ARTICLE

Spatial keyword query processing: an experimental evaluation

LISI CHEN, Nanyang Technological University, Singapore City, Singapore

GAO CONG, Nanyang Technological University, Singapore City, Singapore

CHRISTIAN S. JENSEN, Aarhus University, Aarhus, Midtjylland, Denmark

DINGMING WU, Hong Kong Baptist University, Hong Kong, Hong Kong

Spatial Keyword Query Processing: An Experimental Evaluation

Lisi Chen[§] Gao Cong[§] Christian S. Jensen[†] Dingming Wu[♯]

[§] Nanyang Technological University, Singapore
[†] Aarhus University, Denmark
[♯] Hong Kong Baptist University
{lchen012@e.,gaocong@}ntu.edu.sg,csj@cs.au.dk,dmwu@comp.hkbu.edu.hk

ABSTRACT

Geo-textual indices play an important role in spatial keyword querying. The existing geo-textual indices have not been compared systematically under the same experimental framework. This makes it difficult to determine which indexing technique best supports specific functionality. We provide an all-around survey of 12 state-of-the-art geo-textual indices. We propose a benchmark that enables the comparison of the spatial keyword query performance. We also report on the findings obtained when applying the benchmark to the indices, thus uncovering new insights that may guide index selection as well as further research.

1. INTRODUCTION

With the proliferation of online objects with both an associated geo-location and a text description, the web is acquiring a spatial dimension. Specifically, web users and content are increasingly being geo-positioned and geo-coded. At the same time, textual descriptions of points of interest, e.g., cafes and tourist attractions, are increasingly becoming available on the web. This development calls for techniques that enable the indexing of data that contains both text descriptions and geo-locations in order to support the efficient processing of spatial keyword queries that take a geo-location and a set of keywords as arguments and return relevant content that matches the arguments [2].

Spatial keyword queries are being supported in real-life applications, such as Google Maps where points of interest can be retrieved, Foursquare where geo-tagged documents can be retrieved, and Twitter where tweets can be retrieved. Spatial keyword querying is also receiving increasing interest in the research community where a range of techniques have been proposed for efficiently processing spatial keyword queries. Three types of spatial keyword queries are receiving particular attention, namely the Boolean kNN query, the top-k kNN query, and the Boolean range query. We proceed to illustrate them with examples.

- **Boolean kNN Query:** "Retrieve the k objects nearest to the user's current location (represented by a point) such that each object's text description contains the keywords $tasty$, $pizza$, and $cappuccino$."

- **Top-k kNN Query:** "Retrieve the k objects with the highest ranking scores, measured as a combination of their distance to the query location (a point) and the relevance of their text description to the query keywords $tasty$, $pizza$, and $cappuccino$."

- **Boolean Range Query:** "Retrieve all objects whose text description contains the keywords $tasty$, $pizza$, and $cappuccino$ and whose location is within 10 km of the query location."

A number of geo-textual indices have been proposed. These indices usually combine a spatial index and a text index structure. The existing indices can be categorized according to the spatial index they utilize: (i) R-tree based indices [4, 7, 9, 12, 16, 18, 21, 26]; (ii) grid based indices [14, 19]; and (iii) space filling curve based indices [5, 6]. Using the text index employed, these indices can also be classified as inverted file based (e.g., [26]) and signature file based (e.g., [9]) indices. In addition, some of the indices (e.g., [26]) loosely combine a spatial and a text index while other indices integrate them tightly, resulting in hybrid index structures (e.g., [9]).

The papers that present the existing geo-textual indices often report on experimental studies that suggest that the proposed indices are competitive with baseline indices. This state of affairs makes it difficult to decide which index is the most suitable in a particular setting. Furthermore, each index often targets one particular type of query and is evaluated for that type of query only. It is also difficult to obtain an overview of the performance of the existing geo-textual indices for common types of spatial keyword queries. As a consequence, there is a clear need for a benchmark that offers in-depth insight into the performance of the existing geo-textual indices.

To the best of our knowledge, this paper contributes the first all-around evaluation of geo-textual indices. The evaluation features several representative geo-textual datasets, three types of common spatial-keyword queries, and a carefully designed evaluation procedure. Datasets with different scales, text sizes, and spatial distributions are employed to measure index storage requirements and evaluate query performance. Workloads of different types of queries are generated with a variety of settings for important parameters. The evaluation procedure enables a systematic study of query runtime and I/O performance in a broad variety of settings. By applying the evaluation procedure to existing geo-textual indices, the paper also offers new insight into the properties and relative merits of the geo-textual indices.

Proceedings of the VLDB Endowment, Vol. 6, No. 3

The rest of the paper is organized as follows: Section 2 describes the problem setting. Section 3 presents a categorization of the indices covered, and Section 4 surveys the indices. Section 5 details the evaluation procedure and experimental settings. Section 6 reports on the evaluation results. Finally, Section 7 concludes and summarizes the key findings.

2. PROBLEM DEFINITION

As does much of previous work, we assume that each geo-textual object has a point location with latitude, longitude, and a text description, and we consider three kinds of common spatial keyword queries on such objects.

Dataset Setting Let D be a geo-textual dataset. Each spatial object $o \in D$ is defined as a pair ($o.\rho$, $o.\psi$), where $o.\rho$ is a 2-dimensional geographical point location and $o.\psi$ is a text document.

We study the performance of processing three kinds of popular spatial keyword queries, namely the Boolean range query (BRQ), the Boolean kNN query (BkQ), and the top-k kNN query (TkQ).

Boolean range Query (BRQ) Given a BRQ $q = \langle \psi, r \rangle$, where $q.\psi$ is a set of keywords and $q.r$ is a spatial region, the result of q, $q(D)$, is a subset of D containing objects such that $\forall o \in q(D)(o.\rho \in q.r \wedge q.\psi \subseteq o.\psi)$. In other words, the BRQ retrieves objects containing all the query keywords and belonging to the query region. The result objects are not ranked. However, the BRQ can be easily modified to rank the result objects based on their text relevance, and it is easy to extend query processing algorithms to achieve this.

Boolean kNN Query (BkQ) A BkQ $q = \langle \psi, \rho, k \rangle$ takes three arguments, where $q.\psi$ is a set of keywords, $q.\rho$ is a spatial query point, and $q.k$ is the number of objects to retrieve. The result of a BkQ, $q(D)$, is a set of k objects, each of which covers all the keywords in $q.\psi$. Objects are ranked according to their distances to $q.\rho$. Formally, $\forall o \in q(D)((\nexists o' \in D \setminus q(D))(dist(o'.\rho, q.\rho) \leq dist(o.\rho, q.\rho)) \wedge q.\psi \subseteq o'.\psi)$.

Top-k kNN Query (TkQ) A TkQ $q = \langle \psi, \rho, k \rangle$ takes the same arguments as BkQ. It retrieves k objects ranked according to a score that takes into consideration spatial proximity and text relevance. Specifically, the ranking score of object o for a TkQ q is defined in the following equation:

$$ST(o, q) = \alpha \cdot SDist(o.\rho, q.\rho) + (1 - \alpha) \cdot TRel(o.\psi, q.\psi),$$

where $SDist(o.\rho, q.\rho)$ is the spatial proximity between $o.\rho$ and $q.\rho$, $TRel(o.\psi, q.\psi)$ is the text relevance between $o.\psi$ and $q.\psi$, and $\alpha \in [0, 1]$ is a query preference parameter that makes it possible to balance the spatial proximity and text relevance. The spatial proximity is defined as the normalized Euclidian distance: $SDist(o.\rho, q.\rho) = \frac{dist(o.\rho, q.\rho)}{dist_{max}}$,where $dist(o.\rho, q.\rho)$ is the Euclidian distance between o and q, and $dist_{max}$ is the maximum distance between any two objects in D. The text relevance $TRel(o.\psi, q.\psi)$ can be computed using an information retrieval model, such as the language model [7], cosine similarity [18], or BM25 [6], and is normalized to a scale similar to the proximity. We use the language model in this evaluation.

Other Query Types Zhang et al. [24, 25] consider a so-called m-closest keywords query and introduce an R-tree based structure called the bR*-tree. This query retrieves a set of objects, each of which contains m query keywords. The query minimizes the maximum distance between any two objects in the set.

Cao et al. [3] propose the *spatial group keyword query* with two instantiations: (1) find a group of objects that covers all the query keywords such that the sum of their distances to the query point is minimized; (2) find a group of objects that covers all the query keywords such that the sum of the maximum distance from an object in the group to the query point and the maximum distance between two objects in the group is minimized.

Lu et al. [17] propose the *reverse spatial and textual k nearest neighbor* query, which takes an object with a spatial point and a set of keywords as arguments. The query retrieves the objects that have the query object as one of their k most similar objects with regards to both spatial proximity and text relevancy.

Wu et al. [22] and Huang et al. [13] consider processing the *moving top-k spatial keyword* query, which takes as arguments a moving spatial point and a set of keywords. The query continuously finds k objects that best match a query with respect to spatial proximity and text relevancy.

