Processing Spatial-Keyword (SK) Queries in Geographic Information Retrieval (GIR) Systems

Ramaswamy Hariharan, Bijit Hore, Chen Li, Sharad Mehrotra
Donald Bren School of Information and Computer Sciences
University of California, Irvine
{rharihar, bhore, chenli, sharad}@ics.uci.edu

Abstract

Location-based information contained in publicly available GIS databases is invaluable for many applications such as disaster response, national infrastructure protection, crime analysis, and numerous others. The information entities of such databases have both spatial and textual descriptions. Likewise, queries issued to the databases also contain spatial and textual components, for example, "Find shelters with emergency medical facilities in Orange County," or "Find earthquake-prone zones in Southern California." We refer to such queries as spatial-keyword queries or SK queries for short. In recent times, a lot of interest has been generated in efficient processing of SK queries for a variety of applications from Web-search to GIS decision support systems. We refer to systems built for enabling such applications as Geographic Information Retrieval (GIR) Systems. An example GIR system that we address in this paper is a search engine built on top of hundreds of thousands of publicly available GIS databases. Building a search engine over such large repositories is a challenge. One of the key aspects of such a search engine is the performance. In this paper, we propose a framework for GIR systems and focus on indexing strategies that can process SK queries efficiently. We show through experiments that our indexing strategies lead to significant improvement in efficiency of answering SK queries over existing techniques.

1. Introduction

Publicly available GIS databases contain vital location-based information and play an important role in many applications such as disaster management, national infrastructure protection, crime analysis, etc. Such information has played a significant role during national disasters like *9/11* or more recent *Hurricane Katrina*, in saving lives, resources and properties. Federal and state government attempts (e.g., www.geodata.gov, www.fgdc.gov) [7] to make GIS databases accessible at one place underline the urgent need to make such location-based information readily available to a large number of organizations who deal with such data on a day-to-day basis. Consider the following example queries:

- Find shelters with emergency medical facilities in Orange County.
- Find earthquake-prone zones in Southern California.

The information that above queries seek is critical for decision makers and first-responders during emergency operations. Such information is highly geographic in nature and contained in GIS databases explicitly created for such purposes. To the best of our knowledge there are no search engines that can provide such information.

Recently, there has been lot of commercial interest in local (or location-based) information such as restaurants, businesses, entertainment centers, etc. As a consequence, popular search engines such as Google, MSN, and Yahoo [16, 21, 9] have extended their text searching capabilities to provide local information. Our application domain is quite different from the application domain that these search engines are meant for. They make heavy use of local business directories like yellowpages and geographic references in Web pages that are quite different from the kind of information that publicly available GIS databases contain. In fact, all these popular search engines failed to return relevant results in response to such queries. For example, when we searched in Google for "earthquake zones in Irvine, CA," it returned lots of irrelevant Web pages as top results.

The underlying location-based information entities present in GIS databases fundamentally comprise of two components: 1) spatial or location information and 2) textual information. Spatial information refers to the geographic location, size and shape of an entity. The shape could be a point, or extended in space such as a line or a polygon. Textual information refers to free-form keywords describing the entity. The queries asked on location-based entities also contain spatial and textual components. We refer to them as spatio-keyword queries or *SK* queries for short. Hence, to handle *SK* queries we need both spatial and text processing techniques. We refer to retrieval of spatial and textual information as Geographic Information Retrieval or GIR for short.

Mapdex [14], a search engine that points to GIS databases on the Web, catalogs about 400,000 GIS databases. This number only constitutes a small percentage, and such databases on the Web are growing at a very fast rate. In order to build a search engine containing a large number of databases with billions of records, ensuring a small query response time is a key requirement. Given the modern search engine's sub-second query response time, the expectation will also be similar for GIS search engines, especially for time-critical applications such as disaster response. We address this very aspect in greater detail in this paper.

Recently, the academic community has started addressing this problem [3, 10, 11, 22] in the context of building geographic search engine for Web pages. These search engines extend text capabilities to include the geographic information in Web pages. Even though our application is different, some of the indexing techniques proposed in literature can be adapted to building search engine for GIS databases. In order to process *SK* queries, we need to build index structures that can handle both spatial and textual information.

All work till now uses separate index structures or some naive form of combined index for processing spatial and textual data, where the performance is the main bottleneck. Our main focus in this paper is to develop a hybrid index structure that aims to reduce this performance bottleneck by combining both spatial and textual information in a meaningful way. Given that most of the textual queries contain a set of keywords and the relevant results retrieved contain all of the keywords in them, we consider *SK* queries with *AND* semantics similar to conjunctive keyword queries for the textual part. In this paper, we propose efficient indexing strategies that process *SK* queries with *AND* semantics.

We make the following contributions in this paper:

- We propose a framework for query processing in Geographic Information Retrieval (GIR) Systems.
- We develop a novel indexing structure called *KR*-tree* that captures the joint distribution of keywords in space and significantly improves performance over existing index structures.
- We have conducted experiments on real GIS datasets showing the effectiveness of our techniques compared to the existing solutions.

The remainder of the paper is organized as follows. In Section 2, we describe a framework for GIR systems and explain how different application domains fit within the framework. In Section 3, we formally define *SK* queries with *AND* semantics. In Section 4, we explain the existing indexing strategies proposed for *SK* queries followed by discussion on their limitations. In Section 5, we describe *KR*-tree* in detail with an example. In Section 6, we discuss experimental results performed on real GIS datasets comparing our techniques with others. In Section 7, we briefly describe the related work. We conclude our paper and outline open problems for future work in Section 8.

