4-2011

IR-Tree: An Efficient Index for Geographic Document Search

Zhisheng LI
Singapore Management University

Ken C. K. LEE
University of Massachusetts Dartmouth

Baihua ZHENG
Singapore Management University, bhzheng@smu.edu.sg

Wang-Chien LEE

Dik Lun LEE

See next page for additional authors

IR-tree: An Efficient Index for Geographic Document Search

Zhisheng Li[†] Ken C. K. Lee[‡] Baihua Zheng[♯] Wang-Chien Lee[‡] Dik Lun Lee[§] Xufa Wang[†]

[†] University of Science and Technology of China, China. zsli@mail.utsc.edu.cn, xfwang@utsc.edu.cn
[‡] Pennsylvania State University, USA. {cklee,wlee}@cse.psu.edu
[♯] Singapore Management University, Singapore. bhzheng@smu.edu.sg
[§] The Hong Kong University of Science and Technology, Hong Kong. dlee@cs.ust.hk

***Abstract*— Given a geographic query that is composed of query keywords and a location, a geographic search engine retrieves documents that are the most *textually* and *spatially* relevant to the query keywords and the location respectively, and ranks the retrieved documents according to their joint textual and spatial relevances to the query. The lack of an efficient index that can simultaneously handle both the textual and spatial aspects of the documents makes existing geographic search engines inefficient in answering geographic queries. In this paper, we propose an efficient index, called *IR-tree*, that together with a top-k document search algorithm facilitates four major tasks in document searches, namely, spatial filtering, textual filtering, relevance computation and document ranking in a fully integrated manner. In addition, IR-tree allows searches to adopt different weights on textual and spatial relevance of documents at the run time and thus caters for a wide variety of applications. A set of comprehensive experiments over a wide range of scenarios has been conducted and the experiment results demonstrate that IR-tree outperforms the state-of-the-art approaches for geographic document searches.**

I. Introduction

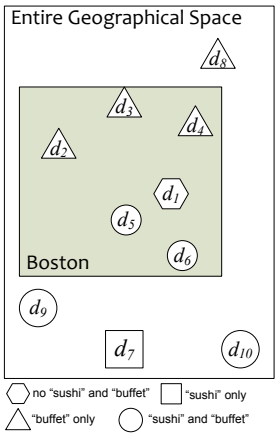
The World Wide Web (WWW) has become the most popular and ubiquitous information media. According to wikipedia, there are 25 billion indexable web pages and over 100 million web sites recorded in 2009, and these numbers continue to grow. Due to the massive number of web pages, search engines that search and rank documents based on their relevances to user queries become essential for information seeking. Search engines are required to determine relevant web pages within a short latency. In other words, *high search efficiency* is one of the key design and implementation objectives of search engines. To achieve this goal, efficient indexing techniques that organize web pages according to their contents are mandated for search engines.

Although web pages are accessible worldwide over the Internet, users are usually only interested in information (such as business listings or news) related to certain locations, e.g., "Las Vegas's restaurant reviews", "Boston's hotels and bars", and "New York's weather". We refer to these queries, which consist of both textual and spatial conditions on documents, as *geographic queries* (or queries, for short), and search engines specialized for answering geographic queries as *geographic search engines*.[1] Here, we use Example 1 as a running example in this paper to illustrate the idea of geographic queries and the function of geographic search engines.

Example 1: (**Running example**) Suppose in a geographic search engine there is a set of 10 documents, $D = \{d_1, \cdots d_{10}\}$, each of which is associated with one spatial location and consists of multiple keywords. Figure 1(a) and Figure 1(b) plot the frequencies of keywords "sushi" and "buffet" and the spatial locations of the documents, respectively.

	document words	
	sushi	*buffet*
d_1	0	0
d_2	0	6
d_3	0	8
d_4	0	3
d_5	1	1
d_6	2	1
d_7	1	0
d_8	0	1
d_9	2	2
d_{10}	1	7
df	2	5
idf	$\log\frac{6}{2}$	$\log\frac{6}{5}$

(a) Word and document frequency



(b) Document spatial distribution

Fig. 1. Example documents

Assume that a user Alice in Boston issues a geographic query "Boston's sushi buffet" and the top-3 documents are to be returned. In this query, "sushi" and "buffet" are query keywords, and "Boston" represents a location/area of her interest.[2] Regardless of the order of their relevances, this example shows a set of candidate result documents $\{d_2, d_3, d_4, d_5, d_6\}$, with respect to both textual relevance and spatial relevance.[3] Here, d_1, although being within "Boston", is not relevant because it contains neither "sushi" nor "buffet". On the other hand, d_7, d_8, d_9, and d_{10}, although being textually relevant to the query, are not within "Boston". The top-3 documents in the candidate set that are most relevant are returned. □

In the past few years, due to increasing application demands and rapid technological advances in geographical information

[1] We refer to "documents" as units of textual information such as web pages in our discussion.

[2] In Figure 1(b), we use a rectangle to annotate the "Boston" area for presentational simplicity.

[3] The definitions of textual and spatial relevances of documents with respect to queries are formalized in Section II.

systems, *geographic search engine* has been receiving a lot of attention from both industry and academia [12], [15], [21], [22], [26]. Same as the conventional search engines, a geographic search engine is required to quickly return documents of high relevance in both textual and spatial aspects to a given geographic query. Serving as the core of search engines, index structures apparently are very important. However, designing an efficient index structure for both textual and spatial information is not trivial, as four major challenges need to be overcome. First, each keyword in the documents is usually treated as one dimension in the document space. Indices for document search need to cover a very large high-dimensional search space. Second, words and locations in geographic documents have different forms of representations and measurements of relevances to a query. A coherent index that can seamlessly integrate these two aspects of geographic documents is very desirable. Third, the words and location of a document have separate influences on the overall relevance of the document to a query, while the relative importance of textual and spatial relevance is very much subjective to the user. Various combinations of these two factors are necessary to accommodate diversified user needs. Thus, an ideal index should allow search algorithms to adapt to different weights between textual and spatial relevance of documents at the run time. Last but not the least, the index structure together with an appropriate search algorithm has to facilitate efficient determination of both textual relevance and spatial relevance of the documents while performing document ranking in order to guarantee high search efficiency.

Existing index structures for geographic search engines can be roughly classified into two approaches. Approach I uses separated indices (one for textual aspect and another for spatial aspect) and Approach II utilizes a combined index that carries both textual and spatial information in a single index structure [12], [15], [18], [22], [26]. Unfortunately, existing approaches are inefficient in processing geographic document search, which motivates this research. In this paper, we design an efficient index structure, namely, *IR-tree*, for geographic search engines which effectively addresses all four challenges discussed above. The strength of IR-tree lays in its ability to perform document search, document relevance computation and document ranking in an integrated fashion. In brief, IR-tree indexes both the textual and spatial contents of documents that enables *spatial pruning* and *textual filtering* to be performed at the same time during query processing. A top-k document search algorithm based on IR-tree combines both the search and ranking processes, thus effectively reducing the number of documents examined. A set of comprehensive experiments over a wide range of system and query parameters has been conducted. The experiment results demonstrate that IR-tree significantly outperforms the state-of-the-art approaches for geographic document searches. In summary, the contributions of this paper are listed as follows.

- We propose *IR-tree* which indexes both the textual and spatial contents of documents to support document retrievals based on their combined textual and spatial relevances, which in turn can be adjusted with different relative weights.
- We design a rank-based search algorithm based on IR-tree to effectively combine the search process and ranking process to minimize I/O costs for high search efficiency.
- We perform a cost analysis for IR-tree and conduct a comprehensive set of experiments over a wide range of parameter settings to examine the efficiency of IR-tree.

The remainder of the paper is organized as follows. Section II provides a background and reviews existing works related to geographic search engines. Section III presents the structure of IR-tree, and discusses variants of aggregated document information and storage schemes as well as manipulations. Section IV details a top-k document search algorithm based on IR-tree. Section V evaluates the performance of the proposed IR-tree in comparison with the state-of-the-art approaches. Finally, Section VI concludes this paper.

II. Preliminaries

In this section, we first define both geographic document search and geographic document ranking based on textual relevance and spatial relevance. Then, we discuss the measurements of textual relevances and spatial relevances, and review existing works proposed for geographic search engines.

A. Geographic Document Search and Ranking

We assume each document d in a given document set D is composed of a set of words W_d, and is associated with a location L_d. Given a query q that specifies a set of query keywords W_q and a query spatial scope S_q, the textual relevance and spatial relevance of a document d to q are formalized in Definition 1 and Definition 2, respectively.

Definition 1: Textual relevance. A document d is said to be textually relevant to a query q if d contains some (or all) of queried keywords, i.e., $W_d \cap W_q \neq \emptyset$. To quantify the relevance of d to q, a weighting function denoted by $\phi_q(d)$ is adopted. Thus, for a given q, $\phi_q(d_1) > \phi_q(d_2)$ means document d_1 is more textually relevant to q than document d_2. □

Definition 2: Spatial relevance. A document d is said to be spatially relevant to a query q if the location of d overlaps with the query spatial scope of q, i.e., $L_d \cap S_q \neq \emptyset$. Let $\varphi_q(d)$ be a scoring function to quantify the spatial relevance of d to q. Thus, $\varphi_q(d_1) > \varphi_q(d_2)$ indicates that document d_1 is more spatially relevant to q than document d_2. □

Accordingly, *geographic document search* finds from D those documents that are both textually relevant and spatially relevant to a given query (as stated Definition 3), and *geographic document ranking* ranks the documents based on the joint textual and spatial relevances with respect to the query (as formalized in Definition 4).