Li et al. [15] propose the *direction-aware spatial keyword query*, which takes as arguments a spatial point, a direction, and a set of keywords. It finds k spatially nearest neighbors of the query such that they are in the query direction and contain all the query keywords.

SKIF [14] is also developed for a new type of query. Like the TkQ, the query in SKIF aims to find k objects with the highest weighted scores in terms of both spatial relevancy and text relevancy. SKIF assumes the location of each object to be a region rather than a point, and the spatial relevancy is expressed as the overlap between the query region and region of an object. The concept of text relevancy is the same as that of the TkQ.

3. INDEXING CLASSIFICATION

We classify the geo-textual indices according to three different aspects: the spatial indexing scheme used, the text index employed, and hybrid manner of the spatial index and the text index. The classification is summarized in Figure 2. The symbol "$\surd$" denotes that an index is originally designed for the corresponding type of query; the symbol "$\triangle$" means that the index can be employed to efficiently process a type of query with an extended query processing algorithm, which makes the index suitable for the type of query.

3.1 Spatial Indexing Scheme

Considering the spatial indexing scheme used, we classify the indices into three categories, namely R-tree based indices, grid based indices, and space filling curve based indices.

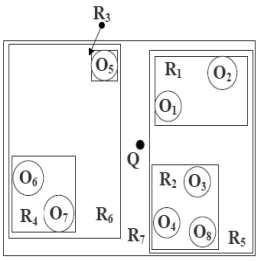
R-tree based This category of indices use the R-tree [11] or a variation (e.g., the R*-tree [1]). Most geo-textual indices belong to this category and use the inverted file for text indexing. In early work [26], the R-tree based indices loosely combine the R-tree and inverted files to organize the spatial and text data separately. In contrast, recent indices tightly combine the R-tree with a text index.

Grid based This category of indices combine a grid index with a text index (e.g., the inverted file). The grid indices divide space into a predefined number of equal-sized square or rectangular cells. The grid index and the text index can be organized [19] either separately or combined tightly [14].

Space filling curve based These indices combine inverted files with a space filing curve, and they include a Hilbert curve based index [5] and a Z-curve based index [6]. These indices are based on the property that points close to each other in the native space are also close to each other on the space filling curve.

3.2 Text Indexing Scheme

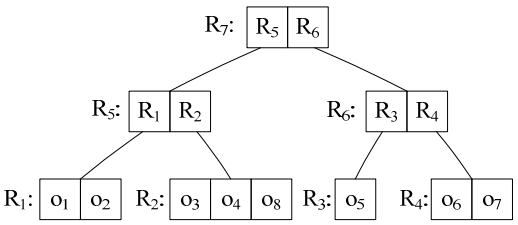
Inverted file Most geo-textual indices [6,7,10,12,14,16,18,19,26] use the inverted file for text indexing. An inverted file has a vocabulary of terms, and each term is associated with an inverted list.



(a) Objects and their bounding rectangles

Object	Terms and term frequencies
o_1	(Italian, 5)(restaurant, 5)(expensive, 2)
o_2	(coffee, 5)(restaurant, 5)(expensive, 1)
o_3	(Italian, 7)(pizza, 1)(expensive, 1)
o_4	(restaurant, 7)(pizza, 1)(expensive, 1)
o_5	(Italian, 4)(restaurant, 4)
o_6	(coffee, 4)(restaurant, 3)
o_7	(Italian, 1)(coffee, 1)(restaurant, 4)(pizza, 1)
o_8	(coffee, 3)(restaurant, 3)(expensive, 1)

(b) Text information of objects in (a)



(c) R-tree for objects in Fig. 1(a)

Figure 1: Example

Index	Abbr	Spatial part	Textual part	Combination scheme	BkQ	TkQ	BRQ
ST [19]	ST	Grid	inverted file	spatial-first			√
TS [19]	TS	Grid	inverted file	text-first			√
IF-R*-Tree [26]	IF-R*	R*-Tree	inverted file	text-first	△		√
R*-Tree-IF [26]	R*-IF	R*-Tree	inverted file	spatial-first		△	√
SF2I [5]	SF2I	SFC	inverted file	spatial-first			√
KR*-Tree [12]	KR*	R*-Tree	inverted file	tightly combined	△		√
IR^2-Tree [9]	IR^2	R-Tree	bitmaps	tightly combined	√		△
IR-Tree [7, 20]	IR	R-Tree	inverted file	tightly combined	△	√	△
IR-Tree [16]	IRLi	R-Tree	inverted file	tightly combined		√	
SKIF [14]	SKIF	Grid	inverted file	tightly combined			√
SKI [4]	SKI	R-Tree	bitmaps	spatial-first	√		
S2I [18]	S2I	R-Tree	inverted file	text-first	△	√	△
WIBR-Tree [21]	WIBR	R-Tree	inverted bitmaps	tightly combined	√		△
SFC-QUAD [6]	SFC-Quad	SFC	inverted file	tightly combined			√

Figure 2: Comparison of existing geo-textual indices

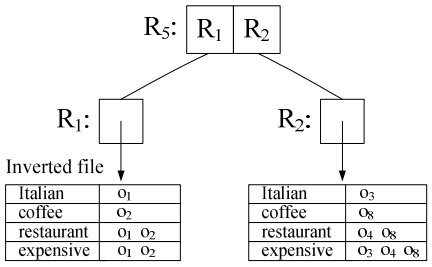


Figure 3: Inverted file under leaf node R_1 and R_2

Each inverted list comprises a sequence of postings, each of which normally contains the identifier of an object o whose description $o.\psi$ contains the term and the frequency of the term in $o.\psi$. The frequency information is not included in the indices that are developed to handle Boolean queries. In general, the postings in each inverted list are sorted by object ID. However, some geo-textual indices order the postings differently, such as ordering them by their orders in grid cells [14] or according to a space filling curve [6].

Bitmap Some R-tree based indices [4, 9, 21] use bitmaps [8] to index the text information in subtrees. Simply put, each bit in a bitmap represents the presence or absence of a term in a document. The IR^2-tree [9] augments each node of the R-tree with a signature file (bitmap) to capture the text information of the objects in the node's subtree. Another study [24] augments the nodes of the R-tree with bitmaps directly. Some geo-textual indices [21] use inverted bitmaps, in which each term corresponds to a bitmap and each bit in a bitmap captures whether a document contains the term.

3.3 Combination Scheme

The geo-textual indices combine spatial and text indexing. We categorize the indices according to how they combine the two, namely text-first loose combination, spatial-first loose combination, and tight combination.

A text-first loose combination index usually employs the inverted file as the top-level index and then arrange the postings in each inverted list in a spatial structure, which can be an R-tree, a grid or a spatial filling curve. In contrast, the top level of a spatial-first index is a spatial structure, and its leaf nodes (resp. grid cells) contain inverted files or bitmaps for the text information of objects contained in the nodes (resp. grid cells).

On the other hand, the tight combination index combines a spatial and a text index tightly such that both types of information can be used to prune the search space simultaneously during query processing. Two types of tight combinations have been used: One integrates a text summary into every node of a spatial index (e.g., [7]), and one integrates the spatial information into each inverted list (e.g., [6]).

4. GEO-TEXTUAL INDICES

We review 12 geo-textual indices that we will later evaluate empirically. Figure 1(a) and Figure 1(b) show the spatial and keyword distribution, respectively, for a set of spatial web objects, which is used as a running example. Figure 1(c) illustrates an R-tree on these objects.

4.1 R-Tree Based Indices

IF-R* and R*-IF The R-tree is arguably the dominant index for spatial queries, and the inverted file is the most efficient index for text information retrieval. Inverted file-R*-tree (IF-R*) and R*-tree-inverted file (R*-IF) [26] are two geo-textual indices that loosely combine the R*-tree and inverted file.

The R*-IF is a spatial-first index. An R*-tree is first built for indexing all objects in D without considering their text components. Next, for each leaf node of the R*-tree, an inverted file is created for indexing the text components of the objects contained in the leaf node. Note that the inverted file is not stored inside a leaf node. Figure 3 illustrates the inverted files for leaf nodes R_1 and R_2.

The IF-R*, a text-first geo-textual index, is the counterpart of R*-IF. For each distinct term t in D, a separate R*-tree is built for the objects in D containing term t. Figure 4 illustrates the structure of IF-R* (R*-tree for the term $expensive$).

Both of them are proposed for tackling the problem of retrieving web documents relevant to a keyword query within a pre-specified

spatial region, i.e., the BRQ. At query time, for the R*-IF, a set of leaf nodes that intersect with the query region is retrieved first. Then, objects whose documents contain query keywords are returned as the result. For the IF-R*, all the R*-trees corresponding to the query keywords need to be accessed.

It is shown [26] that the IF-R* outperforms the R*-IF for the BRQ. Hence we only evaluate the performance of the IF-R*. Additionally, we also employ the IF-R* for processing the BkQ using the extended algorithm [21]. However, there exists no sensible way to use the two indices for the TkQ.

