactions on Database Systems*, *IEEE Transactions on Knowledge and Data Engineering*, and *VLDB Journal*. She received the Best (Student) Paper Award of NDBC 2006, APWebWAIM 2009, ADC 2010, and DASFAA 2012, and also coauthored one of the best papers in ICDE2010, ICDE 2012, and DASFAA 2012. In 2011, she received the Australian Research Council Discovery Early Career Researcher Award.



Xuemin Lin received the BSc degree in applied math from Fudan University in 1984. During 1984 to 1988, he studied for the PhD degree in applied math at Fudan University, and received the PhD degree in computer science from the University of Queensland in 1992. He is a professor in the School of Computer Science and Engineering, the University of New South Wales. He has been the head of database research group at UNSW since 2002. He is a concurrent professor in the School of Software, East China Normal University. Before joining UNSW, he held various academic positions at the University of Queensland and the University of Western Australia. He currently is an associate editor of *the ACM Transactions on Database Systems*. His current research interests include data streams, approximate query processing, spatial data analysis, and graph visualization. He is a fellow of the IEEE.