Definition 3: Geographic document search. Geographic document search identifies those documents in D that are both textually and spatially relevant to q. □

Definition 4: Geographic document ranking. Geographic document ranking returns the k most relevant documents, sorted in descending order of their joint textual and spatial

relevances, denoted by $\psi_q(d)$, with respect to q.[4] Here, k is specified by the user and $\psi(d)$ is formulated in Equation (1).

$$\psi(d) = \begin{cases} \alpha \cdot \phi(d) + (1-\alpha) \cdot \varphi(d) & \text{if } \phi(d) > 0 \text{ and } \varphi(d) > 0 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

The joint relevance of d is the weighted sum of textual relevance and spatial relevance, with a parameter α ($\in [0, 1]$) controlling their relative weights. □

B. Document Relevance Measurement

The accurate estimation of the relevance between documents and user queries is critical to the perceived quality and performance of search engines. Specific to geographic search engines, we study some existing weighting functions for estimating textual relevance and spatial relevance.

Textual relevance. There are various models (e.g., vector space model, probabilistic model, language model, etc. [4]) to measure the relevance of documents to a given query. Among all those, TF-IDF [13], [24] is the most widely used. There are many TF-IDF variants sharing the same fundamental principles, though using different *tf* and *idf* formulations. For simplicity, we consider the generic one hereafter. TF-IDF weighs a term in a document based on *term frequency* (*tf*) and *inverse document frequency* (*idf*) [16].[5] A term frequency $tf_{w,d}$ measures the number of times a word w appears in a document d, which indicates the importance of the word within the document. On the other hand, the inverse document frequency, $idf_{w,D}$ measures the specificity (importance) of a word w in a document set D, i.e., $idf_{w,D} = \log \frac{|D|}{|\{d|d \in D \wedge tf(w,d) > 0\}|}$. However, under the context of geographic document search, the *idf* of a word w, denoted by $idf_{w,D,S}$, is defined corresponding to a candidate document set D_S instead. The documents in D_S have their locations fully covered by the query spatial scope S. Note that the candidate documents are completely subject to S provided at the query time. Equation (2) formulates $idf_{w,D,S}$.

$$idf_{w,D,S} = \log \frac{|D_S|}{df_{w,D_S}} \quad (2)$$

where $D_S = \{d|d \in D \wedge L_d \subseteq S\}$ and $df_{w,D_S} = |\{d|d \in D_S \wedge tf(w,d) > 0\}|$. In this paper, the textual relevance of a document d to q is defined in Equation (3).

$$\phi_q(d) = \sum_{w \in W} \left(tf_{w,d} \cdot idf_{w,D,S}\right) \quad (3)$$

Spatial relevance. The spatial relevance of a document d, denoted as $\varphi(d)$, depends on the types of the spatial relationships defined between a document location L_d and a spatial scope S. Commonly adopted relationships as discussed in [26] include:

1) **Enclosed**. $\varphi(d)$ is set to 1 if the corresponding location is fully enclosed by the query scope, i.e.,

$$\varphi(d) = \begin{cases} 1 & \text{if } L_d \subseteq S; \\ 0 & \text{otherwise.} \end{cases}$$

2) **Overlapping**. $\varphi(d)$ is set to the fraction of the document location that is covered by the spatial scope, i.e., $\varphi(d) = \frac{Area(L_d \cap S)}{Area(L_d)}$.
3) **Proximity**. $\varphi(d)$ is represented by the inverse of the distance between the center of L_d and that of S, i.e.,

$$\varphi(d) = \begin{cases} \frac{1}{dist(L_d,S)} & \text{if } L_d \subseteq S; \\ 0 & \text{otherwise.} \end{cases}$$

Without loss of generality, we focus on the proximity in the following discussion. Other types of spatial relevances can be supported by substituting proximity with a desired spatial relevance calculation.

C. Related Works

Here, we review existing works in textual index, spatial index and geographic document search engines.

Textual Index. To facilitate the calculation of TF/IDF of documents, inverted files, which are collections of inverted lists, are proposed. Each inverted list l_w serves one word w. An element $\langle d, tf_{w,d} \rangle$ in l_w records a document d with $tf_{w,d} > 0$; and each list is arranged in descending order of documents' *tf* values. Because conventional TF-IDF calculation does not consider any query spatial scope, the IDF value of a given word within a document set D can be easily obtained, i.e., $idf_{w,D} = \log \frac{|D|}{|l_w|}$ where $|D|$ and $|l_w|$ represent the cardinality of D and the length of l_w, respectively. Some proposed optimization techniques only read the initial portion of lists to avoid accessing unnecessary documents [2], [3], [23]. In geographic document search, computing $idf_{w,D,S}$ as shown in Eq. (2) is much more challenging than computing $idf_{w,D}$ in conventional search engine.

Spatial Index. Spatial indices [9] have been extensively studied in the spatial database community [25]. Among all the existing spatial indices, R-tree [11] is very well-received. In an R-tree, spatial objects are first abstracted as minimum bounding boxes (MBBs). Those spatial objects whose MBBs are closely located are clustered in leaf nodes. Similarly, leaf nodes with closely located MBBs are grouped to form non-leaf nodes. This grouping process propagates until the root node is formed. Aggregate R-tree (aRtree) [17] extends R-tree to support spatial aggregation queries to find aggregated information within a search area. Also, R-tree and its variants can support run time object ranking [14].

Geographic Search Engine. As briefly mentioned in Section I, two approaches are used by existing geographic search engines, with Approach I using separated indices for spatial information and textual information, and Approach II using a combined index [12], [15], [18], [22], [26]. However, they both are not efficient.

Approach I logically extends conventional textual search engines with spatial filtering capability of Quad-tree, R-tree and Grid index as suggested in [5], [18], [22], respectively. As an example, in [5], the most recent work of Approach I, an inverted file is created to index words of documents and a grid index is created to index locations of documents. Based on two indices, a search generally follows a three-step process.

[4] If the context is clear, we omit the subscript q for notational simplicity hereafter.

[5] In this paper, a term is equivalent to a word.

- Step 1: retrieving textually relevant documents with respect to query keywords via a conventional textual index.
- Step 2: filtering out the documents obtained from Step 1 that are not covered by the query spatial scope.
- Step 3: ranking the documents from Step 2 based on the joint textual and spatial relevances in order to return the ranked results to the user.

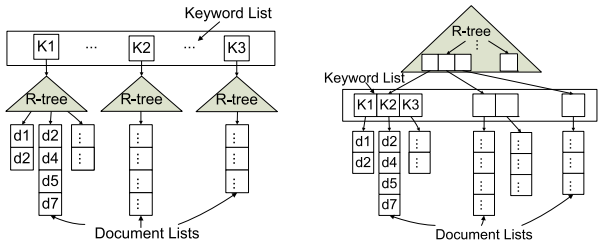
We use the running example (i.e., Example 1) to illustrate the above three-step process. First, Step 1 retrieves all documents textually relevant to query keywords and ignores those textually irrelevant documents (i.e., d_1). As Alice is only interested in the query spatial scope "Boston", documents outside the scope are discarded in Step 2, i.e., d_7, d_8, d_9, and d_{10}. Finally, in Step 3, the remaining documents are ranked according to their TF-IDF scores as listed in Table I; and the top three documents (i.e., d_6, d_3, and d_5) are returned.

TABLE I
GEOGRAPHIC DOCUMENT SEARCH RESULT

	d_6	d_3	d_5	d_2	d_4
TF-IDF	1.03	0.63	0.55	0.48	0.24

Approach I is inefficient due to the following reasons. First, a keyword based search may retrieve a large number of textually relevant documents that are outside the spatial scope. Take our evaluation (to be discussed in Section V) as an example. More than 90% of the textually relevant documents are outside the query spatial scopes. Although it is possible to reorder Step 1 and Step 2 based on their selectivities, performance improvement is rather limited if the selectivities in Step 1 and 2 are both high. Besides, the ranking process is not incremental, i.e., it has to sort *all* of the candidate documents based on the joint textual and spatial relevances in Step 3 in order to find the top-k documents. As the total number of candidate documents is usually much larger than k, document ranking becomes very expensive. Third, these three steps are performed sequentially, prolonging the processing time and requiring a large memory storage to buffer the intermediate results between steps.

To improve the search efficiency, Approach II combines the spatial locations and textual contents of documents together and builds one index on them. Existing works following Approach II include [12], [26], [15]. In [15], the name of a location and every word of a document are combined as a new word. Referring to our running example, d_2 produces a new word "Boston_buffet" (use a location name as a prefix and a word as suffix connected by an underscore). Then, an inverted file based on those new words is created to support geographic searches. However, this approach simply treats locations as texts and cannot deal with various spatial relevance computations (as discussed in the previous subsection). On the other hand, in [26], two hybrid indices are proposed, namely, (a) an inverted file on top of R-trees (see Figure 2(a)), referred to as $\mathsf{Hybrid_I}$, and (b) an R-tree on top of inverted files (see Figure 2(b)), referred to as $\mathsf{Hybrid_R}$. Thus, a search upon $\mathsf{Hybrid_I}$ first locates a collection of documents based on search keywords and then based on locations. The search strategy is reversed for $\mathsf{Hybrid_R}$. However, these hybrid indices do not integrate the textual filtering and spatial filtering seamlessly.