2. Framework for GIR Systems

In this section, we first describe a framework for GIR system, and how our indexing strategies can fit well within the framework in the context of different applications. The framework shown in Figure 1 consists of four major components: 1) GIR database, 2) Indexer, 3) Ranker, and 4) Interface.

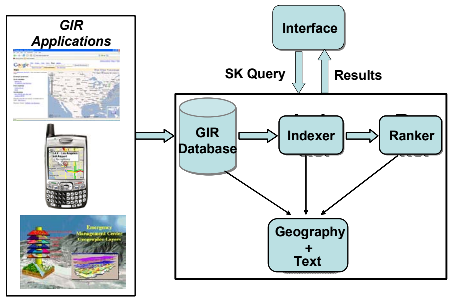

Figure 1. GIR System Framework

GIR Applications: Before describing the components of the framework, we briefly explain different GIR applications that can fit within the framework. The class of applications that take as input *SK* queries, process them and output information sources that are relevant to the queries are called as GIR applications. Geographic (or local) search engines, location-based services (LBS) and GIS search engines are various examples of GIR applications. Geographic search engines treat Web pages as information sources and process *SK* queries on top of them. GIS search engines process *SK* queries on top of GIS databases. LBS applications provide services based on the mobility of users. They process *SK* queries on top of some specialized GIS databases mostly containing local information. Our framework can exploit both the explicit and implicit geographic references found in the information sources. For example, in unstructured sources such as Web pages, geographic references are implicitly found as place names, telephone numbers, addresses, etc., along with the other text matter. These implicit references are extracted and converted to metric measurements. In structured sources such as GIS databases, the geographic references are explicitly defined as spatial attributes along with non-spatial text attributes. The spatial attributes along with non-spatial text attributes constitute our GIR database. For example, any GIS database available in ESRI shapefile format [6] forms an explicit GIR database.

GIR Database: The input to a GIR database can come from unstructured as well as structured data sources. There are a number of pre-processing steps involved in converting a unstructured source to a GIR database. Such tasks are geo extraction, geo matching and geo propagation. Interested readers can refer to [15] for more details. Structured sources like GIS databases also involve pre-processing steps such as coordinate conversion of spatial attributes, identification and extraction of textual attributes, etc. In our experiments, we use structured data sources and explain the details of the pre-processing steps in Section 6.

Indexer: The primary search dimensions for processing *SK* queries are spatial and text attributes. Hence, the indexer in our framework builds data structures on spatial and textual attributes. In particular, we construct a hybrid index that combines both spatial and textual attributes. We can consider each keyword appearing in the text as having a spatial distribution. The spatial distribution of various keywords may be correlated in space. Our indexing technique exploits this spatial correlation of keywords, thereby capturing their joint distribution in space. Capturing this joint distribution significantly improves the retrieval of answers to *SK* queries. In Section 4, we describe our indexing mechanism in detail.

Ranker: The ranker combines the ranking functions of spatial and textual attributes. The goal of the ranking function is to assign score to objects in GIR database based on its relevance to the *SK* query. Given a *SK* query $Q = \{q_r, q_t\}$ [see Definition 1], the overall ranking function is

$$F_{sk} = \alpha_1 \cdot F_r(q_r, o_r) + \alpha_2 \cdot F_t(q_t, o_t),$$

where $F_r(q_r, o_r)$ is a geographic ranking function, and $F_t(q_t, o_t)$ is a keyword-based ranking function. α_1 and α_2 are suitable weights where $\alpha_1 + \alpha_2 = 1$. The common measure for F_t is the cosine measure that computes the relevance of an object to the query (based only on their corresponding textual parts) using the $tf - idf$ model [18]. The ranking function F_r considers geographic relationships such as $contain$, $overlap$, $inside$, etc., between the spatial parts of an object and the query. Other sophisticated ranking mechanisms for geographic data are proposed in [1, 8] that consider geographic hierarchies.

Interface: The interface allows the user to enter *SK* queries using a map and a text interface. Search engines such as Google, MSN, and Yahoo use similar interfaces. Each *SK* query consists of a set of textual keywords and a geographic region of interest specified as a query rectangle. We formally define *SK* query in Definition 1. The interface presents a ranked results to each query.

3. Preliminaries and Definitions

Let S be a GIR database with attributes $A = \{R, T\}$ and size N (number of objects). The domain of R is a rectangle. There are two components of a rectangle, $r = [(x_1, y_1), (x_2, y_2)]$ where $x_1, y_1, x_2, y_2 \in \Re$, lower-left and upper-right corners specified in Cartesian coordinates by (x_1, y_1) and (x_2, y_2), respectively. We take the minimum bounding rectangles (MBR) for line and polygon objects present in S. For point objects, the two corners of the rectangle are represented by the same point, that is $x_1 = x_2$ and $y_1 = y_2$. The value set of T is the bag of keywords $\{t_1, t_2, t_3, \ldots\}$. The domain of T is the set of words. Each object o in S is represented by the triplet $\langle o_{id}, o_r, o_t \rangle$, where o_{id} is the object id, o_r is the object's MBR, and o_t is a set of textual keywords describing the object.

Definition 1: A *SK* query is defined as $Q = \{q_r, q_t\}$, where $q_r \in \Re$ is the spatial part of the query specified as a minimum bounding rectangle (MBR), and $q_t \in T$ is a set of keywords in the query. A *SK* query with the *AND* semantics is defined as the one in which all the keywords in q_t are required to be present in the retrieved records.

Definition 2: The answer set to an *SK* query Q is defined as the set of objects $O_q = \{o_q^i\}$ in S, where each object's MBR has a non-empty intersection with query's MBR (i.e., $o_q^i \cap q_r \neq \phi$), and contains all of the query keywords in q_t.