KR*-Tree Hariharan et al. [12] proposed the KR*-tree (Keyword R*-tree). Each node of the KR*-tree is virtually augmented with the set of keywords that appear in the subtree rooted at the node. The nodes of the KR*-tree are organized into inverted lists as are the objects. This may help prune the tree nodes under which objects do not contain query keywords in query processing stage. Figure 5 shows the content of the KR*-tree list generated from the example dataset.

The KR*-tree is proposed for processing the BRQ. The KR*-tree based query processing algorithm first finds the set of nodes that contain the query keywords. The resulting set then serves as the candidate pool for subsequent search. We also use the KR*-tree to process the BkQ. However, we do not see a sensible way to apply this index structure for processing the TkQ.

IR^2-Tree Felipe et al. [9] proposed an index structure called IR^2-tree, which integrates signature file into each node of the R-tree. The signature file, in the form of bitmap, is stored for each node of the IR^2-tree, and thus the fanout of the tree is dependent on the length of signature file. The signature file of a node is the union of all signatures of its entries, each representing a child node, and it summarizes the presence of terms in the objects rooted at the node. Figure 7 shows the bitmap structure on the node R_6.

The IR^2-tree can be used for processing the BkQ and BRQ. However, since the signature files do not have the frequency information, it cannot be used to process the TkQ

Hybrid Spatial-Keyword Indexing (SKI) Cary et al. [4] proposed SKI, which uses the R-tree and bitmaps to store spatial and text information respectively. An extended R-tree is used to organize the spatial information. The parent node of a leaf node is called a super node in the extended R-tree. Each non-leaf node is augmented with a range of the ids of the super nodes under the non-leaf node. Each super node is associated with a bitmap version of inverted file. Specifically, each term has a bitmap whose bit corresponds to an object under the super node and is set to 1 if the object contains the term. The SKI shares certain similarity to the R*-IF. However, the SKI uses bitmap version of inverted file rather than a normal inverted file. To illustrate the structure of SKI, we give the term bitmaps on super node R_5 in Figure 6. SKI can be used for the BkQ and BRQ.

IR-tree index and its variants The IR-tree [7, 20] augments each node of the R-tree with a summary of the text content of the objects in the corresponding subtree. Specifically, each node contains a pointer to an inverted file that describes the objects in the subtree rooted at the node. The inverted file for a node X contains: 1) A vocabulary of all distinct terms in the text descriptions of the objects in the subtree rooted at X. 2) A set of posting lists, each of which relates to a term t. Each posting list is a sequence of pairs $\langle cp, wt_{cp,t}\rangle$, where cp is a child of X and $wt_{cp,t}$ is the upper bound text relevance score of objects in the subtree rooted at cp for term t. The IR-tree supports all the three types of queries, namely, BRQ, BkQ and TkQ.

Li et al. [16] presents an index structure, which is also called IR-tree. To distinguish it from the IR-tree in references [7,20], we refer to it 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. More specifically, the posting list for each term in the IRLi-tree corresponds to the concatenation of the posting lists of all the nodes of the IR-tree. The fashion that the IRLi-tree organizes the inverted file is similar to that of the KR*-tree, and the difference lies in that the KR*-tree does not store the weight of terms. Thus, the IRLi-tree will reduce to the KR*-tree if it is used to process BkQ and BRQ.

Several variants of the IR-tree exist, which optimize the IR-tree, including the DIR-tree, the CIR-tree, and the CDIR-tree. The DIR-tree [7, 20] takes both spatial and textual information into account during the tree construction by optimizing for a combination of minimizing the areas of MBRs and maximizing the text similarities between the objects of the enclosing rectangles. A parameter β is introduced to balance the weights on the two parts. The CIR-tree optimizes the IR-tree by grouping objects into a number of clusters based on their text descriptions. Compared with the IR-tree, the CIR-tree includes a summary of text content for each cluster in its posting lists. The CDIR-tree [7, 20] is a combination of the DIR-tree and the CIR-tree.

WIR-Tree WIR-tree [21] is also a variant of IR-tree. It aims at partitioning objects into multiple groups such that each group shares as few keywords as possible. To achieve this goal, the objects in D is first partitioned into two groups using the most frequent word w_1: one group whose objects contain w_1 and the other group whose objects do not. We then partition each of these two groups by the next frequent word w_2. The process is repeated iteratively until each partition contains a certain number of objects. After partitioning, each group of objects becomes the leaf node of the WIR-tree. Then the tree is constructed following the structure of the IR-tree. When used for processing Boolean queries, the WIR-tree [21] uses the inverted bitmap to replace the inverted file, which is denoted as the WIBR-tree, where a bitmap position corresponds to the relative position of an entry in its WIR-tree node. The length of a bitmap is equal to the fanout of a node. Like the IR-tree, the WIR-tree can handle all the three types of query.

Spatial Inverted Index (S2I) Based on the inverted file and R-tree, Rocha-Junior et al. [18] proposed an index called S2I that employs two different strategies for indexing frequent items and infrequent terms. Specifically, the S2I maps objects containing each frequent term to an aggregate R-tree (aR-tree). In the aR-tree, each node stores an aggregated value that captures the maximum impact (in terms of the text relevance score) of the term on the objects in the subtree rooted at the node. The aR-tree can be regarded as the IR-tree [7,20] for a single term. The S2I organizes the infrequent terms and the objects containing them by inverted file, and each posting list of a term is organized by blocks of a fixed size. The threshold to distinguish frequent terms from infrequent terms needs to be set empirically.

S2I is originally designed for TkQ, but it can be employed to support BkQ and BRQ.

4.2 Grid Based Spatial-Textual Indices

ST and TS Vail et al. [19] proposed two grid based spatial-textual indexing schemes, Spatial Primary Index (ST) and Text Primary Index (TS). They are the earliest grid based geo-textual indices. ST and TS can be classified as spatial-first and text-first loose combination, respectively, in terms of the combination scheme.

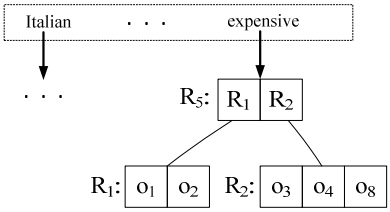


Figure 4: R-tree under the word $expensive$

Keywords	Tree nodes
Italian	$R_1, R_2, R_3, R_4, R_5, R_6$
coffee	R_1, R_2, R_4, R_5, R_6
restaurant	$R_1, R_2, R_3, R_4, R_5, R_6$
pizza	R_2, R_4, R_5, R_6
expensive	R_1, R_2, R_5

Figure 5: KR*-tree list

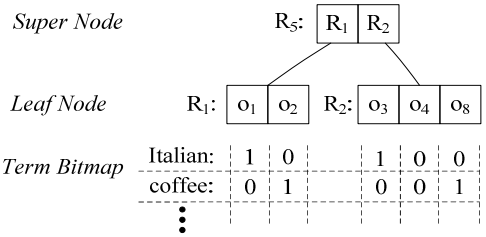


Figure 6: Term bitmaps for super node R_5

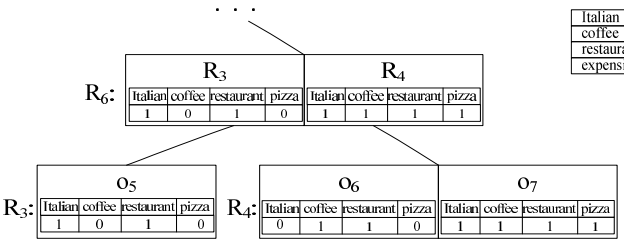


Figure 7: IR^2-tree with bitmaps

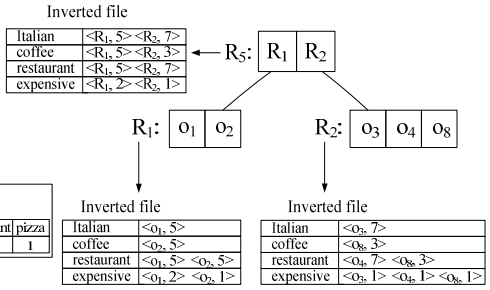


Figure 8: IR-tree and its inverted files

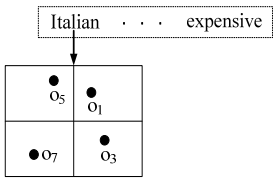


Figure 9: Grid structure under keyword $Italian$

Since TS outperforms ST consistently as shown in the work [19], we only evaluate TS in this paper. The structure of TS, which is illustrated in Figure 9, is a modified inverted index of which each posting list is associated with a grid cell based on its location.

TS is designed for the BRQ, and its query processing method is similar to that of the IF-R*. There exists no method to extend TS to efficiently handle BkQ and TkQ. A straightforward extension would not work better than using inverted file alone.

Spatial-Keyword Inverted File (SKIF) SKIF [14] employs an inverted-file-like structure to store both spatial and text information for objects so that spatial and textual parts of data can be handled simultaneously. SKIF assumes the location of each object as a region, rather than a single point, in the setting of this evaluation.

The text part of SKIF employs the inverted file, i.e., each term is associated with an inverted list. For spatial part, each distinct grid cell is also represented by an inverted list. Each inverted list comprises a list of postings, each consisting of an object ID and its spatial idf value, where the object overlaps with the grid cell and the idf is measured by the overlapping degree.