(a) $\mathsf{Hybrid_I}$ (b) $\mathsf{Hybrid_R}$

Fig. 2. Two hybrid indexing schemes

KR*-tree is another type of hybrid indices which supports searches for spatial objects based on their textual contents [12]. It extends $\mathsf{Hybrid_R}$ by augmenting with a set of words in the internal nodes. Thus, it can support both spatial and textual filtering simultaneously. The query processing algorithm finds the nodes that are spatially relevant to the query spatial scope and containing the query keywords. It then evaluates all the objects in these nodes for ranking. Along the same line, IR^2-tree [8] builds an R-tree and uses signature files (rather than a set of words) to record the document words associated with nodes in the index. Signature files reduce the storage overhead and R-tree can quickly determine the documents spatially covered by a query spatial scope. However, signature file can only determine whether a given document contains query keywords but fail to order them based on the textual relevance. In brief, $\mathsf{Hybrid_R}$, KR*-tree and IR^2-Tree are not efficient due to separation of document search and document ranking. After the document search step, a large number of candidate documents are usually retrieved but only k of them are returned after document ranking. Consequently, the evaluation of those non-result candidates is a waste. Finally, although KR*-tree, IR^2-Tree and our IR-tree proposed in this paper are built on top of R-tree, they are very different in terms of structures, functionalities, and extensibility to searches with various relevance requirements.

III. IR-TREE

In this section, we present *IR-tree*, an efficient index that provides the following required functions for geographic document search and ranking: i) **spatial filtering:** all the spatially irrelevant documents have to be filtered out as early as possible to shrink the search space; ii) **textual filtering:** all the textually irrelevant documents have to be discarded as early as possible to cut down the search cost; and iii) **relevance computation and ranking:** since only the top-k documents are returned and k is expected to be much smaller than the total number of relevant documents, it is desirable to have an incremental search process that integrates the computation of the joint relevance and document ranking seamlessly so that the search process can stop as soon as the top-k documents are identified.

In addition, IR-tree is designed by taking into account the storage and access overheads since a document set is very large in terms of numbers of documents and their words. In the following, we first detail the design of IR-tree indexing structure. Then, we discuss the notion of *document summaries* (and several variants) that facilitate search space exploration

and pruning as well as ranking. Thereafter, we discuss the storage schemes and index manipulation methods.

A. IR-tree Indexing Structure

In order to support efficient geographic document search, the IR-tree clusters a set of documents into disjointed subsets of documents and abstracts them in various granularities. By doing so, it enables the pruning of those (textually or spatially) irrelevant subsets. The efficiency of IR-tree depends on its pruning power, which, in turn, is highly related to the effectiveness of document clustering and the search algorithms. Our IR-tree clusters spatially close documents together and carries textual information in its nodes. These designs distinguish our IR-tree from other hybrid indices (as reviewed in Section II-C). IR-tree associates each leaf entry with an inverted file and associates a document summary that provides textual information of documents with each node so that the *tf* and *idf* values of the document words can be estimated at nodes without examining individual documents. Figure 3 depicts an IR-tree indexing structure. An inverted file consists of a list of words, with each corresponding to a word w and pointing to a list of documents that contain w. On the other hand, for each node i, a document summary about a set of documents D_i indexed beneath i is captured as a three-element tuple:

$$\langle A_i, |D_i|, \cup_{w \in W_i} \{df_{w,D_i}, TF_{w,D_i}\} \rangle.$$

In the tuple, the first element A_i is the minimal bounding box (MBB) covering all of the locations L_d of documents d in D_i (i.e., $A_i = MBB(\cup_{d \in D_i} L_d)$). Next, $|D_i|$ refers to the cardinality of the document set D_i. The third element is a set of (df_{w,D_i}, TF_{w,D_i}) pairs. For each word w that appears in at least one document in D_i (i.e., W_i), df_{w,D_i} represents the number of documents in D_i that contain w and TF_{w,D_i} is the aggregated information about the *tf* values of w in D_i. We investigate two different representations of TF_{w,D_i}, and they will be discussed in the next subsection. Notice that the document summary of a non-root node i is stored with i's parent node h. Then, given a query that reaches i's parent node h, it can decide whether i contains potential result documents (i.e., whether the examination of i is necessary) based on the document summary.

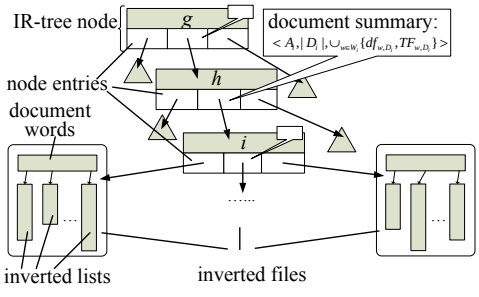
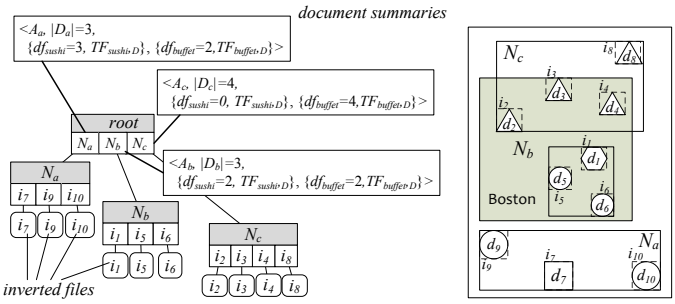


Fig. 3. IR-tree indexing structure

To facilitate our discussion, we use Example 2 to illustrate an IR-tree based on our running example.

Example 2: Figure 4(a) shows an IR-tree for the example document set, with the minimum and maximum node fanouts set to 3 and 4, respectively and Figure 4(b) shows the distributions of MBBs. Documents d_1, d_5, and d_6 that are spatially close to each other are grouped into node N_b. Similarly, d_7, d_9, and d_{10} form the node N_a and d_2, d_3, d_4 and d_8 form the node N_c. These three nodes are further grouped together to form the root node. □



(a) IR-tree content (b) Node MBBs

Fig. 4. An example IR-tree

As defined in Section II-B, the textual relevance between a document and a query is dependent on both the *tf* values and the *idf* values of documents with respect to query keywords. To facilitate the discussion, let D_i represent the set of documents indexed beneath node i and W_i represent the set of words appearing in at least one document d ($\in D_i$) (i.e., $W_i = \cup_{d \in D_i} W_d$). Since A_i, $|D_i|$ and df_{w,D_i} values are maintained in document summaries, the candidate document set D_S (i.e., $\subseteq D$) can be formed as early as the search reaches a set of nodes N_S such that A_i in a document summary for any node $i \in N_S$ is fully bounded by the query spatial scope S, i.e., $\forall_{i \subseteq N_S} A_i \subseteq S$. Then, the *idf* value for a given query keyword w can be determined over those identified nodes N_S without scanning the documents indexed beneath them as:

$$idf_{w,D,S} = \log \frac{\sum_{\forall i \in N_S} (|D_i|)}{\sum_{\forall i \in N_S} (df_{w,D_i})}. \quad (4)$$

Then, with TF_{w,D_i} values (or TF values for simplicity) maintained by a document summary available, the textual relevance of a document $d \in D_i$ to q can be estimated by $\phi(i) = \sum_{w \in W_q} idf_{w,D,S} \cdot TF_{w,D_i}$. We formulate TF in such a way that $\phi(i)$ provides a good estimation of $\phi(d)$ for $d \in D_i$. Consequently, the estimated values, although not necessarily the exact values, are useful to prune those nodes that do not contain any qualified documents in an early stage of a search process. Notice that this pruning is very flexible. It can be based on the joint spatial and textual relevance, the pure spatial relevance, or the pure textual relevance. Further, based on the estimated values, a ranking algorithm can order the candidate documents and give higher priorities to those nodes which are more likely to contain result documents. The top-k document search algorithm will be detailed in Section IV.

Although at the first glance it looks similar to Hybrid_R, IR-tree is indeed *structurally* and *functionally* different from Hybrid_R. From the design perspective, Hybrid_R focuses *purely* on document search, not supporting document ranking that is critical to top-k document retrieval. On the contrary, IR-tree maintains document summaries at different levels which enables an estimation of the joint relevances of documents to a

given query without reaching the leaf level, such that the top-k result documents can be determined without getting all the candidates ranked. On the other hand, IR-tree, KR*-tree and IR^2-Tree are all based on R-tree structure. While KR*-tree and IR^2-Tree are mainly for serving document filtering, they do not support document ranking. Nevertheless, IR-tree supports *both* geographic document search and ranking. Besides, IR-tree can be tailored, though not explored in this paper, to cater for various application needs by adjusting the content of document summaries to support other relevant measurements based on different TF-IDF variants.

B. Two Alternative TF_{w,D_i}

As shown earlier, TF plays an important role in estimating the textual relevance $\phi(i)$ at node i. Thus, two TF variants are considered in this work.

TFs. Given a query q related to a query spatial scope S, the textual relevance of a document d with respect to a query keyword w depends on the product of $tf_{w,d}$ and $idf_{w,D,S}$. As such, a straightforward approach to present TF_{w,D_i} is to maintain in each document summary the $tf_{w,d}$ values for each word $w \in W_i$; hence, $TF_{w,D_i} = \cup_{d \in D_i \wedge tf_{w,d}>0}\{\langle d, tf_{w,d}\rangle\}$. In other words, TF_{w,D_i} resembles an inverted file with respect to all the documents indexed beneath node i. With document information available at each level of IR-tree, the search efficiency is improved at the expense of extremely high storage overhead for $O(|D_i| \times |W_i|)$ TF values. Therefore, keeping all tfs in every document summary is very storage expensive.