4. Indexing Mechanisms

Existing indexing work in geographic information retrieval proposed can be broadly classified into two categories: 1) separate index for spatial and text attributes and 2) hybrid index that combine spatial and text attributes. Next, we briefly discuss the ideas of the techniques.

4.1 Separate Index for Spatial and Text Attributes

In this approach, separate index structures are built for spatial data and text data. The choice of index structure for spatial data can be grid, quadtree, or R*-tree. One commonly used structure is R*-tree [19], and the choice of other indices are also possible. For text, [18] proposed an inverted file index. The inverted file index stores for each keyword, a sorted list of object ids in which the keyword appears, its score, and frequency.

Using this approach, *SK* queries can be answered in two ways. First, a set of candidate object ids that satisfy the spatial part of the query are retrieved using the spatial index. The object ids are sorted and for each retrieved object id, the textual keywords of the query are looked up in its corresponding inverted list index. Finally, all the object ids that satisfy the query are collected, ranked and presented as sorted results to the user.

The second approach is to first filter object ids based on the query keywords. Inverted index list for each query keyword are looked up, and a set of object ids that are present in the intersection of the lists are passed to the next stage for spatial filtering. Finally, the scores for each object are computed by combining the ranking of textual and spatial parts. The performance of both approaches depends on the selectivity of the objects satisfying the text or spatial part of a query. If the number of objects in the spatial region of the query is small, it is better to do the spatial filtering first and vice versa. In [11], the choice for the spatial index is a grid and inverted file index for textual keywords. [3, 22] also discuss variants of this approach.

In [3], the authors propose a number of improved techniques to the above basic approaches. First, they suggest storing spatial data in the disk by following Hilbert curve ordering [6]. This ordering maintains the spatial closeness of objects thereby speeding up the disk access operations in retrieving the spatial data. Secondly, the objects in the inverted index list are assigned ids according to Hilbert ordering and sorted based on these ids. A grid-based structure is built in memory to store the ids of the spatial data in each tile of the grid. When a query is issued by the user, the relevant tiles of the grid that overlap the spatial region of the query are retrieved. The object ids contained in the tiles are sorted and looked up against the inverted indices. Since the object ids are arranged in the disk following Hilbert ordering, the relevant objects are retrieved using a small number of scans.

Advantages and Limitations: The main advantage of the above strategies is the ease of maintaining two separate indices. However, the main performance bottleneck lies in the number of candidate objects generated during the filtering stage. If spatial filtering is done first, many objects may lie within a query's spatial extent, but very few of them are relevant to the query keywords. This increases the disk access costs by generating a large number of candidate objects. The subsequent stage of keyword filtering becomes expensive. The same is true, if keyword filtering is done first. Moreover, the above strategies assume a memory resident spatial index which is not reasonable for large GIR databases. In [3], this issue is discussed by proposing to reduce the granularity of spatial index, so that it fits in main memory. However, if the grid is too coarse, it loses its pruning capabilities. We build a disk resident spatial index.

4.2 Hybrid Indices

Hybrid indexing techniques combine the spatial and inverted file indices. In [11], the inverted list was modified as the following. The list for each keyword was augmented with the space in which the objects contained in the list appear. For instance, if w_1 is the keyword, its list was augmented as: $w_1 = \{r_1(o_1, o_2, ...), r_2(o_2, ...), ...\}$ where $r_1, r_2, ..$ are bounding rectangles in space. When a query is issued, the corresponding keyword lists are loaded, and objects are filtered using the associated spatial index. This strategy still requires scanning the entire list.

The closest work to ours is the hybrid indexing structures proposed in [22]. The first hybrid data structure shown in Figure 2 is called *Inverted File-R*-tree*. In this structure, first an inverted index file is built. Then, the file is modified by building a R*-tree to the set of objects's MBR pointed to by each keyword in the file. The leaf node of R*-tree points to a page list of object ids whose entry contain the keyword and the MBR. This entry is called a *geo-keyword*. When a query is issued, the query keywords are filtered using the inverted index. Later, the R*-tree corresponding to each query keyword are used to filter the spatial part of the query. The intersection of object ids from the R*-trees produces the final answer set.

The second data structure proposed in [22] is called *R*-tree-Inverted File*. As shown in Figure 3, an R*-tree is first built for all the objects' MBR irrespective of keywords. An inverted index file is created for keywords that appear in the leaf node of the tree. Each keyword in this inverted index points to a page list of object ids whose entries contain both the MBR and the keyword, again referred to as geo-keyword. When a query is issued, a set of leaf nodes that intersect with the query rectangle is retrieved first. Then using the retrieved node's inverted file index, object ids satisfying the query keywords are obtained.

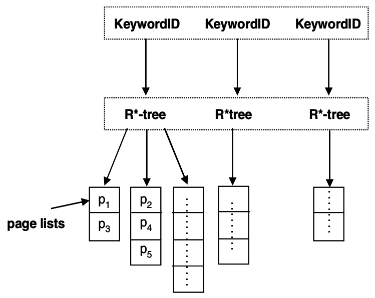

Figure 2. Inverted File - R*-tree

Advantages and Limitations: The first approach proposed in [22] is highly insensitive to *SK* queries with the *AND* semantics. This approach does not take advantage of the association of keywords in space. Hence, when query contains

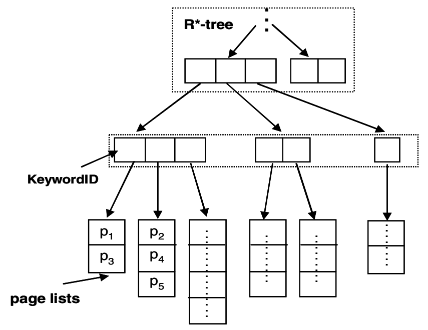

Figure 3. R*-tree - Inverted File

keywords that are closely correlated in space, this approach suffers from paying extra disk costs accessing different R*-trees and high overhead in the subsequent merging process.

