Keyword Search on Spatial Databases*

Ian De Felipe Vagelis Hristidis Naphtali Rishe

School of Computing and Information Sciences
Florida International University
Miami, FL 33199
{ian.de.felipe, vagelis, rishen}@cis.fiu.edu

Abstract— **Many applications require finding objects closest to a specified location that contains a set of keywords. For example, online yellow pages allow users to specify an address and a set of keywords. In return, the user obtains a list of businesses whose description contains these keywords, ordered by their distance from the specified address. The problems of nearest neighbor search on spatial data and keyword search on text data have been extensively studied separately. However, to the best of our knowledge there is no efficient method to answer spatial keyword queries, that is, queries that specify both a location and a set of keywords.**

In this work, we present an efficient method to answer top-k spatial keyword queries. To do so, we introduce an indexing structure called IR^2-Tree (Information Retrieval R-Tree) which combines an R-Tree with superimposed text signatures. We present algorithms that construct and maintain an IR^2-Tree, and use it to answer top-k spatial keyword queries. Our algorithms are experimentally compared to current methods and are shown to have superior performance and excellent scalability.

I. Introduction

An increasing number of applications require the efficient execution of nearest neighbor (NN) queries constrained by the properties of the spatial objects. Due to the popularity of keyword search, particularly on the Internet, many of these applications allow the user to provide a list of keywords that the spatial objects (henceforth referred to simply as objects) should contain, in their description or other attribute. For example, online yellow pages allow users to specify an address and a set of keywords, and return businesses whose description contains these keywords, ordered by their distance to the specified address location. As another example, real estate web sites allow users to search for properties with specific keywords in their description and rank them according to their distance from a specified location. We call such queries *spatial keyword queries*.

A spatial keyword query consists of a query area and a set of keywords. The answer is a list of objects ranked according to a combination of their distance to the query area and the relevance of their text description to the query keywords. A simple yet popular variant, which is used in our running example, is the *distance-first spatial keyword query*, where objects are ranked by distance and keywords are applied as a conjunctive filter to eliminate objects that do not contain them.

Figure 1, which is our running example, displays a dataset of fictitious hotels with their spatial coordinates and a set of descriptive attributes (name, amenities). An example of a spatial keyword query is "find the nearest hotels to point *[30.5, 100.0]* that contain keywords *internet* and *pool*". The top result of this query is the hotel object H_7.

Unfortunately there is no efficient support for top-*k* spatial keyword queries, where a prefix of the results list is required. Instead, current systems use ad-hoc combinations of nearest neighbor (NN) and keyword search techniques to tackle the problem. For instance, an R-Tree is used to find the nearest neighbors and for each neighbor an inverted index is used to check if the query keywords are contained. We show that such two-phase approaches are inefficient.

We present a method to efficiently answer top-*k* spatial keyword queries, which is based on the tight integration of data structures and algorithms used in spatial database search and Information Retrieval (IR). In particular, our method consists of building an Information Retrieval R-Tree (*IR^2-Tree)*, which is a structure based on the R-Tree [Gut84]. At query time an incremental algorithm is employed that uses the IR^2-Tree to efficiently produce the top results of the query.

The IR^2-Tree is an R-Tree where a signature (Faloutsos and Christodoulakis [FC84]) is added to each node *v* of the IR^2-Tree to denote the textual content of all spatial objects in the subtree rooted at *v*. Our top-*k* spatial keyword search algorithm, which is inspired by the work of Hjaltason and Samet [HS99], exploits this information to locate the top query results by accessing a minimal portion of the IR^2-Tree. This work has the following contributions:

- The problem of top-*k* spatial keyword search is defined.
- The IR^2-Tree is proposed as an efficient indexing structure to store spatial and textual information for a set of objects. Efficient algorithms are also presented to maintain the IR^2-Tree, that is, insert and delete objects.
- An efficient incremental algorithm is presented to answer top-*k* spatial keyword queries using the IR^2-Tree. Its performance is evaluated and compared to current approaches. Real datasets are used in our experiments that show the significant improvement in execution times.

Note that our method can be applied to arbitrarily-shaped and multi-dimensional objects and not just points on the two

* This research was supported in part by NSF grants CNS-0320956, CNS-0220562, HRD-0317692, and IIS-0534530

dimensions, which are used in our running examples for clarity.

This paper is organized as follows. Section 2 formally defines the top-*k* spatial keyword search problem. Section 3 presents required background knowledge. Section 4 presents the IR^2-Tree and its maintenance algorithms. Section 5 presents our incremental search algorithm along with other baseline algorithms. Section 6 experimentally compares our search algorithm to baseline algorithms. Section 7 discusses related work and we conclude in Section 8.

	Name	Latitude	Longitude	Amenities
H_1	Hotel A	25.4	-80.1	tennis court, gift shop, spa, Internet
H_2	Hotel B	47.3	-122.2	wireless Internet, pool, golf course
H_3	Hotel C	35.5	139.4	spa, continental suites, pool
H_4	Hotel D	39.5	116.2	sauna, pool, conference rooms
H_5	Hotel E	51.3	-0.5	dry cleaning, free lunch, pets
H_6	Hotel F	40.4	-73.5	safe box, concierge, internet, pets
H_7	Hotel G	-33.2	-70.4	Internet, airport transportation, pool
H_8	Hotel H	-41.1	174.4	wake up service, no pets, pool

Figure 1: Sample dataset of hotel objects.