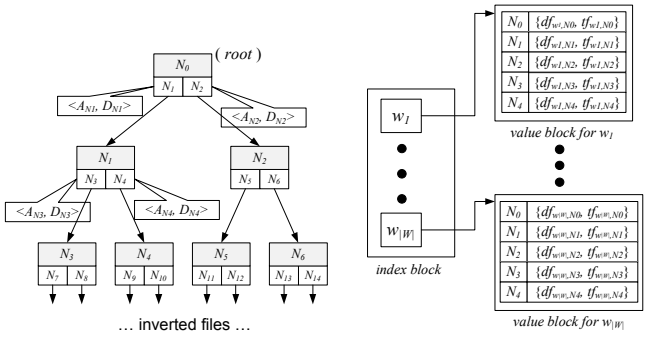
TF Maximum. In order to alleviate the storage overhead, we store at node i the TF_{w,D_i} value as the maximum among all tf_{w,D_i} values for all documents in D_i, i.e., $\mathsf{max}_{d \in D_i}(tf_{w,d})$ (denoted by tf^{max}_{w,D_i}). This reduces the overall storage overhead for TF values down to $O(|W_i|)$ and provides a reasonably good estimation of the textual relevance of the underlying documents as $\forall d \in D_i$, $(tf_{w,d} \cdot idf_{w,D,S}) \leq (tf^{max}_{w,D_i} \cdot idf_{w,D,S})$. The search for top-$k$ documents can follow a path through node i to reach result documents based on maximal TF values, and the details about the search will be presented in Section IV.

C. Keyword Based Storage Scheme

As the index I/O cost directly affects the search performance, how an index is organized on disk is an important issue. In this subsection, we present the IR-tree storage scheme.

In an IR-tree, every non-leaf node h maintains pointers to individual child nodes i together with their document summaries $\langle A_i, |D_i|, \cup_{w \in W_i}\{df_{w,D_i}, TF_{w,D_i}\}\rangle$. Consider a scenario where $|W_i| = 2000$ and TF_{w,D_i} as the TF maximum, df_{w,D_i} and w take 2, 1 and 1 bytes, respectively. Thus, one document summary consumes $(2+1+1) \times 2000$ bytes, i.e., eight 1KB pages. Further, a non-leaf node with fanout f takes $8f$ 1KB pages. Since only a few keywords are queried, loading complete document summary corresponding to all the words is not necessary as it accesses unnecessary DF/TF values for non-query keywords.

Instead, we partition each document summary at node i, i.e., $\langle A_i, |D_i|, \cup_{w \in W_i}\{df_{w,D_i}, TF_{w,D_i}\}\rangle$ into different segments, such as $\langle A_i, |D_i|\rangle$, $\langle\{df_{w_1,D_i}, TF_{w_1,D_i}\}\rangle$, $\cdots$ $\langle\{df_{w_{|W_i|},D_i}, TF_{w_{|W_i|},D_i}\}\rangle$. The first segment $\langle A_i, |D_i|\rangle$, which is independent to document words, is stored with the parent node h of node i. The remaining segments are stored separately. Observing that a search for documents textually relevant to a word w only requires DF/TF values related to w, we strategically store $\langle\{df_{w,D_x}, TF_{w,D_x}\}\rangle$ related to the same word w but for different nodes together in the same memory block. Here, a memory block refers to a linked list of pages.



(a) IR-tree tree hierarchy (b) DF/TF value blocks

Fig. 5. Keyword based storage scheme

Figure 5 illustrates our storage scheme, namely *keyword based storage scheme*. It consists of three components, namely, (i) a tree hierarchy that presents the nodes' parent-child relationship in an IR-tree to support traversals for searches, (ii) (conventional) inverted files, and (iii) DF/TF value blocks, which keep TF and DF values for the same words in the same memory blocks. An example tree hierarchy is shown in Figure 5(a), in which every node has a fanout of 2. As shown, $\langle A_{N_3}, D_{N_3}\rangle$ and $\langle A_{N_4}, D_{N_4}\rangle$ of node N_3 and N_4, respectively, are kept at the parent node N_1. Notice that in our implementation, each node sits in one disk page. The inverted files pointed by leaf node entries are stored independently. Due to limited space, they are not depicted. Figure 5(b) illustrates DF/TF value blocks. It has two parts, namely, index block and value blocks. Index block consists $|W|$ entries. Each entry for one word w points to a value block that contains DF/TF values for w. Inside a value block of word w is a list of $\{i, df_{w,D_i}, TF_{w,D_i}\}$ entries.

When a geographic query with query keywords W_q and a query spatial scope S is processed, DF/TF value blocks for those query keywords W_q are preloaded, and a traversal starts from the root of the tree hierarchy. When a node is traversed, its child node i with A_i stored in tree hierarchy is compared with S and the corresponding DF/TF values in the preloaded DF/TF value blocks are accessed and evaluated. The detailed search algorithm will be presented in Section IV.

D. IR-tree Manipulations

The IR-tree can be manipulated with three operations, namely, bulkloading documents, inserting documents and deleting documents. Given a set of documents, bulkloading creates an IR-tree from scratch. Tne pseudo-code is depicted in Algorithm 1. As a brief description, it first clusters documents based on their spatial locations into leaf-level entries, and then

groups the formed entries as nodes in a bottom-up fashion repeatedly until the root is formed.

Algorithm 1 IR-tree Construction

Input: a document set, D; minimal node fanout, min; maximal node fanout, max;
Output: the root of an IR-tree;
Procedure:
1: $N_e \leftarrow \emptyset$
2: **for** each $d \in D$ **do**
3: geo-code d and represent L_d with MBB m_d;
4: **if** $\exists e \in N_e, m_e = m_d$ **then**
5: add d to e's document set D_e;
6: **else**
7: create a new entry e;
8: set $m_e \leftarrow m_d$ and $D_e \leftarrow \{d\}$;
9: $N_e \leftarrow N_e \cup \{e\}$;
10: **for** each $e \in N_e$ **do**
11: build inverted file with each list l_w w.r.t. every word w in at least one document $d \in D_e$;
12: **while** $|N_e| > n_{max}$ **do**
13: cluster N_e according to min/max into nodes, represented as new entries N'_e; form document summary for e in N'_e;
14: $N_e \leftarrow N'_e$;
15: create the root node to cover N_e and their document summaries;
16: **output** the root node;

Here, we assume that each document is mapped to one location L_d, and documents mapped to the same location are collected in a set of entries N_e (line 2-9). Accordingly, an inverted file is created for each entry $e \in N_e$ to keep the term frequencies of different words (line 10-11). Further, entries in N_e are clustered according to their locations to form IR-tree nodes, each of which is associated with a document summary. The number of entries included into one node is bounded by the minimum and the maximum fanouts, i.e., min and max, respectively (line 12-14). Typically, min is set to 30% of max and max is determined as the quotient of disk page size divided by the maximum entry size. The entries for generated nodes (i.e., N'_e) are grouped by the same clustering logic. At last, when the number of generated entries (i.e., $|N_e|$) is small enough ($\leq max$) to be represented by a node, a root node is formed and returned to complete this bulkloading (line 15-16).

Besides, the structure of IR-tree can be easily adapted when new documents are added and/or existing documents are deleted. When a new document d is inserted, based on L_d, we traverse an IR-tree to reach a leaf node that provides the smallest expanded area after L_d is included. Then, the document summaries of all nodes on the path from the leaf node to the root are updated to accommodate d. On the other hand, deletion of a document d is handled by first locating the leaf node holding d. Then, the document summaries of all nodes on the path from the leaf node to the root are updated to reflect the removal of d. We omit the detailed insertion and deletion algorithms and handling of some situations (such as node overflow and node underfull) because they are similar to those for conventional R-trees.

IV. Top-k Document Search

In light of that the size of the candidate set is much larger than the required number of result documents, k, we develop a top-k document search algorithm based on IR-tree to improve the efficiency of geographic document search. The top-k search algorithm effectively avoids the computation of the relevance scores for most of, if not all, non-result candidate documents. Algorithm 2 outlines the logic of the top-k search algorithm. It is composed of two steps, namely, (i) IDF-Calculation and (ii) Top-k Document Retrieval. As the names suggest, the former determines IDF for query keywords; and the latter computes the relevance of candidate documents and returns k most relevant documents. In the following, we detail these two steps and discuss the advantage of this algorithm.

Algorithm 2 Top-k Document Search

Input: the root of an IR-tree r;
a query q with keywords, W_q, and spatial scope, S_q;
the requested number of result documents, k;
the ratio between spatial and textual relevance, α;
Output: a set of top k documents
Procedure:
1: $(\{idf_{w,D,S_q}, \forall w \in W_q\}, B) \leftarrow$ IDF Calculation(r, W_q, S_q);
2: **if** $B \neq \emptyset$ **then**
3: **return** Top-k Document Retrieval($\{idf_{w,D,S_q}, \forall w \in W_q\}$, B, W_q, S_q, α, k);
4: **return** $\emptyset$;

A. Step One: Derivation of IDF values

As defined in Equation (2), *idf*, a component to TF-IDF for textual relevance, is a fraction between $|D_S|$ and df_{w,D_S}. As the first step, we determine all of the documents that are located inside the query scope S_q for a query q. With $|D_i|$ (as a part of document summary) available with every node i in an IR-tree, the search only needs to traverse to the nodes i with $A_i \subseteq S_q$. Notice that if node i is already fully covered by S_q, it is not necessary to visit i's child/descendant nodes in order to determine df_{w,D_S} and $|D_S|$. Meanwhile, we need to identify some candidate nodes that contain result documents being spatially and textually relevant to a query. Documents beneath those nodes fully covered by S are spatially relevant to the query but may not be textually relevant. To determine whether a node i contains any textually relevant document, we examine the df_{w,D_i} values corresponding to each query keyword $w \in W_q$ and discard the node if $\forall w \in W_q, df_{w,D_i} = 0$. Based on these ideas, we formulate Algorithm 3.