In the second approach proposed in [22], the leaf nodes point to inverted index lists that are usually small. This is because any leaf node covers only a small sub-space of the entire data space and subsequently the number of distinct keywords that exist in this sub-space is expected to be small. This approach leverages the above advantage and hence the keyword filtering of objects are usually fast. However, the main disadvantage is the spatial filtering stage which generates many candidate object ids.

Considering the limitations of the previous approaches, we propose a new approach that exploits the association of keywords in space. We discuss the main ideas of our technique in the following section.

5. KR*-Tree: A Novel Indexing Mechanism

We propose a new indexing strategy called *KR*-tree*, which is an acronym for **K**eyword-**R*-tree**. We measure the effectiveness of indexing strategies in answering *SK* queries with respect to the following criteria:

- Pruning text and space.
- Handling queries with multiple keywords.

First, the indexing methods previously proposed use the pruning power of space and text, either separately or one followed by the other. As a consequence, *SK* queries are answered in a two-step filtering process, space followed by text or vice-versa. In *KR*-tree*, we exploit the pruning power of both space and text simultaneously, thus merging the two steps into one.

Secondly, in previous methods the keywords are maintained separately. Hence queries are answered by the intersection of object ids from the inverted index file or R*-trees of query keywords. In *KR*-tree*, we capture the joint distribution of keywords and hence the object ids containing the query keywords are directly obtained without merging any lists. These characteristics greatly enhance the performance of *KR*-tree* in answering *SK* queries.

At the outset, *KR*-tree* is similar to *R*-tree-Inverted File* data structure, but we make the following modifications.

- All internal and leaf nodes of *KR*-tree* are augmented with a set of distinct keywords that appear in the space covered by the nodes. Thus, many keywords appear in the upper level nodes of the tree and smaller number of keywords appear in the lower level nodes of the tree.
- Since the number of keywords that appear in each node varies, we do not store the keywords in the node. We construct a special list called *KR*-tree List* that stores the keywords appearing in the nodes. We give more details of *KR*-tree List* in the next subsection.

5.1 *KR*-tree* Construction

The *KR*-tree* is constructed in a way similar to how an R*-tree is constructed, but with minimal overhead in handling the keywords. There are two distinct steps involved in the construction of the tree.

1. First, the set of keywords corresponding to each node of the tree is determined.
2. Next, the set of keywords in each node is converted to a *KR-tree List*.

Step 1. In this step, we construct *KR*-tree* by inserting GIS data objects as shown in Algorithm 1. The algorithm is initialized with the data object O that needs to be inserted and the root node N_r of the tree. For any non-leaf node the algorithm computes the area of each of its child node by expanding it with O's rectangle (lines 2-4). The keyword list of N is updated with the keyword list of O (line 5). Then, it chooses the child with minimal area and inserts O recursively (lines 6 and 7). For a leaf node, the algorithm inserts O and updates the keyword list of N with O's list (line 10-11). If the leaf node gets full, N is split (line 13), otherwise, the algorithm returns.

The node split mechanism is explained in Algorithm 2. For any node N, the algorithm first creates two new nodes N_a and N_b (line 1). It then distributes the children of N to N_a and N_b and updates their bounding rectangles. The keyword list of N_a and N_b are updated using the keyword list of their children nodes (line 2-3). If N is not a root node, N_a and N_b are attached to the parent of N, otherwise a new root node is created. If the parent node gets full, then split occurs recursively (lines 4-14).

After all the data objects are inserted, each node of *KR*-tree* contain a set of distinct keywords that are covered under its space.

Algorithm 1 INSERT (Object O, Node N)

```
1: if $N$ is NOT leaf node then
2:   for all child $N_i$ of $N$ do
3:     $A_i = Area(N_i.r \cup O.r)$
4:   end for
5:   $N.t = \{N.t \cup O.t\}$
6:   Select $N_i$ with least $A_i$
7:   INSERT $(O, N_i)$
8:   Return
9: else
10:  Insert $O.r$ in $N$
11:  $N.t = \{N.t \cup O.t\}$
12:  if $N$ is full then
13:    SPLIT $(N)$
14:  else
15:    Return
16:  end if
17: end if
```

Algorithm 2 SPLIT (Node N)

```
1: Create Nodes $N_a$ and $N_b$
2: Distribute children of $N$ to $N_a$ and $N_b$
3: Update $N_a.t, N_a.r, N_b.t, N_b.r$
4: if $N$ is NOT root node then
5:   Attach $N_a$ and $N_b$ to $N$'s parent node $N_p$
6:   if $N_p$ is full then
7:     SPLIT $(N_p)$
8:   else
9:     Return
10:  end if
11: else
12:  Create new root node $N_r$
13:  Attach $N_a$ and $N_b$ to $N_r$
14: end if
```

Step 2. The number of keywords that appear in each node varies. Hence, we construct an auxiliary structure called *KR*-tree List*. This list stores for each keyword, the node ids of *KR*-tree* in which the keyword appears, along with its parent id and children ids. The *KR*-tree List* is similar to that of an inverted file index, the only difference is that it stores the node ids instead of object ids. The number of internal nodes of *KR*-tree* is considerably smaller compared to the number of objects indexed. Hence, *KR*-tree List* occupies only a small space.