SKIF is developed for a type of query that is different from the three types of queries we evaluate. Since the query considered by SKIF bears resemblance to the BRQ, we modify its query processing algorithm to handle the BRQ. However, we do not see a sensible way to process the other two types of queries based on SKIF.

4.3 SFC-QUAD

Christoforaki et al. [6] proposed several hybrid indices that combine the space filling curve and inverted file. Among them, SFC-QUAD index is shown to perform the best according to the experimental results [6]. The SFC-QUAD index is essential an inverted file in which the docIDs and frequencies of objects are compressed using the OPT-PFD algorithm [23]. The docIDs in each inverted list are assigned and ordered based on their spatial positions on the Z-curve. SFC-QUAD also maintains a Quad-tree in memory so that the Z-curve order could be easily acquired simply by traversing the Quad-tree in a depth-first manner. When processing a query with a given spatial region, a set of object ID ranges that fall in the query region are acquired through traversing the Quad-tree. Then, these ranges are merged into a smaller number of ranges for reducing random disk I/O costs. Subsequently, the corresponding parts of inverted lists of query keywords within the s ranges of IDs are retrieved from disk through s disk sweeps. Finally, objects containing all the query keywords are found through document-at-a-time query processing technique along with forward skip optimization.

SFC-QUAD is designed for the BRQ. If it is extended for handling BkQ and TkQ, it will be reduced to normal inverted list since its unique indexing structure and query processing method cannot be used for processing BkQ and TkQ.

The earlier work [5] combines inverted file with the Hilbert curve in a similar way as the method [6] though the method [6] uses more optimization. The inverted lists are laid out along a Hilbert curve on disk. In the work [5], an object has a large region of spatial footprint or multiple locations. Similar to SFC-QUAD, the index [5] is not for BkQ and TkQ.

5. EVALUATION PLAN

The goal of the experimental study is to make an all-around evaluation on important aspects of the geo-textual indices covered in Section 4 and compare their performance. All indices and algorithms[1] were implemented in Java running the Windows 7 Ultimate OS. Two servers were used for evaluation, one is equipped with Intel(R) Xeon(TM) CPU X5650 @2.66GHz, 24GB RAM, and a 1TB SATA disk, and the other one is equipped with Intel(R) Xeon(TM) CPU E5620 @2.4GHz, 48GB RAM, and a 1.5TB 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 8GB.

5.1 Datasets

Our experiments were conducted on three datasets: a small real dataset EURO, a larger synthetic dataset WEBSPAM, and the

[1]The code of S2I is from its inventors. We extended it to process all the three types of queries. All code developed by us will be publicly available.

Table 1: Dataset properties

Property	EURO	WEBSPAM	TWITTER
Total number of objects	179,506	2,248,135	20,000,000
Total number of unique words	68,052	2,898,463	8,977,384
Average number of unique words	7	429	15
Total number of words	1,155,045	965,127,684	308,584,627

largest dataset TWITTER. The properties of the three datasets are listed in Table 1. Details of each dataset are stated as follows.

EURO: EURO[2] is a real dataset that contains points of interest (e.g., park, hospital, supermarket) in Europe. Each point of interest, which can be regarded as a spatial web object, contains a geographical location and a short description (name, categories, etc.).

WEBSPAM: WEBSPAM is a synthetic dataset generated from two real datasets, namely a spatial dataset modeling the roads in California[3] and web documents on WEBSPAM-UK2007[4] (which are generally longer than the descriptions of EURO objects). We assign each document to a geographical location from the California road network. To evaluate the effect of text length, 4 additional datasets are generated by ramdomly selecting part of the text for each object to guarantee the average number of words per object to be 100, 200, 300, and 400.

TWITTER: TWITTER is a dataset generated from 20 million real tweets in USA. A total of 0.5% of the tweets are geo-tagged. We combine the tweets without locations with a real spatial dataset that models the road network of USA.[5] To evaluate scalability, four additional datasets are generated by enlarging TWITTER from 40 to 100 million tweets while keeping the spatial and term distribution of the objects the same.

5.2 Performance Evaluation Procedure

Our benchmark evaluates both the time and space efficiency of each index. Three metrics are used: (i) elapsed time, (ii) the number of simulated I/Os, and (iii) the size of an index on disk. We use (i) and (ii) to evaluate the time efficiency of an index and (iii) to evaluate its space efficiency.

Note that multiple layers of cache (e.g., disk driver cache, operating system cache, application cache) exist between a Java application and the physical disk. Rather than measuring physical I/Os from the disk using Java, we report simulated I/O costs. If a node for the R-tree or B^+-tree is visited, the number of simulated I/Os is increased by 1, and if an inverted list or signature file is loaded into memory, the number of simulated I/Os is increased by the number of blocks (4K per block) used for storing the list or signature file.

For each index, we evaluate the following aspects.

A1 **The number of objects.** The number of objects in the dataset varies from 20M to 100M by enlarging the dataset TWITTER.
A2 **The number of query keywords.** We use AND semantics to connect the query keywords in BkQ and BRQ. The number of query keywords varies from 1 to 6.
A3 **Query region size.** All range queries are circle-shaped with their radius ranging from 1km to 20km.
A4 **Value of k.** For BkQ and TkQ, the value k in the top-k objects varies from 1 to 50.
A5 **Value of query preference parameter α.** For TkQ, parameter α allows users to set their preferences between location proximity and text relevance. We vary α from 0.1 to 0.9.
A6 **Buffer size.** When the data layout on disk with caching, we use an LRU buffer, and we vary the buffer size from 4MB to 16MB for EURO and from 0.5GB to 2GB for WEBSPAM.
A7 **R-tree page size.** For R-tree based indices, the page size is varied from 2K to 32K.
A8 **Space requirement.** For each index, we report its space requirement for storing the index file.
A9 **Text length.** We evaluate the performance with regard to the average number of words per object ranging from 100 to 400 on WEBSPAM.

[2]http://www.pocketgpsworld.com
[3]http://barcelona.research.yahoo.net/webspam/datasets/uk2007
[4]http://www.rtreeportal.org
[5]http://www.dis.uniroma1.it/challenge9/download.shtml

Table 2: Parameters and their settings

Parameter	Setting
Number of query keywords	1 2 **3** 4 5 6
Number of top-k results	1 5 **10** 20 50
Query preference ratio α	0.1 **0.3** 0.5 0.7 0.9
R-tree page size[6]	2K **4K** 8K 16K **32K**
Buffer size (MB)	**0(0)** 4(512) 8(1024) 12(1536) 16(2048)
Query region radius (km)	1 2 5 **10** 20

Table 2 summarizes the parameters, where the values in bold represent the default values used. For each query type, we evaluate these aspects where applicable. For A7 and A9, we only report results for BkQ and BRQ, respectively, since the results on the TkQ are qualitatively similar.

5.3 Query Generation

To make the queries resemble what users would likely use, we generate the query sets by the following steps. For BkQ and TkQ, we first randomly pick an object in the dataset and regard the location of the object as the query location. Then, we randomly choose a specified number of words from the object as the query keywords. Thus, each query will return at least one result. For BRQ, an object is picked randomly and then a bounding circular region with a specified radius is generated subsequently with the location of this object being the center of the region. The generation procedure for query keywords is the same as that for BkQ and TkQ.

For each round of experiment, we generate 6 query sets on each dataset, in which the number of query keywords is from 1 to 6. Each set consists of 300 queries.

For evaluating the query performance with different buffer sizes, we generate 500 and 20,000 queries for experiments on EURO and WEBSPAM, respectively, to warm up the LRU buffer.

Table 3: Distance from query point to kth result object

top-k	BkQ (kilometer)			TkQ (kilometer)		
	EURO	WEBSPAM	TWITTER	EURO	WEBSPAM	TWITTER
1	0	0	0	0	0	0
5	56	17	241	34	11	227
10	85	23	374	53	20	342
20	133	35	482	96	29	417
50	251	76	1070	179	63	934

Table 4: Number of result objects returned by BRQ

Query Region Radius	EURO	WEBSPAM	TWITTER
1km	1	2	1
2km	2	5	2
5km	4	20	4
10km	7	45	10
20km	15	98	32

We summarize the average distance from the query point to the kth object for BkQ and TkQ in Table 3 and the average number

[6]Based on the result in Figure 26, we use 4K for IR^2 and SKI, and 32K for the other R-tree based indices.

of result objects for TkQ in Table 4 (using the default values for other parameters). The object density for the three datasets could be implied by the two tables.

5.4 Index Settings

As a precursor to applying the benchmark to the geo-textual indexing techniques described in Section 4, we proceed to cover pertinent details of our implementations and settings of the indices. Note that the construction of each index may involve some specific parameters. We conducted experiments to find the best settings for such parameters.

R-tree Based Indices For the R-tree based geo-textual indices that use the inverted file as the text indexing scheme, we apply the B^+-tree for indexing the inverted lists.