In the algorithm, we use a depth-first traversal strategy to determine *idf* for all query keywords and to collect nodes with candidate documents. Notice that breath-first traversal is also applicable. In details, an entry ϵ that represents a node is examined from a stack in each iteration. If ϵ has its area A_ϵ not covered by S_q, ϵ and all the nodes indexed beneath it can be discarded immediately. Otherwise, A_ϵ and S_q overlap in two cases: (i) A_ϵ covered by S_q in part, and (ii) A_ϵ completely covered by S_q. For (i), the child nodes of ϵ, that have smaller area than ϵ, are pending for later examination (line 12). For (ii), we accumulate $|D_\epsilon|$ to D_S (line 7). In addition, we check whether ϵ can contribute to DF_w by checking df_{w,D_ϵ} corresponding to each query keyword $w \in W_q$. Once a non-zero df_{w,D_ϵ} value is found, DF_w will be updated accordingly and ϵ is inserted into B. Here, B is a buffer to keep track

Algorithm 3 IDF Calculation

Input: the root of an IR-tree, r;
query keywords W_q and query spatial scope S_q;
Output: $\{idf_{w,D,S_q}, \forall w \in W_q\}$, and a buffer of nodes B;
Procedure:

1: $D_S \leftarrow 0$; $DF_w \leftarrow 0, \forall w \in W_q$;
2: push r to an empty stack T;
3: **while** T is not empty **do**
4: pop an entry ϵ from T;
5: **if** $A_\epsilon \cap S_q \neq \emptyset$ **then**
6: **if** $A_\epsilon \subseteq S_q$ **then**
7: $D_S \leftarrow D_S + |D_\epsilon|$; // count the no. of document;
8: **if** $\exists_{w \in W_q} df_{w,D_\epsilon} > 0$ **then**
9: $DF_w \leftarrow DF_w + df_{w,D_\epsilon}, \forall w \in W_q$; // sum of df's
10: $B \leftarrow B \cup \{\epsilon\}$; // collect the nodes in B
11: **else**
12: push all child entries of ϵ to T;
13: **output** $(\{idf_{w,D,S_q} = \log \frac{D_S}{DF_w}, \forall w \in W_q\}, B)$;

of nodes fully covered by S_q during the traversal and it can facilitate the document retrieval step, as to be discussed next. The traversal repeats until the stack T is empty.

Lemma 1: When a node i with *zero* df_{w,D_i} value for each query keyword (i.e., $\forall w \in W_q, df_{w,D_i} = 0$), all the documents indexed beneath i are textually irrelevant to q. □

Proof. Suppose there is a document $d \in D_i$ that is textually relevant to q, i.e., $\exists w \in W_q, tf_{w,d} \times idf_{w,D,S_q} > 0$. Consequently, $tf_{w,d} > 0$ and hence df_{w,D_i} is at least one which contradicts our assumption to complete the proof. ■

B. Step Two: Top-k Document Retrieval

After buffer B containing candidate IR-tree nodes is returned by the IDF Calculation algorithm, Top-k Document Retrieval algorithm as the second step of the search runs to identify the result documents. As the candidate set might contain far more documents than k, this step tries to avoid examining non-result documents. Our strategy is to evaluate the documents based on their joint spatial and textual relevances with respect to a given query q and to terminate the process once the top-k result documents are obtained.

Algorithm 4 lists the pseudo-code of top-k document retrieval. It maintains a priority queue Q that orders the pending entries (either nodes or documents) in descending order of their relevance with respect to q (line 1-3). Based on idf_{w,D,S_q}, tf^{max}_{w,D_ϵ}, and the distance between ϵ and the query spatial scope S_q, the upper bound of the relevance value of documents d within a queue entry ϵ (if it is a node) to q can be estimated as follows: $\psi(\epsilon) = \alpha \cdot \sum_{w \in W_q} (tf^{max}_{w,\epsilon} \cdot idf_{w,D,S_q}) + (1 - \alpha)\frac{1}{dist(A_\epsilon, S_q)}$. Here, A_ϵ is the MBB corresponding to ϵ. We take the distance between the center of A_ϵ and the center of S as $dist(A_\epsilon, S_q)$. Notice that this formula also applies to individual documents to determine their exact relevance.

The search iteratively examines the head entry (ϵ,$\psi(\epsilon)$) in Q (line 4-15). If ϵ is a non-leaf node (line 14-15), its child nodes are all enqueued to Q for later processing. Notice that documents d not in any l_w should have their $tf_{w,d}$ equal to zero. Therefore, in case that ϵ is a leaf node (line 10-12), documents appearing in l_w ($\forall w \in W_q$) are enqueued to Q.

Algorithm 4 Top-k Document Retrieval

Input: a set of idf values, $\{idf_{w,D,S_q}, \forall w \in W_q\}$;
a candidate set, B; query keywords, W_q;
query spatial scope S_q;
a ratio between textual and spatial relevance, α;
the number of returned documents, k
Output: the k most relevant documents, R;
Procedure:

1: **MACRO**: $\psi(\epsilon) = \alpha \cdot \sum_{w \in W_q} (tf^{max}_{w,\epsilon} \cdot idf_{w,D,S_q}) + \frac{(1-\alpha)}{dist(A_\epsilon, S_q)}$;
2: **for** each entry $\epsilon \in B$ **do**
3: enqueue $(\epsilon, \psi(\epsilon))$ to Q; // initialize Q with entries in B
4: **while** Q is not empty **do**
5: dequeue an entry ϵ from Q;
6: **if** ϵ is a document **then**
7: $R \leftarrow R \cup \{\epsilon\}$;
8: **if** $|R| = k$ **then**
9: **goto** 16;
10: **else if** ϵ is a leaf node **then**
11: **for** each document d in any ϵ's inverted list l_w, $\forall w \in W_q$ **do**
12: enqueue $(d, \psi(d))$ to Q;
13: **else**
14: **for** each child c of ϵ **do**
15: enqueue $(c, \psi(c))$ to Q;
16: **output** R;

Here, the threshold algorithm and its variants [7], [10] can be adopted to improve the performance of reading entries from inverted lists. Finally, if ϵ is a document (6-7), it is directly put into a result set R as ϵ has the greatest relevance among all in Q. Once R contains k documents or no more documents can be found as implied by an empty priority queue, the algorithm stops and outputs R.

C. Discussion

In order to illustrate our top-k search algorithm, Example 3 describes how it operates on our running example.

Example 3: Continue Example 1. We consider TF-IDF and assume the top-1 document is requested (i.e., $k = 1$). The search is based on the IR-tree shown in Figure 4(a). Initially, those entries covered by Boston are visited. The trace is shown in Figure 6(a), with $B = \{i_2, i_3, i_4, N_b\}$. The counters D, D_{sushi}, and D_{buffet} are 6, 2 and 5, respectively, and the IDFs of 'sushi' and 'buffet' are $\log(6/2) = 0.477$ and $\log(6/5) = 0.079$, respectively.

Entry	Stack (T)	Buffer (B)
	$root$	$\emptyset$
$root$	N_b, N_c	$\emptyset$
N_b	N_c	N_b
N_c	i_2, i_3, i_4, i_8	N_b
i_2	i_3, i_4, i_8	N_b, i_2
i_3	i_4, i_8	N_b, i_2, i_3
i_4	i_8	N_b, i_2, i_3, i_4
i_8		N_b, i_2, i_3, i_4

(a) Trace of candidate selection

Entry	Priority queue (Q)
	N_b, i_3, i_2, i_4
N_b	i_6, i_3, i_5, i_2, i_4
i_6	d_6, i_3, i_5, i_2, i_4
d_6	i_3, i_5, i_2, i_4

(b) Trace of ranking

Fig. 6. Trace of candidate selection and ranking

Thereafter, ranking starts. The priority queue, Q, is initialized with entries i_2, i_3, i_4 and N_b. N_b, with the largest TF-IDF, is explored first. Its child entries (i.e., i_5 and i_6) that contain

at least one query word are put back to Q. Next, i_6 is retrieved and the corresponding document (i.e., d_6) is put into Q. Then, d_6 with the largest TF-IDF is dequeued and inserted into the result set R to complete this query. Figure 6(b) summarizes all the steps. □

As illustrated by the example, the proposed top-k document search algorithm based on IR-tree has five advantages over existing geographic document search algorithms. First, it performs spatial filtering and textual filtering simultaneously to discard as early as possible branches that are out of the query spatial scope and/or branches that do not contain any textually relevant documents. Second, it postpones the expensive calculation of the relevance between each document d and the query q until d is known to have a high chance to be included into the final result set. Third, it terminates once the top-k documents with highest relevances are identified. Fourth, it can support variations of textual and spatial relevance measures and different weights in combining the two relevances. Last but not the least, it enables an early detection of queries that return empty result sets.

V. Performance Evaluation

In this section, we evaluate the performance of IR-tree (labeled as IR-tree in our discussion) through both cost analysis and simulations. In Section V-A, we derive a cost model to analyze storage overhead and index I/O cost incurred for IR-tree. Based on the model, we observe several critical performance factors and validate these observations through experiments using synthetic document sets. Then, in Section V-B, we used two sample document sets, namely, newspaper clips from LA Times (LATimes'94) [20] and news archives from Dow Jones Factiva (Factiva) [6][6] to evaluate the search performance of IR-tree in comparison with two existing state-of-the-art approaches, including the hybrid index Hybrid_R that puts an R-tree upon inverted files [26] and KR*-tree (labeled as KR*-tree), as reviewed in Section II-C. Note that we hire Hybrid_R as the representative hybrid index as it performs much better than Hybrid_I.

A. Performance Analysis

In the first place, we present a cost model for IR-tree to analyze its storage overhead and I/O cost for a document set D with documents uniformly distributed in $|L|$ locations and in total having $|W|$ words. In this analysis, we assume the fanout of an IR-tree index is a constant f. We further use synthetic dataset to validate the observations made from the cost model and to compare IR-tree with Hybrid_R and KR*-tree.

1) Cost Model: Firstly, we analyze the storage overhead and I/O cost for IR-tree. As the search time is highly dependent on the I/O cost, we do not include the search time analysis due to space limitation.