5.2 Answering *SK* Queries

We answer a *SK* query as explained in Algorithm 3. The algorithm is initialized with query Q and the root of *KR*-tree*. First, the children node ids n_{ids} of current node N that contain all the keywords present in Q are obtained (lines 1-4). Next, each child of N that has a non-empty intersection with Q's rectangle and also whose id is present in the list n_{ids}, is chosen for further traversal (lines 5-6). If N is a leaf node, the qualified children are added as results, otherwise, the algorithm traverses recursively (line 7-13). Finally the set of objects O_{res} that satisfy Q are returned in line 14.

Algorithm 3 PROCESS (Query Q, Node N)

```
1: $n_{ids} = GetNodeIds(Q.t_1, N)$
2: for all $t_i$ in $Q.t$ such that $i > 1$ do
3:   $n_{ids} = GetNodeIds(t_i, N) \cap n_{ids}$
4: end for
5: for all Child $N_j$ of $N$ do
6:   if ($N_j.r \cap Q.r \neq \phi$ And $N_j \in n_{ids}$) then
7:     if $N_j$ is NOT leaf node then
8:       PROCESS $(Q, N_j)$
9:     else
10:      $O_{res} = O_{res} \cup N_j$
11:    end if
12:  end if
13: end for
14: Return $O_{res}$
```

5.3 Keyword Spread Problem

While we exploit the joint distribution of keywords in space, the flip side of it is what we encounter as *keyword-spread* problem. This problem occurs because *KR*-tree* indexes all the objects together, hence the number of nodes in which a keyword appear increases across the *KR*-tree*. This increase is large compared to that of the number of *KR*-tree* nodes that index only objects containing that particular keyword. We address this problem by slightly modifying our *KR*-tree* as shown in Figure 4. We traverse n levels up the *KR*-tree* and maintain an inverted file index for each keyword appearing below that level. Since we are maintaining a list of separate keywords, the list becomes small and controls the keyword-spread problem across *KR*-tree* nodes. In Figure 4, we can see the number of pages for the keyword `earthquake` gets compressed to single page below nodes n_1 and n_2, which otherwise would have occupied more pages. We use our modified *KR*-tree* in all of our experiments. The specific value of n depends on the number of objects that the *KR*-tree* indexes.

5.4 *KR*-Tree* Example

Consider a simple emergency GIS database presented in Table 1. The database has 10 objects whose spatial distribution and the corresponding *KR*-tree* nodes are shown in Figure 5. Now, let us consider an *SK* query: $< q, \{earthquake, shelter\} >$ that asks for *earthquake shelter* in the spatial region q. In order to answer this query, we will consider the following plans:
Space First. To answer our example query, we access the following nodes $\{r, n_1, n_2, n_3, n_4, n_5, n_6\}$ to get the following 5 object ids $\{o_2, o_5, o_6, o_8, o_{10}\}$. Next, the inverted

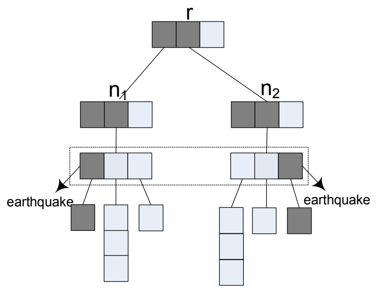

Figure 4. Modified KR*-tree

OID	X	Y	keywords
1	x_1	y_1	{fire, medical}
2	x_2	y_2	{earthquake, medical}
3	x_3	y_3	{fire, medical}
4	x_4	y_4	{facility, medical}
5	x_5	y_5	{earthquake, hazard}
6	x_6	y_6	{earthquake, medical}
7	x_7	y_7	{fire, facility}
8	x_8	y_8	{earthquake, shelter}
9	x_9	y_9	{fire, shelter}
10	x_{10}	y_{10}	{earthquake, shelter}

Table 1. Example GIS Database

keywords	Object List
earthquake	$\{o_2, o_5, o_6, o_8, o_{10}\}$
fire	$\{o_1, o_3, o_7, o_9\}$
medical	$\{o_1, o_2, o_3, o_4, o_6\}$
shelter	$\{o_8, o_{10}\}$
hazard	$\{o_5\}$
facility	$\{o_4, o_7\}$

Table 2. Inverted File Index

Leaf Node	keywords	Object List
n_3	fire	$\{o_1\}$
	medical	$\{o_1, o_2\}$
	earthquake	$\{o_2\}$
n_4	fire	$\{o_3\}$
	medical	$\{o_3, o_4\}$
	facility	$\{o_4\}$
	earthquake	$\{o_5\}$
n_5	earthquake	$\{o_6\}$
	medical	$\{o_6\}$
	fire	$\{o_7\}$
	facility	$\{o_7\}$
	hazard	$\{o_5\}$
n_6	earthquake	$\{o_8, o_{10}\}$
	shelter	$\{o_8, o_9, o_{10}\}$
	fire	$\{o_9\}$

Table 3. R-*tree Inverted Index

indices of textual keywords `earthquake` and `shelter` shown in Table 2 are looked up to further filter the object ids that contain both the keywords. Finally, the answer set for the query contains the objects $\{o_8, o_{10}\}$.

Using *R*-tree-Inverted File*. For the same example query, the R*-tree nodes $\{r, n_1, n_2, n_3, n_4, n_5, n_6\}$ are accessed to generate the candidate object set. Next, the inverted index list of leaf nodes $\{n_3, n_4, n_5, n_6\}$ are accessed to filter objects that contain the keywords `earthquake` and `shelter` to finally arrive at the answer set $\{o_8, o_{10}\}$.