The construction of the CDIR-tree involves two parameters. One parameter β ($0 \leq \beta \leq 1$) balances the weight between spatial proximity and text similarity when building a CDIR-tree. We vary β from 0 to 1 and find that the CDIR-tree performs the best on both datasets at $\beta = 0.9$. The other parameter for the CDIR-tree is the number of clusters c. According to the experimental results, we set c at 10, 30, and 40 on EURO, WEBSPAM, and TWITTER, respectively.

For the IR^2-tree, the signature length l_s affects performance. Longer signatures provide more accurate text information so that we can avoid more unnecessary traversal of tree nodes. On the other hand, the space overhead increases as we increase l_s, which incurs more I/O during query processing. We empirically find that the IR^2-tree performs the best at $l_s = 7000$ on EURO, at $l_s = 12000$ on WEBSPAM, and at $l_s = 20000$ on TWITTER.

For the S2I, we need to set a threshold to distinguish frequent words and infrequent terms. Based on the results of additional experiments, we set 100 for EURO and 1000 for WEBSPAM, which yield the best overall query performance. Note that S2I is so space-consuming that we are not able to conduct evaluation for S2I on TWITTER due to the space limitation of our hard disk.

Before constructing the WIBR-tree, we need to iteratively partition all objects in a dataset into two groups using top w frequent words. According to our experimental result, it is of little help in query performance when w exceeds 200, 600, and 1500 on EURO, WEBSPAM, and TWITTER, respectively.

Grid Based Indices For SKIF, the inverted file is not organized in a B^+-tree. Instead, we store the inverted lists and spatial inverted lists in blocks. The size of each block is 4KB, which is the same as that of a B^+-tree node. The offset for each inverted list is loaded into memory before query processing. As for the TS index, we construct a distinct grid based index for each keyword.

The grid resolution affects the performance. We find that for SKIF, 150×150 cells for EURO, 500×500 cells for WEBSPAM, and 1500×1500 cells for TWITTER render the best performance. As for the TS, the number of grid cells is determined by the number of objects that contain the corresponding keyword. We find that 50 objects in each cell gives the best performance.

SFC Based Indices For SFC-Quad, the inverted lists are also stored in blocks. Each block contains 128 postings and is compressed using the OPT-PFD algorithm [6]. Hence, we set the inverted file just as suggested in [6]. In addition, the number of disk sweeps for fetching relevant blocks in inverted lists is set to 1 since 1-sweep performs the best in our datasets.

6. EVALUATION RESULTS

We report on the results of applying the benchmark on the indices covered in Section 4. First, we present the space requirement for each index on the three datasets. Then, we present the results for query processing performance on the BkQ, TkQ, and BRQ, respectively. For each group of experiment, we report the results on EURO, followed by the results on WEBSPAM and TWITTER.

6.1 Space Requirements

Table 5 presents the space requirement for each index. Note that the posting lists for the inverted file of the R*-IF, IR, IRLi, CDIR, and S2I contain both object ID and weight, which is a prerequisite for supporting TkQ, while the posting lists of the other indices only contain object ID. R+IF represents an index scheme using the R-tree and the inverted file, which is introduced in [7] as a baseline method for processing the TkQ query.

As we can see, the space cost of SFC-Quad is significantly lower than those of the others. This is achieved by the OPT-PFD compression algorithm. In addition, the R-tree based text-first indices, namely the IF-R* and S2I, require much more space than the other indices designed for the same type of query (TkQ). This is because the objects may be repeatedly indexed in the IF-R*-tree and S2I. Moreover, the space cost of an index is closely related with the querying performance when a buffer is used, which will be discussed in Section 6.5.

Table 5: Index size

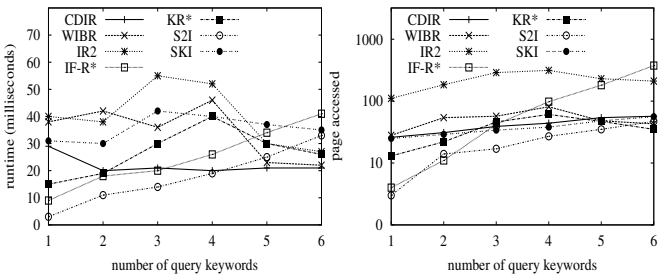
Index	EURO	WEBSPAM	TWITTER
IF-R*	586MB	38GB	173GB
R+IF	25MB	13GB	5.2GB
IR2	246MB	2.1GB	51GB
IR	150MB	75GB	119GB
IRLi	148MB	75GB	118GB
CDIR	158MB	81GB	132GB
WIBR	54MB	13GB	42GB
KR*	187MB	23GB	71GB
S2I	623MB	207GB	N/A
TS	191MB	15GB	12GB
SKIF	38MB	3.2GB	1.4GB
SKI	134MB	4.9GB	35GB
SFC-Quad	19MB	1.4GB	0.8GB

6.2 Varying Query Parameters for BkQ

In this round of experiments, we evaluate the performance of each geo-textual index for processing the BkQ query.

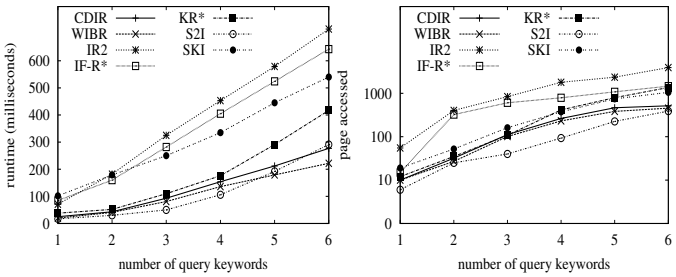
A2 Effect of the number of query keywords: Figures 10, 11, and 12 show the effect of the number of query keywords on datasets EURO, WEBSPAM, and TWITTER.

- S2I performs the best on both EURO and WEBSPAM when the number of query keywords is smaller than 5. Note that S2I cannot run on Twitter since the storage requirement is beyond the available disk space. When the number of query keywords is larger than 5, CDIR performs the best on EURO, and WIBR performs the best on WEBSPAM and TWITTER.
- Among the indices that tightly combine the R-tree with the inverted file, the performance of CDIR is close to that of WIBR, which is the best in most situations, and KR* is the worst.
- The spatial-first index IF-R* that loosely combines the R-tree and the inverted file performs well when the number of keywords is 1, and its performance deteriorates as the number of keywords increases.
- The indices that combine the R-tree and the bitmap (or signature) perform worse than those that combine the R-tree and the inverted file.



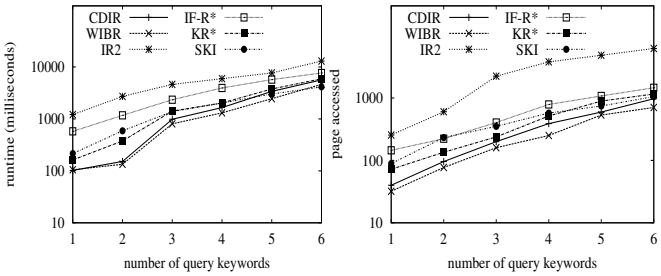
(a) Runtime (b) I/O

Figure 10: Varying # query keywords on EURO for BkQ



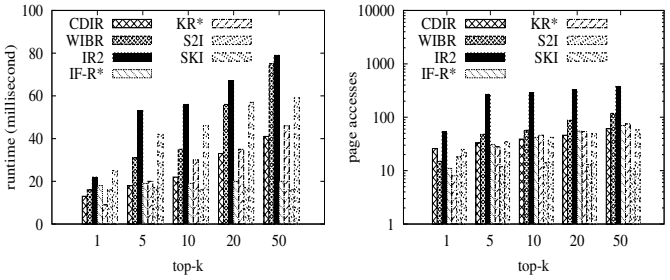
(a) Runtime (b) I/O

Figure 11: Varying # query keywords on WEBSPAM for BkQ



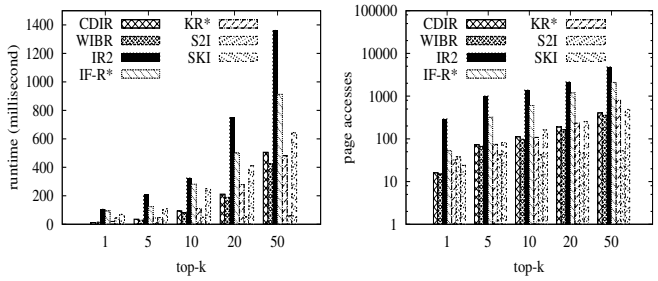
(a) Runtime (b) I/O

Figure 12: Varying # query keywords on TWITTER in BkQ



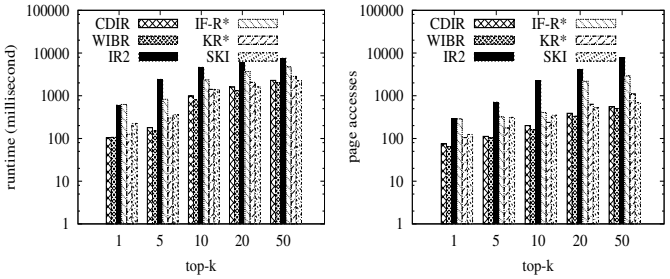
(a) Runtime (b) I/O

Figure 13: Varying top-k on EURO for BkQ



(a) Runtime (b) I/O

Figure 14: Varying top-k on WEBSPAM for BkQ



(a) Runtime (b) I/O

Figure 15: Varying top-k on Twitter for BkQ

As we can see, all indices display an increasing tendency on WEBSPAM and TWITTER in terms of both runtime and I/Os as the number of keywords increases. For the text-first indices, which include IF-R* 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 indices and tightly combined indices, including CDIR, WIBR, IR^2, SKI, 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. However, on EURO as the number of query keywords increases the text-first indices present an increasing trend while the runtime of the other indices drops when the number of keywords is larger than 5. This is because the number of objects in EURO containing 5 or more query keywords is small, and thus the number of nodes containing all the query keywords is small (although they may be far from the query point).