Storage Overhead. For IR-tree, its storage is contributed by three components: (i) inverted files denoted as S_{inv}, (ii) tree hierarchy denoted as S_{inv}, and (iii) document summaries denoted as S_{ds}. The total storage for IR-tree, $S_{IR-tree}$, can be estimated as follows.

$$S_{IR-tree} = S_{inv} + S_{tree} + S_{ds} \tag{5}$$

Correspondingly, S_{inv} involves $|L|$ inverted files, each consisting of $O(|W|)$ inverted lists. As each document is mapped to one location, each list has $O(|D|/|L|)$ TFs and document IDs. Hence, $S_{inv} = O(|W| \cdot |D|)$. Next, for $|L|$ locations, the height of an IR-tree is $\log_f |L|$. With the root is at level 0, the number of IR-tree nodes, i.e., S_{tree}, is $O(\sum_{l=0}^{\log_f |L|-1} f^l)$. Furthermore, while each node is associated with one document summary, the storage for all document summaries S_{ds} is $O(|W| \cdot \sum_{l=0}^{\log_f |L|-1} f^l)$. Elaborating Equation (5), we obtain a more detailed estimation as in Equation (6), assuming $|L| < |D|$ and $|L| < |W|$.

$$\begin{aligned} & S_{IR-tree} \\ = \;& O(|W| \cdot |D|) + O(\textstyle\sum_{l=0}^{\log_f |L|-1} f^l) + \\ & O(|W| \cdot \textstyle\sum_{l=0}^{\log_f |L|-1} f^l) \\ = \;& O(|W| \cdot |D|) + O(\tfrac{f^{\log_f |L|}-1}{f-1}) + O(|W| \cdot \tfrac{f^{\log_f |L|}-1}{f-1}) \\ \approx \;& O(|W| \cdot |D|) + O(|L|) + O(|W| \cdot |L|) \\ \approx \;& O(|W| \cdot |D|) \end{aligned} \tag{6}$$

From the equation, we can see the storage overhead for IR-tree is dominated by that of inverted files. Also, $|W|$ usually does not increase as $|D|$ grows. For example, LATimes'94 and Factiva have different numbers of documents but they both have similar number of document words in the corpus. Thus, we can consider storage is linear to $|D|$. Following the same idea, the storage overheads for Hybrid_R and KR*-tree can be derived and they produce similar asymptotic storage costs.

Index I/O Cost. There are three types of accesses on IR-tree that constitute the I/O cost, including, (i) node traversal to visit nodes that may contain qualified documents, denoted by IO_{tree}; (ii) lookup of document summaries to get the documents' TF and DF statistics with respect to query keywords, denoted by IO_{ds}; and (iii) accesses of inverted files associated with the leaf nodes in order to evaluate the relevance of candidate documents to the query, denoted by IO_{inv}. Due to the small size of node IDs and relevance scores for IR-tree nodes and candidate documents, we consider that buffers and priority queues used in the rank-based search algorithm are small enough to be retained in main memory. The accesses of those are assumed to incur zero I/O cost. Thus, the I/O cost for IR-tree $IO_{IR-tree}$ can be expressed in Equation (7).

$$IO_{IR-tree} \;=\; IO_{tree} + IO_{ds} + IO_{inv} \tag{7}$$

To facilitate our discussion, we consider a query with $|W_q|$ query keywords, a query spatial scope L_q, requesting for k documents. We assume that all document locations are disjointed and meanwhile of equal size (i.e., $A/|L|$), with A being the total area of the search space. A fraction $\frac{|L_q|}{A}$ of an IR-tree tree hierarchy is approximately accessed.[7] In total, there are $O(\frac{|L_q|}{A} \cdot \sum_{h=0}^{log_f |L|-1} f^h)$ nodes accessed and this

[6]Notice that entire Factiva is a very large document set and we only randomly selected around 4×10^5 documents; whereas we use a complete LATimes'94 document set.

[7]We use $|L_q|$ to represent the size of query spatial scope L_q.

contributes to IO_{tree}. For A being a constant, IO_{tree} involves $O(|L_q| \cdot \sum_{h=0}^{log_f|L|-1} f^h)$. Next, with keyword-based storage scheme, only $|W_q|$ DF/TF value blocks are loaded. Each block has entries with respect to each IR-tree node, and thus IO_{ds} takes $O(|W_q| \cdot \sum_{h=0}^{log_f|L|-1} f^h)$ pages. Finally, $O(|L_q|)$ inverted files are accessed. In other words, $O(|W_q| \cdot |L_q|)$ inverted lists are accessed, with each having $|D|/|L|$ TF and DF values examined. Putting all of them together, we obtain the total I/O cost as in Equation (8).

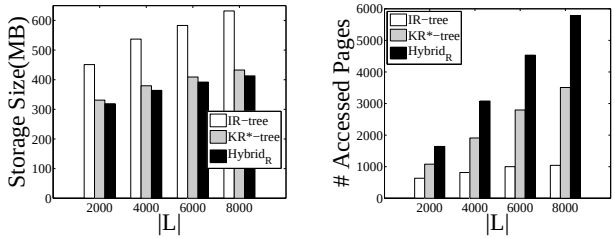
$$\begin{aligned} & IO_{IR-tree} \\ = \; & O(|L_q| \cdot \textstyle\sum_{h=0}^{log_f|L|-1} f^h) + \\ & O(|W_q| \cdot \textstyle\sum_{h=0}^{log_f|L|-1} f^h) + O(|W_q| \cdot |L_q| \cdot |D|/|L|) \\ \approx \; & O(|L_q| \cdot |L|) + O(|W_q| \cdot |L|) + O(|W_q| \cdot |L_q| \cdot |D|/|L|) \end{aligned} \quad (8)$$

Equation (8) well indicates the performance factors. First, the tree traversal cost (i.e., the first term) is mainly dependent on the size of a query spatial scope L_q and the size of an IR-tree which in turn is affected by $|L|$. Second, the overhead of document summary lookups (i.e., the second term) is contributed by the number of queried words $|W_q|$ and the size of an IR-tree. Finally, the access of inverted files (i.e., the third term) is based on $|D|$, $|L|$, $|L_q|$ and $|W_q|$. Notice that this equation does not take k, the number of requested documents, into account. Hence, Equation (8) simply reflects the (worst case) condition that k is very large and/or all result documents are sparsely distributed in different leaf nodes. Of course, when k is small, the search efficiency improves since many nodes and their indexed inverted files are skipped from detailed examinations. On the contrary, Hybrid$_R$ and KR*-tree produce the same asymptotic I/O costs. However, without incorporating ranking into the search as IR-tree, both of them always incur the worst case I/O costs as stated in Equation (8). This observation is consistent with our experiments to be presented next.

2) Validation using Synthetic Documents: Here, we adopt synthetic document sets to validate the observations made from the cost models on IR-tree. As for comparison, we include Hybrid$_R$ and KR*-tree. Since $|W|$ is almost constant for a document set, in our evaluation, we study the factors $|L|$ and $|D|$ only.

The synthetic document sets are generated as follows. First, the document locations distribute uniformly over the search space and we assume each location contains the same number of documents. Second, we assume there are $50,000$ words in total and each document contains 500 words. Regarding the size of locations, queries and other settings, we follow our simulation settings that will be detailed in the next subsection. In the following, we first examine the impact of $|L|$ by changing $|L|$ from 2000 to 8000, with $|D|$ fixed at $100,000$. Then, we evaluate the impact of $|D|$ varying from 100,000 to 1,000,000 while $|L|$ is defaulted at 1000.

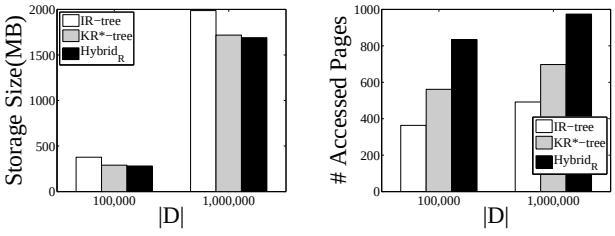
Effect of $|L|$. First, we evaluate the performance of different approaches against the number of locations $|L|$. The evaluation results in terms of storage cost and I/O cost are depicted in Figure 7. In Figure 7(a), the storage incurred by IR-tree grows linearly with $|L|$, consistent to the observation made on Equation (6). The same trends for Hybrid$_R$ and KR*-tree are also observed. In Figure 7(b), similar linear trends are observed for all the three approaches. Compared with the other two, IR-tree performs the best as its cost increases slightly with $|L|$. Here, Hybrid$_R$ produces the smallest indices but it incurs the largest I/O cost. IR-tree and KR*-tree produce indices of similar size but IR-tree incurs less I/O than KR*-tree due to the efficient rank-based search algorithm facilitated by document summaries.



(a) Storage Overhead (b) I/O Cost

Fig. 7. Storage overhead and I/O cost vs. $|L|$ ($|S|$ = 100km $\times$100km, $k = 100$, $\alpha = 0.5$, $|D| = 100,000$)

Effect of $|D|$. Next, we evaluate the impact of the document set size ($|D|$) on storage overhead and index I/O cost. Figure 8 shows the experiment results against $|D|$ with $|L|$ fixed at 1000. It is clear that all approaches follow a linear trend with $|D|$, consistent to the behavior determined in our cost model. Meanwhile, we can also see a significant difference in terms of magnitudes among all those approaches. This can be analyzed and explained by the above discussed reasons.