Using *Inverted File-R*-tree*. For the example query, first the R*-trees of the keywords `earthquake` and `shelter` are loaded to get the candidate object id lists. Then, the intersection of the two candidate object id lists gives the answer set $\{o_8, o_{10}\}$. We do not show the working of this strategy due to space constraints. However, we will show in experiments how this strategy performs.

Using *KR*-tree*. We can do much better than the above strategies, if we know the joint distribution of keywords in the nodes of the *KR*-tree*. In general, if more than one keyword is present in a query, it can reduce the number of objects in which both keywords appear. In other words, the joint frequency of m keywords is no greater than the joint frequency of $m-1$ keywords. In our example, even though the keyword `earthquake` appears manytimes in space, when it combines with `shelter`, the number of appearances of both keywords reduces.

Table 5 shows the keywords that appear in space covered by all the nodes of the *KR*-tree*. For the given query, we access the root first to get its children n_1 and n_2. Now, for each child node, we not only check its spatial intersection with the query region, but also check for the presence of query keywords in it. In this case, n_1 does not contain both the keywords, but n_2 does, hence we only open node n_2. Applying the same principle for n_2's children, we can see that only n_6 satisfies the *SK* query. Hence we access only $\{r, n_2, n_6\}$ to generate the candidate object set.

The *KR*-tree List* for `earthquake` contains the following node ids: $\{n_1, n_2, n_3, n_4, n_5, n_6\}$. The *KR*-tree List* for all the keywords is shown in Table 4.

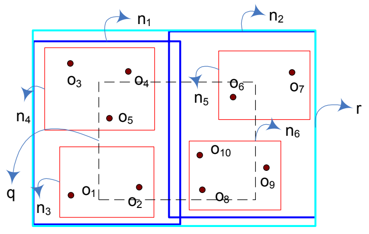

Figure 5. Objects Distributed in Space

6. Experiments

In this section, we describe our experimental results to evaluate our techniques in comparison with the methods

keywords	Node List
earthquake	$\{n_1, n_2, n_3, n_4, n_5, n_6\}$
fire	$\{n_1, n_2, n_3, n_4, n_5, n_6\}$
medical	$\{n_1, n_2, n_3, n_4, n_5\}$
shelter	$\{n_2, n_6\}$
facility	$\{n_1, n_2, n_4, n_5\}$
hazard	$\{n_1, n_4\}$

Table 4. KR*-Tree List

Node	keywords
r	$\{earthquake, medical, shelter, fire, facility, hazard\}$
n_1	$\{fire, medical, earthquake, facility, hazard\}$
n_2	$\{fire, medical, earthquake, shelter, facility\}$
n_3	$\{fire, medical, earthquake\}$
n_4	$\{fire, medical, earthquake, facility\}$
n_5	$\{medical, fire, earthquake, facility\}$
n_6	$\{earthquake, shelter, fire\}$

Table 5. KR*-Tree - Distribution of keywords in Space

presented in Section 4.

6.1 Datasets

In our experiments, we used real datasets downloaded from www.mapdex.org that catalogs vast amounts of GIS databases. Our GIR database is mainly a repository of GIS database(s) that contains different kinds of GIS objects. We employ a number of pre-processing steps to arrive at a more structured GIR database. The downloaded GIS databases are heterogeneous in nature; that is, each of them has different attribute fields. For each database, we identified and discarded the real-valued fields and retained the spatial and text fields. All the text fields are then combined to form one single text attribute. We performed basic pre-processing on the text such as stemming and stop-word removal. The spatial attributes from different databases are all converted to a single coordinate system. Now, our GIR database is more structured and homogeneous containing only spatial and textual data.

We used different sizes of datasets in our experiments. We categorized them as small, medium, and large datasets as shown in Table 6.

Dataset Size	Spatial Objects	keywords
Small	50000	75
Medium	125000	100
Large	1000000	5000

Table 6. Datasets

6.2 Queries

We generated 1000 *SK* queries with their spatial component consisting of different rectangle sizes and textual component consisting of 2 to 3 keywords. All of our queries consisted of a rectangle region of size s and a set of keywords. We randomly generated rectangles of different sizes and combined them with a set of keywords that were carefully chosen so that they resembled a real query trace.

6.3 Query Model

We used the following steps to generate query keywords:

1. Randomly select an object from the database.
2. From the distinct keywords appearing in the object's text field, generate required set of keywords.
3. Combine each set of keywords with the rectangle generated of particular size.

The above method is better than randomly combining keywords. In fact, our approach works superior for such unusual combinations, hence we did not want to bias our experimental results by adding such queries. Since our data structure indexes joint distribution of keywords, it captures the true correlation of keywords in space. For all datasets we generated rectangles of sizes 10km x 10km, 25km x 25km, 50km x 50km and 75km x 75km. These sizes represent geographic area of smaller to bigger regions. We generated two-set keywords and three-set keywords for each rectangle size combination. In our experiments, we report the average performance of queries in terms of random disk IOs. For small and medium dataset, we used a page size of 256KB, and for the large dataset we used a page size of 512KB in constructing the *KR*-tree*.