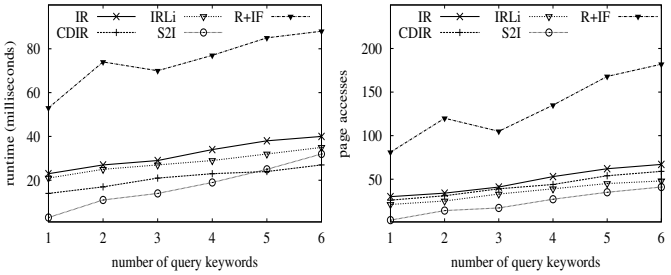
A4 Effect of top-k value: Figures 13, 14, and 15 show results of this experiment in which we investigate the effect of k by varying the value k from 1 to 50. As expected, both the runtime and the I/O cost of each index increase with an increasing value of k. A larger value of k leads to a larger search region in query processing, incurring more accesses to tree nodes.

- S2I consistently performs the best on EURO and WEBSPAM, and WIBR performs the best on TWITTER. The performance of S2I is less affected as we increase the value of k, compared with other indices.
- The performance of the tightly combined indices deteriorates as the k increases. This is because more nodes are accessed to find the k nearest objects containing all the query keywords.

6.3 Varying Query Parameters for TkQ

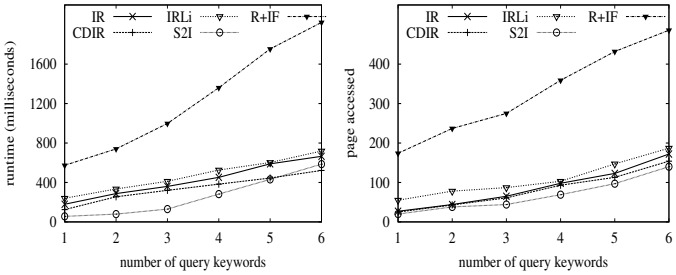
In this round of experiments, we evaluate the performance for processing the TkQ query for each geo-textual index.

A2 Effect of the number of query keywords: Figures 16, 17, and 18 show the result when the number of query keywords is varied from 1 to 6. All the indices present an increasing tendency on both datasets as we increase the number of query keywords, which is different from that for BkQ. This can be explained by the fact that the result objects for TkQ do not necessarily contain all the query keywords, and TkQ always returns k results. We also consider the two baseline methods introduced in the work [7]—the first only uses the inverted file and the second uses both the R-tree and



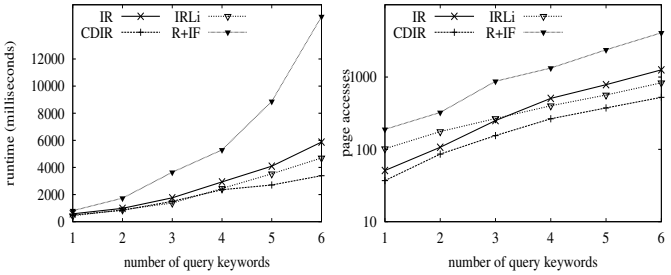
(a) Runtime (b) I/O

Figure 16: Varying #keywords on EURO for TkQ



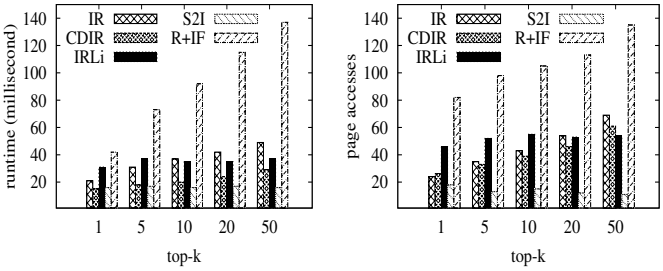
(a) Runtime (b) I/O

Figure 17: Varying #keywords on WEBSPAM for TkQ



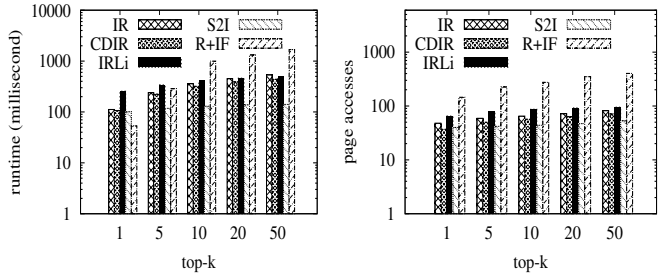
(a) Runtime (b) I/O

Figure 18: Varying #keywords on TWITTER for TkQ



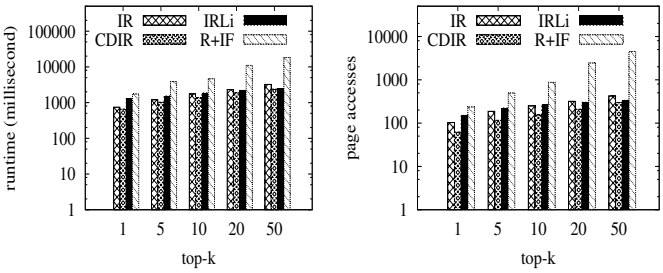
(a) Runtime (b) I/O

Figure 19: Varying top-k on EURO for TkQ



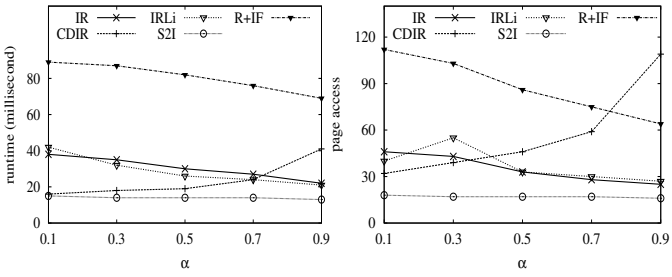
(a) Runtime (b) I/O

Figure 20: Varying top-k on WEBSPAM for TkQ



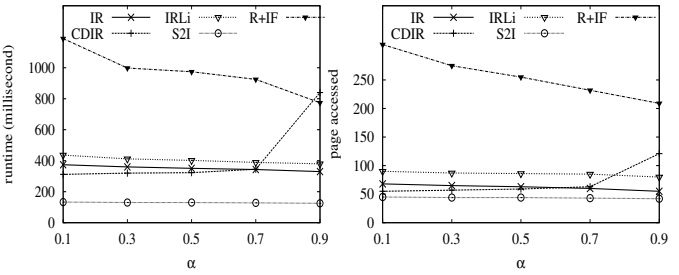
(a) Runtime (b) I/O

Figure 21: Varying top-k on TWITTER for TkQ



(a) Runtime (b) I/O

Figure 22: Varying α on EURO for TkQ



(a) Runtime (b) I/O

Figure 23: Varying α on WEBSPAM for TkQ

the inverted file. They perform worse than the other methods in our experiments. We only include the second baseline, denoted by R+IF, in Figure 16 since the first one performs worse. The interesting findings are stated as follows.

- S2I performs the best when the number of query keywords is smaller than or equal to 4 on EURO and WEBSPAM. However, when it is 5 or 6, CDIR slightly outperforms S2I. The reason is similar to that for the difference on the BkQ. Note again that S2I does not work on TWITTER in our evaluation, and CDIR performs the best on TWITTER.
- CDIR generally performs better than IR and IRLi. This can be explained by its two optimizations over the IR-tree.
- The IR-tree performs similar to the IRLi-tree. It suggests that whether storing the inverted lists for each node separately or storing them together would make little difference on the runtime and I/O cost. The WIR-tree performs similar to the IR-tree and is omitted in the figures.

A4 Effect of top-k value: Figures 19, 20, and 21 report on this experiment in which we investigate the effect of varying the number

of results k from 1 to 50. Figures 19 and 20 show that S2I performs the best on EURO and WEBSPAM and that CDIR performs the best on TWITTER. All the tightly combined hybrid indices significantly outperform the baseline for all values of k in terms of both runtime and I/O cost.