(a) Storage Overhead (b) I/O Cost

Fig. 8. Storage overhead and I/O cost vs. $|D|$ ($|S|$ = 100km $\times$100km, $k = 100$, $\alpha = 0.5$, $|L| = 1000$)

B. Simulations Based on Real Document Sets

Next, we examine the performance of IR-tree on two real document sets, namely, LATimes'94 and Factiva. In what follows, we first present the experiment setup. Then, we discuss the evaluation details and experiment results in terms of search time and search I/O cost against different query parameters. Finally, we present the storage cost incurred for different approaches.

1) Experiment Setup: To prepare the sample documents for experiments, we extract location names from all individual documents and then geo-code the location names into spatial regions, following a commonly used practice (e.g., [5], [26]). Our geo-coder is developed based on a proprietary geographic ontology that covers about $129,784$ worldwide locations [19], and it employs the focus-detecting algorithm [1] to locate *one* location for each document. As different locations cover different granularity, we associate each location with a *type*

(e.g., state, super large city) according to the government or administration types and the population sizes, and assume the locations of the same type cover the regions of the same size. In this evaluation, six types of locations are defined and their sizes are shown in Table II. The overall search space is roughly set to $3000km \times 3000km$. Take a document d_i that is geo-coded to Chicago city as an example. The spatial scope of d_i (i.e., L_{d_i}) is set to a square centered at $l_{Chicago}$ with the side length set to 200km. Here, $l_{Chicago}$ is the point location that represents the center of Chicago, which is pre-defined by the internal geographic ontology. Finally, Table III summarizes the properties of two sample document sets.

TABLE II
LOCATION TYPE AND AREA SIZE

Location type	Area size
State	500km × 500km
Super large city	200km × 200km
Large city	100km × 100km
Medium city	80km × 80km
Small city	50km × 50km
County	20km × 20km

TABLE III
PROPERTIES OF SAMPLE DOCUMENT SETS

	LATimes'94	Factiva
No. of indexed documents	110,273	380,760
Average no. of words per doc	504	522
No. of indexed document words	90,986	103,286
Total size (MB)	421	2560
No. of locations	2119	4007

We implement IR-tree and store the index on disk using keyword based storage scheme, denoted by IR-tree. In addition, we implement the hybrid index Hybrid_R that presents an R-tree on top of inverted files and KR*-tree (denoted as KR*-tree) for comparison. Notice that since the original KR*-tree can only support search for documents that contain *all* query keywords, we improve its query processing algorithm to handle the search for documents that can contain *some* query keywords, for comparison. The improved algorithm keeps all the candidate nodes with each containing at least *one* of the query keywords in the node selection process and then retrieves all the documents from these candidate nodes for top-k document ranking. In our experiment, we fix the disk page size at 1KB, use 4-byte integers to represent a document ID, a node ID, a TF value (i.e., $tf_{w,D}^{max}$ or $tf_{w,d}$), a DF value (i.e., $idf_{w,D,S}$), a pointer individually, and use two 4-byte floating points to store coordinates in a two-dimensional space. All the algorithms were implemented in Microsoft Visual C++, and all the experiments were conducted on Intel Xeon 2.0GHz computers equipped with 8GB main memory running Microsoft Windows Server 2003.

In the following, we present the results of two sets of experiments. The first set is to examine the search performance of IR-tree against that of others in terms of average search time and average I/O cost. The former is the average duration between the time the query is issued to the time all of the result documents are identified over all of the evaluated queries, and the latter stands for the average number of index pages accessed by each query. Since the query time and I/O cost of retrieving the result documents is independent on the algorithms adopted, they are excluded from our simulation results. Each experiment runs 100 random queries with each query containing up to four keywords related to finance, politics, travel and food. Each query on average has 1.91 keywords, and $11,174$ documents of Factiva data set, and $10,437$ documents of Latimes'94 data set are textually relevant to a query on average. All the queries are run independently such that no query results and intermediate states would be left in the system to benefit subsequent ones. The second set of experiments is to examine the index construction cost and the storage overhead of IR-tree in comparison with that of others.

2) Search Efficiency: In the first set of experiments, we investigate the impacts of three factors on the search performance. They are (i) the size of the query spatial scope $|S|$; (ii) the number of requested documents k; and (iii) the relative importance of textual relevance α to spatial relevance. Their settings are shown in Table IV in which the underlined values are the default settings. In each experiment, we only vary one parameter value while fixing the others at their defaults, unless explicitly stated. We do not report the performance of Hybrid_I as it performs much worse than Hybrid_R for all of the cases. As an example, for Factiva, when $k = 100$, $\alpha = 0.5$ and $|S|$ varies from 10^2 to 500^2, the average search time and I/O cost incurred for Hybrid_I is shown in Table V. Comparing with Fig 10, we can easily find out that Hybrid_R outperforms Hybrid_I significantly. The similar results are observed for the rest experiments. The reason behind is that to compute TF-IDF we have to find all of the documents located inside a query spatial scope in order to decide the *idf* value (as defined in Equation (2)). As Hybrid_I maintains R-trees under inverted files, all of the documents that contain any of the query keywords have to be accessed while the majority of them are located outside the query spatial scope.

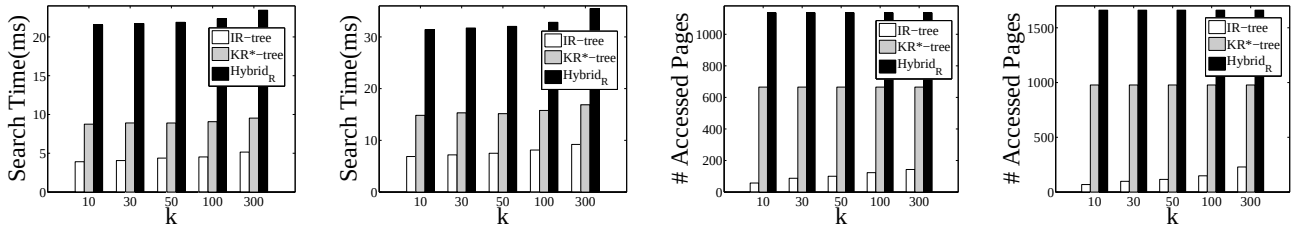
TABLE IV
EXPERIMENT PARAMETERS

Parameters	Values
Spatial scope size ($\|S\|$)	10^2, 20^2, $\underline{100^2}$, 500^2(unit: km^2)
Request no. of documents (k)	10, 30, 50, $\underline{100}$, 300
The rate of textual rel. (α)	0, 0.25, $\underline{0.5}$, 0.75, and 1

TABLE V
SEARCH PERFORMANCE VS. $|S|$ FOR HYBRID_I ($k = 100$, $\alpha = 0.5$)

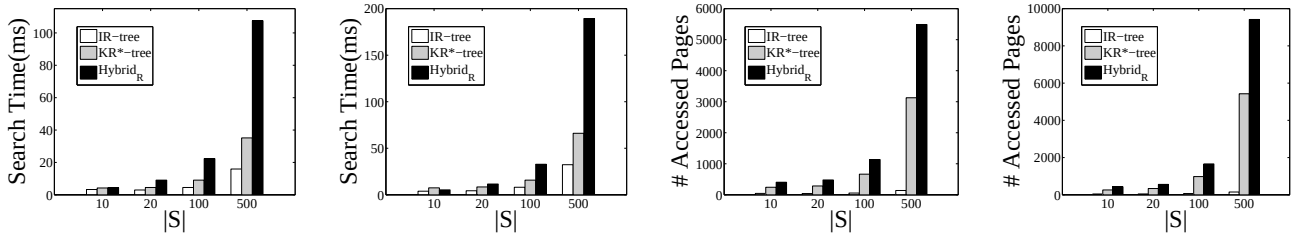
$\|S\|$	Search Time(MS)	I/O Cost (# Page Accessed)
10^2	622.8	904.2
20^2	723.8	1172.4
100^2	809.8	3927.5
500^2	944.5	9657.1

Effect of k. First, we vary the requested number of document, k, from 10 to 300 while $|S|$ is fixed at 100km ×100km and α is fixed at 0.5. Figure 9 shows the experimental results of all three approaches in terms of the average search time and average I/O cost against k. The first finding is that Hybrid_R performs the worst while IRtree performs the best in all the cases. Consider the search time. KR*-tree takes around 48%



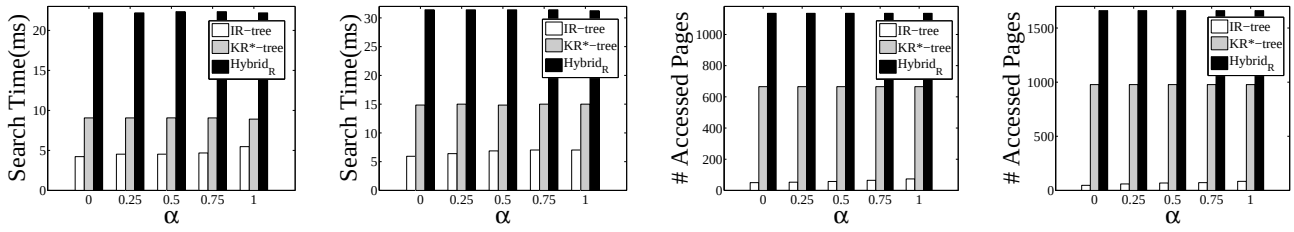
(a) LAtimes'94-Time (b) Factiva-Time (c) LAtimes'94-I/O (d) Factiva-I/O

Fig. 9. Search performance vs. k ($|S| = 100$km $\times 100$km, $\alpha = 0.5$)



(a) LAtimes'94-Time (b) Factiva-Time (c) LAtimes'94-I/O (d) Factiva-I/O

Fig. 10. Search performance vs. $|S|$ ($k = 100$, $\alpha = 0.5$)



(a) LAtimes'94-Time (b) Factiva-Time (c) LAtimes'94-I/O (d) Factiva-I/O

Fig. 11. Search performance vs. α ($|S| = 100$km $\times 100$km, $k = 100$)

of Hybrid_R's search time while IRtree takes around 20% of Hybrid_R's search time under Factiva. For LATimes'94, IR-tree takes around 20% of Hybrid_R's search time and KR*-tree takes around 41% of Hybrid_R's search time.