6.4 Performance Comparison

In Table 7, we show in a nut shell the performance of various indexing structures. If d_1 and d_2 are the disk IOs incurred by index i_1 and index i_2 respectively, we define performance gain through disk IO reduction of i_1 over i_2 as $d_{gain} = \frac{(d_2 - d_1)*100}{d_2}$. Positive value indicates better performance of i_1 over i_2. In this table, we report the d_{gain} of our *KR*-tree* in comparison with others.

keywords	Dataset Size	*KR*-tree* vs. *Inv.R*-tree* (d_{gain})	*KR*-Tree* vs. R*-tree.Inv (d_{gain})
2-keywords	small	37%	66%
	medium	24%	70%
	large	33%	67%
3-keywords	small	43%	61%
	medium	26%	68%
	large	36%	60%

Table 7. Average Disk I/Os

For 2-keyword queries, our indexing strategy achieves a reduction in disk IOs ranging from 24% to 37% and from 60% to 70% compared with *Inverted Index-R*-tree* and *R*-tree-Inverted Index*, respectively. For 3-keyword queries, the reduction is slightly more compared with *Inverted Index-R*-tree*, ranging from 26% to 43%. The improvement occurs bacause our strategy is sensitive to the number of keywords. As the number of keywords increases, *KR*-tree* performs better. On the other hand, *KR*-tree's* performance gain over *R*-tree-Inverted Index* is slightly reduced compared to the 2-keyword queries. This is because, the disk IOs in *R*-tree-Inverted Index* is dominated by the spatial filtering step that is insenstive to the number of keywords. On the other hand, the didk IOs for *KR*-tree* increased overall.

In Figures 6-11, we show detailed plots of disk IOs incurred by various strategies for different sizes of datasets and for different number of query keywords. In all the plots, we can see that *KR*-tree* significantly outperforms *R*-tree-Inverted Index* and moderately outperforms *Inverted Index-R*-tree*.

7. Related Work

There has been lot of interest in building geographic information retrieval systems. The first work of this kind started in the context of digital library (DL) projects such as GIPSY at UC Berkeley and Alexandria Digital Library Project at UC Santa Barbara [12]. In these projects, the main objective is to address the extraction of geographic references found in the text by using ontologies, gazetteers, thesaurus, etc., and convert them to coordinates for retrieving DL contents using geography.

In the context of geographic search engines, there are numerous academic projects. Most of them can be broadly classified under 1) work that focused on extraction of geographic references from documents and/or 2) efficient query processing. We will briefly describe a few of these. In GeoSearch System [10], the geographic scope of Web pages are extracted by analyzing the geographic references in text as well as the geographic location where the Web sites are registered. In [15], the focus is on improving the extraction techniques. In particular, after the relevant geographic references are extracted, ambiguities such as multiple place name references and alternate place names are resolved using techniques such as geo-matching and geo-propagation. Other relevant studies that addressed geographic search on the Web are [11, 5].

In the context of query processing for GIR, indexing techniques for processing text and geographic data are the main focus. In [11], a simple inverted index structure for text and grid file for geographic data are used. They propose a hybrid index structure in which each keyword is combined with different partitions of space. In effect what they are proposing is similar to [22]. The other technique proposed in their work concatenates keyword with region identifier. For example, keyword `earthquake` is combined with spatial region ”R1” and represented as spatio-textual key ”earthquakeR1”. All the documents that are in ”R1” and contains the text `earthquake` are attached as list to the key ”earthquakeR1”. There are drawbacks of this approach. First, for large set of objects, this approach will generate a large number of false positives. For query containing multiple keywords and spatial region, a number of such keys have to be looked up and filtered.

In a recent work [3], the authors propose to maintain individual indices for spatial and textual data. They propose various approaches to retrieve data from each index before the final merging of results. The spatial objects indexed in their applications are complex footprints that are extended regions in space. They approximate them by using MBRs and use memory-resident spatial index. Their approach does not scale well with increasing size of the dataset. To alleviate the problem, they propose to compress the MBRs, but the attempt generates large candidate set that needs to be fetched from the disk, with a high rate of false positives. This will become a major performance bottleneck for large scale GIR applications. In our work, we use disk-resident spatial index for GIR applications. Our data structure performs significantly better than their approach with respect to two aspects: 1) first it reduces the number of disk accesses in identifying the candidate objects and as a consequence 2) it reduces the overhead in merging the candidate objects.

In another very related work [22], the authors proposed a hybrid index by combining the spatial and inverted list structures. Their approaches either use multiple R*-trees to answer queries or generates more candidates for further filtering. The main limitations of their approach are already discussed in detail in Section 4.

8. Conclusions and Future Work

In this paper we proposed a novel indexing data structure called *KR*-tree* for processing *SK queries* with the *AND* semantics by capturing the joint distribution of keywords appearing in space. Given the common nature of such queries, we discussed the performance bottlenecks of current indexing mechanisms in processing them. Then we showed through an example GIS database and later through detailed experiments that our approach significantly reduces such bottlenecks by directly reducing the disk IOs incurred during spatial filtering of objects.

We plan to address some open problems as future work. First, we would like to explore how to adapt our indexing when there are many GIS databases. Here the main problem is that of database discovery not the individual records within the databases. In such a setting, the query semantics *(AND/OR)* changes and the user may require only *top-k* databases relevant to the query. The ranking is computed for the entire database using the individual records within the database. This requires storing the aggregated pre-computed scores of objects within our *KR*-tree* and coming up with pruning strategies to generate *top-k* results.

9 Acknowledgements

This work is supported by the National Science Foundation under Award Number 0331707.

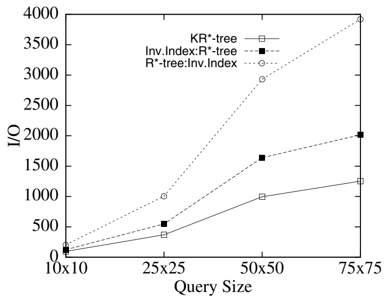

Figure 6. Small, 2-keywords.