- CDIR outperforms IR and IRLi in almost all cases. When the value of k is smaller than 10, IR outperforms IRLi on EURO; but when we proceed to increase k, both runtime and I/O cost for IRLi would be lower than IR. A similar trend can be also observed on WEBSPAM. This is because the I/O of IRLi for posting lists are fixed for a query, irrespective of k although the I/O for its R-tree component increases as the value of k increases. However the I/O of IR for posting lists correlates to the number of accessed nodes of the IR-tree, which increases with the increase of k. Note that IRLi has a single posting list for each word, while each node of the IR-tree has a posting list for a word appearing in the node. This can be observed from the I/O results. The I/O of IRLi is insensitive to the value k while the I/O of IR increases as k is increased.
- S2I is almost not affected by varying the value of k. This is a characteristic of text-first indexing structures. Its I/O almost remains the same as the value of k is varied.

A5 Effect of α: Figures 22 and 23 report on this experiment that examines the effect of parameter α. From Figure 22, we find that the performance results for both runtime and I/O cost are consistent with those of the previous evaluations, where S2I and baseline respectively bear the lowest and highest runtime or I/O cost. Note that IR and IRLi perform better for large value of α while the CDIR performs better for small value of α since the CDIR takes into account document similarity and the benefit is significant when the text relevance is given higher weight. S2I presents a similar tendency as do IR and IRLi with the variation of α.

6.4 Varying Query Parameters for BRQ

In this round of experiments, we evaluate the BRQ processing performance for each geo-textual index. We only report the runtime due to the space limitation.

A2 Effect of number of query keywords: Figure 24 shows that SFC-Quad performs the best on WEBSPAM and TWITTER, especially for TWITTER, on which it runs with overwhelming superiority. Even on EURO, SFC-Quad achieves the best runtime performance in most cases, though the performance of S2I is close and sometimes WIBR is the best. The good performance of SFC-Quad for the BRQ can be ascribed to the OPT-PFD technique applied for compressing the inverted file, the benefit of the space filling curve while processing BRQ as well as various optimizations.

As expected, nearly all the R-tree based indices display results that are qualitatively similar to those for the BkQ and TkQ if applicable. The performance of the IR^2-tree is not affected by the number of query keywords, as the size of signature file to be loaded is not affected by the number of query keywords (the spatial search range is determined by the query region).

The two grid based indices are slower than most of the other indices. The query processing algorithm for TS is based on a brute force searching technique without any pruning strategy.

A3 Effect of query region size: This experiment investigates the effect of the query region size. We vary the region radius from 1 to 20 km. Figure 25 shows that in general SFC-Quad beats all the other indices on all the three datasets in terms of runtime and I/O cost. In addition, the superiority of SFC-Quad over the other indices becomes more obvious as the region becomes larger. The reason is that multiple ranges of objects that fall in the query region can be easily retrieved based on the Z-curve. Hence, instead of loading the whole inverted list for a particular keyword, SFC-Quad just needs to load the portions of posting lists that fall into relevant ranges. In contrast, the other indices normally need to load more posting lists and index nodes as the query region becomes larger.

6.5 Varying Indexing Parameters

We proceed to evaluate query performance with regard to the R-tree page size for R-tree based indices and the LRU buffer size.

A7 Effect of R-tree page size: Figure 26 shows the effect of varying the R-tree page size from 2K to 32K for the indices based on the R-tree on dataset EURO. The results on WEBSPAM and TWITTER are qualitatively similar. We find that except the IR^2-tree and SKI, all the other R-tree based indices perform better as we increase the page size. This may be explained as follows.

- When the page size is increased, the fanout of the R-tree would also increases. Hence, when a node is loaded into memory, more entries for child nodes are acquired, which might reduce the I/O cost. However, a larger page size also increases the I/O cost of accessing a node. When we further increase the page size to 64K, the performance of these indices does not become better.
- Although the IR^2-tree and SKI benefit from the effect mentioned above, more signatures or longer bitmaps are also fetched when the fanout is increased, which increases the I/O cost.

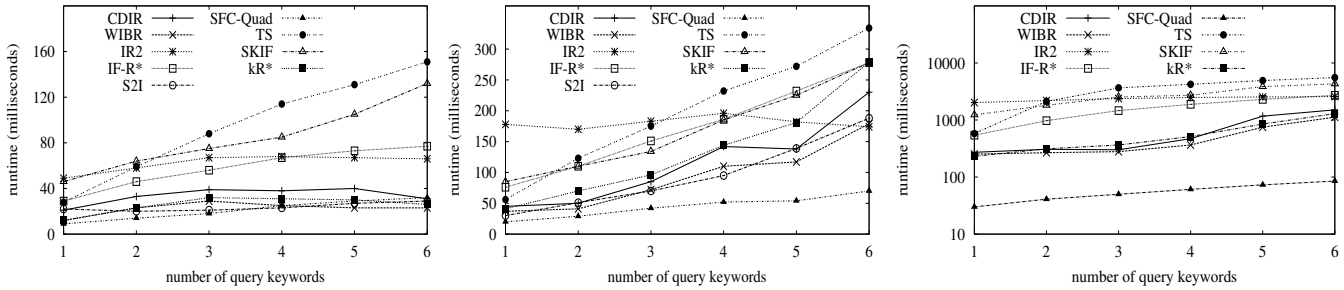
A6 Effect of buffer size: The results are shown in Figures 27, 28, and 29. We simulate the number of accessed pages for each index using an LRU buffer, varying the buffer size from 4MB to 16MB for EURO and from 0.5GB to 2GB for WEBSPAM. As expected, more buffer space improves the I/O performance of all indices. The results also suggest that indices with lower space requirements tend to achieve a more significant improvement of the I/O performance as the buffer size increases. The I/O costs of the IR^2, R+IF, SKI, T-S, SKIF, and SFC-Quad are markedly reduced when the buffer size increases. All of these indices are relatively less space-consuming, which is shown on Table 5. In contrast, the S2I, which requires much more space in comparison to the others, exhibits a nearly constant performance when we vary the buffer size.

6.6 Scalability

In this round of evaluation, we first conduct an experiment varying the number of objects to evaluate the scalability of the indices for each type of query. In addition, to evaluate the effect of the text size of each object, we conduct an experiment varying the average number of words per object for the BRQ on WEBSPAM.

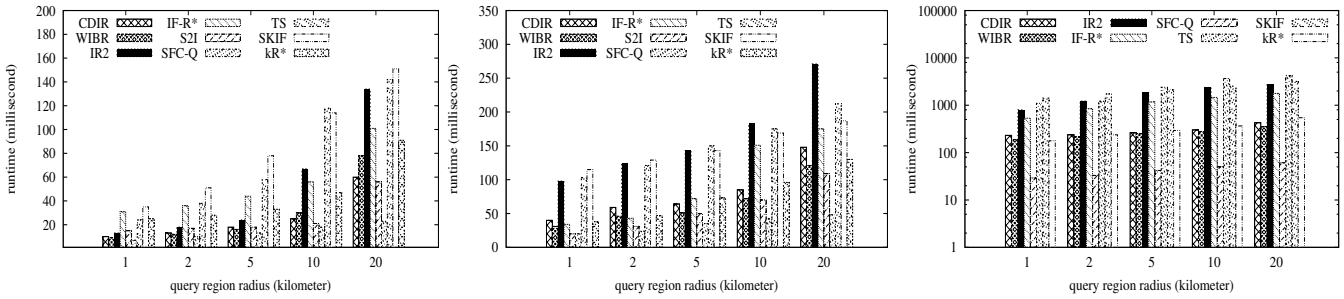
A1 Effect of the number of objects: Figures 30(a), 30(b), and 30(c) show the results of varying the number of objects from 20M to 100M. Increasing the number of objects leads to increasing numbers of tree nodes or grid cells. Also, the length of each posting list when employed increases. We observe that all the indices exhibit a linear growth trend in terms of query runtime for all the three types of queries when we increase the number of objects in dataset.

A9 Effect of text length: Figure 30(d) shows the results of varying the average number of words per object and keeping the number of objects constant. We find that IF-R*, S2I, and TS perform worse as we increase the text length, while the performance of the others is much less affected by the text length. These findings can be ex-



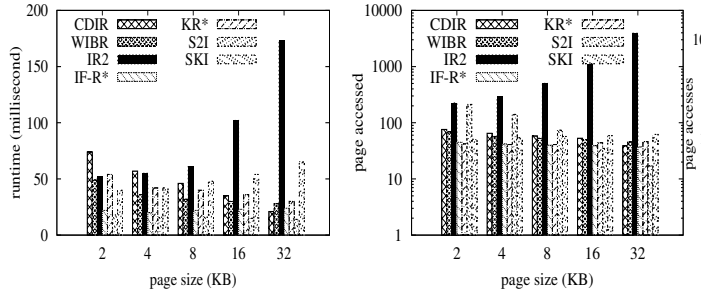
(a) Runtime on EURO (b) Runtime on WEBSPAM (c) Runtime on TWITTER