The second finding is that the I/O cost of both Hybrid_R and KR*-tree is not affected by k while that of IR-tree increases as k increases. This is because Hybrid_R evaluates all of the documents that are spatially and textually relevant to the query, regardless of k. Though KR*-tree only explores the documents in the leaf nodes which are spatially and textually relevant to the query, it still needs to retrieve all the documents in those nodes first to get the top-k documents. However, IR-tree adopts a ranking-based document retrieval. It evaluates the documents based on the likelihood that a document will be included into the final result, and documents which are more likely to become result documents will be evaluated earlier. The advantage of rank-based document retrieval over blind evaluation becomes more significant when the difference between k and the cardinality of the candidate set is larger. As an illustration, when $|S| = 100$km $\times$ 100km, Hybrid_R retrieves 440 documents and KR*-tree evaluates 222 documents for all the k settings under Factiva data set. On the other hand, IR-tree evaluates 182 documents when $k = 30$ and 203 documents when $k = 100$. Overall, IR-tree performs much better than both Hybrid_R and KR*-tree in terms of I/O cost.

Effect of Query Scope Size($|S|$). Next, we evaluate the performance of the approaches under different query spatial scope sizes $|S|$ (ranging from 10km $\times$10km to 500km $\times$500km), with k fixed at 100 and α fixed at 0.5. The number of documents that are spatially relevant to S is listed in Table VI, and the experiment result is plotted in Figure 10.

TABLE VI
SPATIAL SELECTIVITY

$\lvert S\rvert$	LATimes'94	Factiva
$10km \times 10km$	819.5	1872.13
$20km \times 20km$	946.66	2458.17
$100km \times 100km$	2267.86	4808.12
$500km \times 500km$	8045.85	21334.3

In general, IR-tree performs the best while Hybrid_R does the worst, i.e., the performance trend is consistent with that under various k values. It is observed that the superiority of IR-tree over Hybrid_R becomes more significant when $|S|$ increases. In the figure, IR-tree incurs 40% of Hybrid_R's search time when $|S| = 20$km $\times$20km, but only 17% of Hybrid_R's search time when $|S| = 500$km $\times$500km. This is because the top-k search algorithm adopted in IR-tree only accesses some, but not all, of the candidate documents. As $|S|$ becomes larger, more documents are covered by queries, resulting in larger candidate sets. Consequently, Hybrid_R suffers from the exhaustive scans of every single candidate document independent of k. Consider the results for Factiva. When $|S| = 100$km $\times$100km and $k = 100$, Hybrid_R and KR*-tree evaluate 440 and 222 documents, respectively, while IR-tree evaluates 203 documents. When $|S| = 500$km $\times$500km and $k = 100$, Hybrid_R and KR*-tree

evaluate $2,210$ and 960 documents, respectively, while IR-tree evaluates 601 documents.

Effect of α. Third, we evaluate the impact of α with k fixed at 100 and $|S|$ fixed at 100km $\times 100$km. The result is shown in Figure 11. Again, IR-tree outperforms both Hybrid_R and KR*-tree significantly under all cases. Since both Hybrid_R and KR*-tree need to rank all of the candidate documents before determining the top-k relevant documents, their performances do not change much under different α values. However, the performance of IR-tree drops slightly as α increases. Notice that even when only textual relevance, but not spatial relevance, is considered (i.e., $\alpha = 1$), IR-tree still performs much better than others, which indicates that the document summaries do provide good guidance of document retrieval based on textual relevance.

3) Index Construction Cost and Storage Overhead: In the last set of experiments, we evaluate the construction cost and storage overhead of different index structures, with results presented in Table VII and Table VIII. IR-tree takes a bit longer than KR*-tree because it consumes more time to aggregate the information of DF and TF. In terms of storage overhead, for KR*-tree and IR-tree, we separate the document summary from an R-tree to show the storage overheads of the different components of the index. Notice that IR-tree requires around 25% extra space compared with the hybrid approaches, since it maintains the summary information of DF and TF values of different keywords in the internal nodes. On the other hand, as KR*-tree only stores the IDs of the tree nodes for each query keyword, it needs less storages than IR-tree. Given the requirement of high search performance in search engines and the fact that IR-tree significantly improves the search performance as shown in the previous set of experiments, the extra storage overhead is well paid off.

TABLE VII
INDEX CONSTRUCTION COST (UNIT: HOUR)

	LAtimes'94	Factiva
Hybrid_R	3.25	8.3
KR*-tree	3.35	8.4
IR-tree	4.55	8.6

TABLE VIII
INDEX STORAGE (UNIT: MB)

	LAtimes'94		Factiva	
	Index	Addi. Overhead	Index	Addi. Overhead
Hybrid_R	272	0	901	0
KR*-tree	272	8.44(KR*-list)	901	19(KR*-list)
IR-tree	272	83.1(Doc. Sum.)	901	196(Doc. Sum.)

The cost of adding/deleting documents in the IR-tree is composed of three parts: (i)adding/deleting the documents from the inverted files; (ii)spliting/deleting nodes in the R-tree when node overflow or node underfull is happened; (iii)updating the document summaries in the nodes. The first and second part are the same as the updating cost of inverted files or R-tree. For the third part, since we update the document summaries of all the nodes on the path from the leaf node to the root, its cost is linear with the hight of R-tree, i.e., $O(log_f|L|)$. Due to the space limitation, we ignore the detailed experiment results of the update cost for IR-tree.

VI. CONCLUSION

In this paper, we focus on the efficiency issue of geographic document search. We propose an efficient indexing structure, namely, IR-tree, along with a top-k document search algorithm for geographic document search.

We are prototyping IR-tree as a public geographic document search facility and building a testbed based on IR-tree for future research. We also plan to further enhance the IR-tree index based on various access patterns.

REFERENCES

[1] E. Amitay, N. Har'El, R. Sivan, and A. Soffer. Web-a-where: Geotagging Web Content. In *SIGIR'04*, pages 273–280, 2004.

[2] V. N. Anh, O. de Kretser, and A. Moffat. Vector-Space Ranking with Effective Early Termination. In *SIGIR'01*, pages 35–42, 2001.

[3] V. N. Anh and A. Moffat. Pruned Query Evaluation using Pre-Computed Impacts. In *SIGIR'06*, pages 372–379, 2006.

[4] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.

[5] Y.-Y. Chen, T. Suel, and A. Markowetz. Efficient Query Processing in Geographic Web Search Engines. In *SIGMOD'06*, pages 277–288, 2006.

[6] Dow Jones Factiva. `http://www.factiva.com`.

[7] R. Fagin, A. Lotem, and M. N. Optimal. Optimal aggregation algorithms for middleware. In *PODS'01*, pages 102–113, 2001.

[8] I. D. Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *ICDE'08*, pages 656–665, 2008.

[9] V. Gaede and O. Günther. Multidimensional Access Methods. *ACM Computing Survey*, 30(2):170–231, 1998.

[10] U. Güntzer, W.-T. Balke, and W. Kiessling. Optimizing multi-feature queries for image databases. In *VLDB'00*, pages 419–428, 2000.

[11] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD'84*, 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'07*, pages 16–25, 2007.

[13] D. Hiemstra. A Probabilistic Justification for Using TF x IDF Term Weighting in Information Retrieval. *Int. J. on Digital Libraries*, 3(2):131–139, 2000.

[14] G. R. Hjaltason and H. Samet. Distance Browsing in Spatial Databases. *TODS*, 24(2):265–318, 1999.

[15] C. B. Jones, A. I. Abdelmoty, D. Finch, G. Fu, and S. Vaid. The SPIRIT Spatial Search Engine: Architecture, Ontologies and Spatial Indexing. In *GIS'04*, pages 125–139, 2004.

[16] K. S. Jones. A Statistical Interpretation of Term Specificity and Its Application in Retrieval. *Journal of Documentation*, 28(1):11–21, 1972.

[17] I. Lazaridis and S. Mehrotra. Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure. In *SIGMOD'01*, pages 401–412, 2001.

[18] R. Lee, H. Shiina, H. Takakura, Y. J. Kwon, and Y. Kambayashi. Optimization of Geographic Area to a Web Page for Two-Dimensional Range Query Processing. In *WISEW'03*, pages 9–17, 2003.

[19] Z. Li, C. Wang, X. Xie, X. Wang, and W.-Y. Ma. Indexing implicit locations for geographical information retrieval. In *GIR'06*, 2006.

[20] Los Angeles Times. `http://www.latimes.com`.

[21] A. Markowetz, Y.-Y. Chen, T. Suel, X. Long, and B. Seeger. Design and Implementation of a Geographic Search Engine. In *WebDB*, pages 19–24, 2005.

[22] K. S. McCurley. Geospatial Mapping and Navigation of the Web. In *WWW'01*, pages 221–229, 2001.

[23] A. Ntoulas and J. Cho. Pruning Policies for Two-Tiered Inverted Index with Correctness Guarantee. In *SIGIR'07*, pages 191–198, 2007.

[24] G. Salton and C. Buckley. Term-Weighting Approaches in Automatic Text Retrieval. *IPM*, 24(5):513–523, 1988.

[25] S. Shekhar, S. Chawla, S. Ravada, A. Fetterer, X. Liu, and C.-T. Lu. Spatial Databases - Accomplishments and Research Needs. *TKDE*, 11(1):45–55, 1999.

[26] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma. Hybrid Index Structures for Location-Based Web Search. In *CIKM'05*, pages 155–162, 2005.