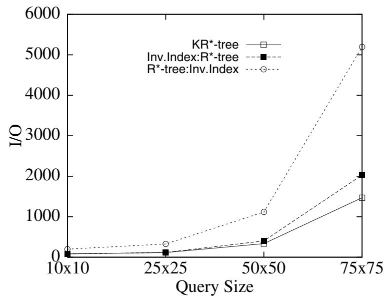

Figure 7. Medium, 2-keywords.

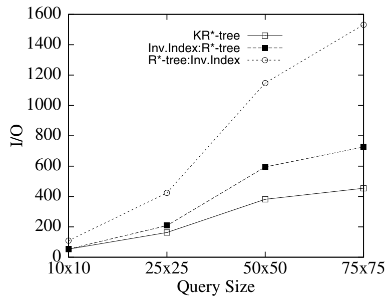

Figure 8. Large, 2-keywords.

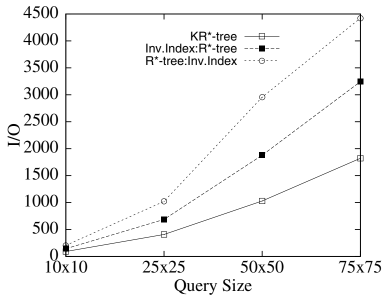

Figure 9. Small, 3-keywords.

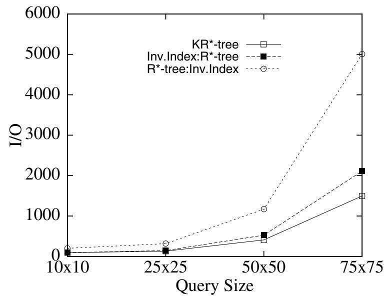

Figure 10. Medium, 3-keywords.

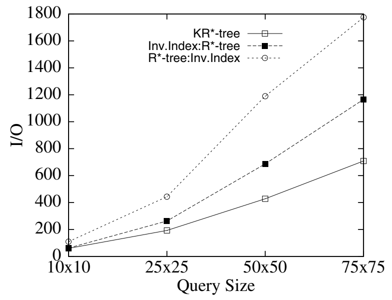

Figure 11. Large, 3-keywords.

References

[1] M.K. Beard and V. Sharma. Multidimensional Ranking in Digital Spatial Libraries. In *Special Issue of Metadata*. Journal of Digital Libraries, Vol.1, No. 1, 1997.

[2] O. Buyukkoten, J. Cho, H. Garcia-Molina, L. Gravano, and N. Shivakumar. Exploiting Geographical Information of Web Pages. In *WebDB*, pages 91-96, 1999.

[3] Y. Chen, T. Suel, and A. Markowetz. Efficeint Query Processing in Geographic Web Search Engines. In *Proc. of ACM SIGMOD*, pages 277-288, June 2006.

[4] J.Ding, L. Gravano, and N. Shivakumar. Computing Geographical Scopes of Web Resources. In *Proc. of VLDB*, pages 545-556, September 2000.

[5] M. Egenhofer. Toward the Semantic Geospatial Web. In *Proc. of ACM GIS*, pages 1-4, November 2002.

[6] ESRI Shapefile Technical Description. An ESRI White Paper - July 1998. http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

[7] Geospatial One-Stop E-Government Initiative. www.geodata.gov.

[8] S. Gobel and P. Klein. Ranking Mechanisms in Meta-Data Information Systems for Geo-Spatial Data. In *Proc. of EOGEO*, 2002.

[9] Google Local Search. http://local.google.com.

[10] L. Gravano. Geoserach: A Geographically-Aware Search Engine, 2003. http://geosearch.cs.columbia.edu

[11] C.B. Jones, A.I. Abdelmoty, D. Finch, G. Fu, and S. Vaid. The Spirit Spatial Search Engine.: Architecture, Ontologies and Spatial Indexing. In *Proc. of GIScience*, pages 125-139, October 2004.

[12] R.R. Larson. Geographic Information Retrieval and Spatial Browsing. In *GIS and Libraries: Patrons, Maps and Spatial Information*, pages 81-125, 1996.

[13] R.R. Larson. Geographic Information Retrieval (GIR): Searching Where and What. In *In Proc. of the 27th Intl. ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, page 600, July 2004.

[14] Mapdex Geographic Data Search. www.mapdex.org.

[15] A. Markowetz, Y. Chen, T. Suel, X. Long, and B. Seeger. Design and Implementation of a Geographic Search Engine. In *Proc. of WebDB*, June 2005.

[16] MSN Local Search. http://local.live.com.

[17] C. Ohm, G. Klump, and H. Kriegel. Xz-ordering: A Space-Filling Curve for Objects with Spatial Extension. In *Proc. of the 6th Intl. Symp. on Advances in Spatial Databases*, pages 75-90, July 1999.

[18] A. Singhal. Modern Information Retrieval: A Brief Overview. In *IEEE Data Engineering Bulletin*, Special Issue on Text and Databases, Vol. 24, No. 4, December 2001.

[19] Timos K. Sellis. Review - The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles In *ACM SIGMOD Digital Review 2*, 2000.

[20] S. Vaid, C.B. Jones, H. Joho, and M. Sanderson. Spatio-Textual Indexing for Geographical Search on the Web. In Proc. of SSTD, 2005.

[21] Yahoo Local Search. http://local.yahoo.com.

[22] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W. Ma. Hybrid Index Structures for Location-Based Web Search. In *Proc. of CIKM*, pages 155-162, November 2005.