Figure 24: Varying the number of keywords for BRQ



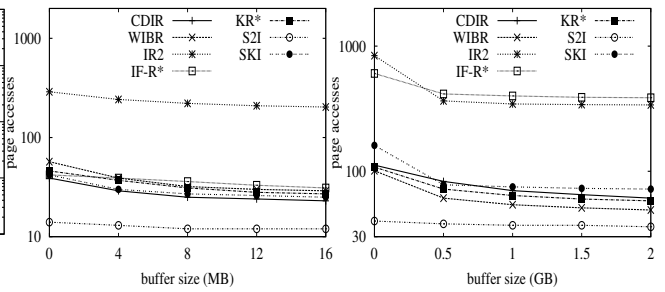
(a) Runtime on EURO (b) Runtime on WEBSPAM (c) Runtime on TWITTER

Figure 25: Varying the query region size for BRQ



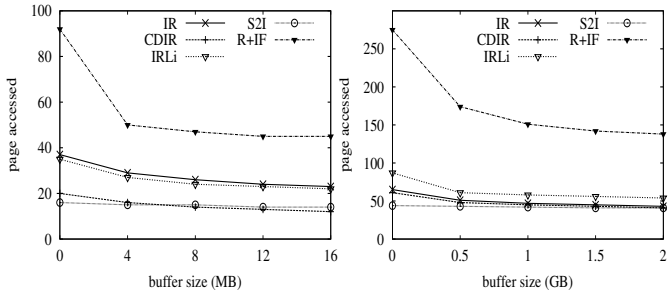
(a) Runtime (b) I/O

Figure 26: Varying the page size



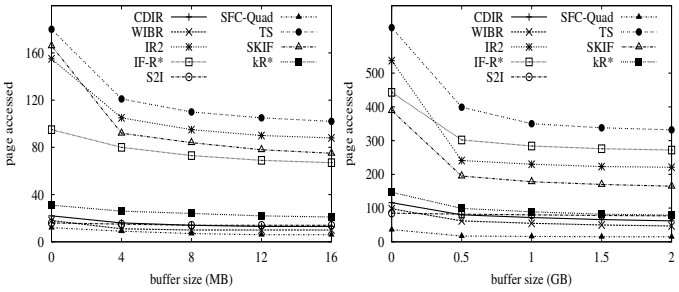
(a) I/O on EURO (b) I/O on WEBSPAM

Figure 27: Varying the buffer size for BkQ



(a) I/O on EURO (b) I/O on WEBSPAM

Figure 28: Varying the buffer size for TkQ



(a) I/O on EURO (b) I/O on WEBSPAM

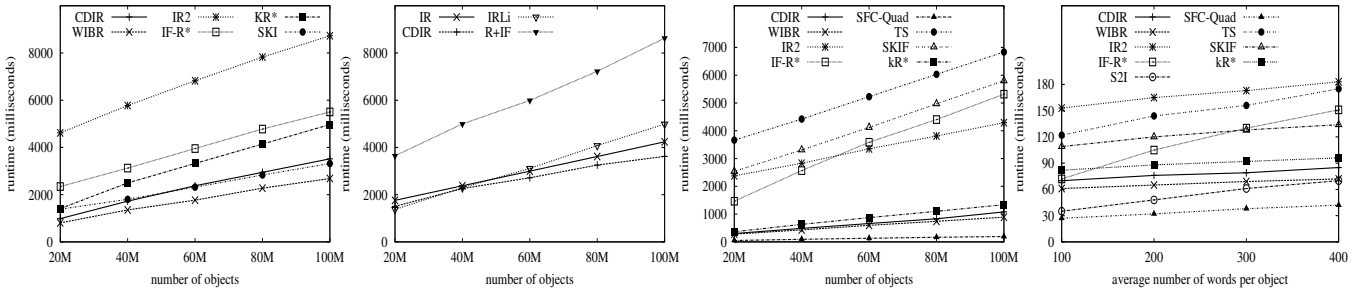
Figure 29: Varying the buffer size for BRQ

plained by that text-first indices would index some objects increasingly many times in the R-trees or cells for different keywords.

7. SUMMARY AND CONCLUSION

By surveying and subjecting 12 geo-textual indexing techniques empirical study, the paper offers structure that may help the area of geo-textual indexing progress more effectively. The paper considers the support for three fundamental kinds of geo-textual queries. We summarize the experimental findings below. Additional details are found in the main body.

- Applications that target the BRQ are best served by the SFC-Quad that beats all the other indices for this type of query.
- For applications that target the BkQ, if the dataset is relatively small and the number of keywords in queries is small, S2I is the most efficient index. However, when the number of query



(a) Varying #objects for BkQ (b) Varying #objects for TkQ (c) Varying #objects for BRQ (d) Varying text length for BRQ

Figure 30: Scalability

keywords is large or a space limitation has to be strictly satisfied, WIBR is preferable since it scales well with the number of query keywords and is much less space-consuming than S2I.

- Likewise, the S2I beats all the other indices on the TkQ except when the number of query keywords exceeds 5, at which point the CDIR starts to perform better. Consequently, if space use is not an issue and the number of query keywords is modest, S2I is the most suitable index; otherwise, CDIR is the best choice. In addition, if the top-k parameter for TkQ is big, IRLi is also a good alternative.
- While a previous study [18] finds that S2I is an order of magnitude better than DIR in terms of both runtime and I/O cost, we find that S2I is just slightly better than CDIR, while CDIR even performs better in some cases. One possible reason for this discrepancy is the performance differences between DIR and CDIR.
- Grid based indices are not attractive for the BRQ compared with the other indices.
- We do not find dramatic difference in relative performance among the indexing techniques for any of the three datasets.
- The query processing of the indices scales linearly with the number of objects and text length per object. Text-first indices are more sensitive to text length than the other types of indices.
- The R-tree page size for all R-tree based indices is experimentally shown to be a factor that makes a great difference on query performance.

8. ACKNOWLEDGMENTS

This work is supported in part by a grant awarded by a Singapore MOE AcRF Tier 2 Grant (ARC30/12) and by the Geocrowd Initial Training Network funded by the European Commission.

9. REFERENCES

[1] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990.

[2] X. Cao, L. Chen, G. Cong, C. S. Jensen, Q. Qu, A. Skovsgaard, D. Wu, and M. L. Yiu. Spatial keyword querying. In *ER*, pages 16–29, 2012.

[3] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *SIGMOD*, pages 373–384, 2011.

[4] A. Cary, O. Wolfson, and N. Rishe. Efficient and scalable method for processing top-k spatial boolean queries. In *SSDBM*, pages 87–95, 2010.

[5] Y.-Y. Chen, T. Suel, and A. Markowetz. Efficient query processing in geographic web search engines. In *SIGMOD*, pages 277–288, 2006.

[6] M. Christoforaki, J. He, C. Dimopoulos, A. Markowetz, and T. Suel. Text vs. space: efficient geo-search query processing. In *CIKM*, pages 423–432, 2011.

[7] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.

[8] C. Faloutsos and S. Christodoulakis. Signature files: An access method for documents and its analytical performance evaluation. *ACM Trans. Inf. Syst.*, 2(4):267–288, 1984.

[9] I. D. Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *ICDE*, pages 656–665, 2008.

[10] R. Göbel, A. Henrich, R. Niemann, and D. Blank. A hybrid index structure for geo-textual searches. In *CIKM*, pages 1625–1628, 2009.

[11] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[12] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems. In *SSDBM*, page 16, 2007.

[13] W. Huang, G. Li, K.-L. Tan, and J. Feng. Efficient safe-region construction for moving top-k spatial keyword queries. In *CIKM*, pages 932–941, 2012.

[14] A. Khodaei, C. Shahabi, and C. Li. Hybrid indexing and seamless ranking of spatial and textual features of web documents. In *DEXA*, pages 450–466, 2010.

[15] G. Li, J. Feng, and J. Xu. Desks: Direction-aware spatial keyword search. In *ICDE*, pages 474–485, 2012.

[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 TKDE*, 23(4):585–599, 2011.

[17] J. Lu, Y. Lu, and G. Cong. Reverse spatial and textual k nearest neighbor search. In *SIGMOD*, pages 349–360, 2011.

[18] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg. Efficient processing of top-k spatial keyword queries. In *SSTD*, pages 205–222, 2011.

[19] S. Vaid, C. B. Jones, H. Joho, and M. Sanderson. Spatio-textual indexing for geographical search on the web. In *SSTD*, pages 218–235, 2005.

[20] D. Wu, G. Cong, and C. S. Jensen. A framework for efficient spatial web object retrieval. *VLDBJ*, 21(6):797–822, 2012.

[21] D. Wu, M. L. Yiu, G. Cong, and C. S. Jensen. Joint top-k spatial keyword query processing. *IEEE TKDE*, 24(10):1889–1903, 2012.

[22] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong. Efficient continuously moving top-k spatial keyword query processing. In *ICDE*, pages 541–552, 2011.

[23] H. Yan, S. Ding, and T. Suel. Inverted index compression and query processing with optimized document ordering. In *WWW*, pages 401–410, 2009.

[24] 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*, pages 688–699, 2009.

[25] D. Zhang, B. C. Ooi, and A. K. H. Tung. Locating mapped resources in web 2.0. In *ICDE*, pages 521–532, 2010.

[26] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma. Hybrid index structures for location-based web search. In *CIKM*, pages 155–162, 2005.