II. Problem Definition

In this work, a (spatial) object T is defined as a pair $(T.p, T.t)$, where $T.p$ is a location descriptor in the multi-dimensional space, and $T.t$ is a text document (textual description). Let D be the universe of all objects in a database. In Figure 1, $T.p$ is the point composed of "latitude" and "longitude", while $T.t$ is the concatenation of the "name" and "amenities" attributes.

A *top-k spatial query* Q_s searches through the multi-dimensional space to find the k nearest objects to the specified query point p. The spatial objects are ranked by distance such that an object closer to p has a higher rank. In particular, $score(T) = distance(T.p, p)$. For example, in Figure 1, object H_4 is ranked first, given $p=[30.5, 100.0]$.

A *keyword query* Q_w is a set of keywords $w_1,\ldots,w_m$. The result of Q_w is a list of objects ordered by the relevance $IRscore(T.t, Q_w)$ of their textual descriptions to the query keywords, as measured by an IR ranking function [Sin01].

A special case, used in our running examples, is the Boolean keyword query which returns the set of all objects whose text document contains all of w1,…,wm. That is,

$$Ans(Q_w) = \{T \in D \mid \forall w \in Q_w, w \in T.t\}$$

For example, in Figure 1, objects H2, H7 are the results of Boolean keyword query {"internet", "pool"}.

A *top-k spatial keyword query* Q is a combination of a top-k spatial query and a keyword query. In particular, Q is defined as a number $Q.k$ of requested results, a point $Q.p$, a set $Q.t = \{w_1,\ldots,w_m\}$ of keywords, and a ranking function:

$f(distance(T.p, Q.p), IRscore(T.t, Q.t))$

The result of Q is a list of the top-k objects T ranked according to the ranking function f.

A special case is the *distance-first top-k spatial keyword query* Q, used in our running examples, which returns a ranked list of the k objects that contain all of $w_1,\ldots,w_m$ and are closest to $Q.p$. That is, *distance-first top-k spatial keyword query* is a combination of a top-k spatial query and a Boolean keyword query. It is,

$$Ans(Q) = \left\{ Q.k \text{ first } T \in D \text{ ordered by distance } (T.p, Q.p) \;\middle|\; \forall w \in Q.t, w \in T.t \right\}$$

For example, in Figure 1, objects H7, H2 are the results of a distance-first top-k spatial query with $Q.k = 2$, $Q.p = [30.5,100.0]$ and $Q.t = \{$"internet", "pool"$\}$. Our work tackles the problem of efficiently answering top-k spatial keyword queries.

III. Background on Incremental NN

Figure 2 shows an example of an R-Tree using the hotel dataset of Figure 1. An MBR is represented by its southwest and its northeast points. An R-Tree is typically stored on disk and each R-Tree node takes a whole disk block; hence access to a node requires one disk I/O. The number of children each node can reference is called *node capacity*.

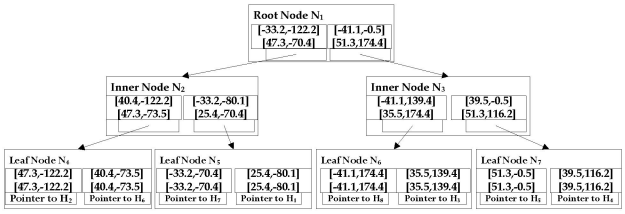


Figure 2: R-Tree for dataset of Figure 1.

The Incremental Nearest Neighbor algorithm presented by Hjaltason and Samet [HS99] uses the structure of an R-Tree to access a minimal number of R-Tree nodes and objects to retrieve the objects nearest to a given point or area in an incremental fashion. Figure 3 shows the Incremental Nearest Neighbor algorithm for two-dimensional objects. The input parameters are a point p, which is the query point (an area could be used instead), and a priority queue U which is initialized with the root of the R-Tree R. Line 2 returns the queue element which has the smallest distance from the query point.

```
NearestNeighbor(p,U)
    /* priority queue U initially contains root node of R with distance 0 */
1   while not U.IsEmpty()
2     E ← U.Dequeue()
3     if E is a non-Leaf Node
4       for each (NodePtr,MBR) in E
5         U.Enqueue(LoadNode(NodePtr),Dist(p,MBR))
6     else if E is a Leaf Node
7       for each (ObjPtr,MBR) in E
8         U.Enqueue(ObjPtr,Dist(p,MBR))
9     else /* E is an object pointer */
10      return E as next nearest object pointer to p
```

Figure 3: Incremental Nearest Neighbor algorithm.

If the element is a leaf node, then each child object, referenced by *ObjPtr*, is inserted in the queue based on its distance. If it is a non-leaf node, each child node, referenced

by *NodePtr*, is inserted in the queue. Finally, if the element is a pointer to a spatial object, it is reported as the next result of the algorithm, as show in Line 10. The `Dist` function computes the distance between the query point *p* and a MBR. We assume that the R-Tree is disk resident, thus, the `LoadNode` function loads the node from the disk block.

Example 1: ***Executing the Incremental Nearest Neighbor algorithm on the R-Tree of Figure 2 for the query point [30.5, 100.0] results in the following sequence of steps:***

1. Enqueue N_1; U={(N_1, 0.0)}
2. Dequeue N_1; Enqueue N_2, N_3; U={(N_3, 0.0), (N_2, 170.4)}
3. Dequeue N_3; Enqueue N_6, N_7; U={(N_7, 9.0), (N_6, 39.4), (N_2, 170.4)}
4. Dequeue N_7; Enqueue H_5, H_4; U={(H_4, 18.5), (N_6, 39.4), (H_5, 102.6), (N_2, 170.40)}
5. Dequeue and Return H_4

If we continue, objects H_3, H_5, H_8, H_6, H_1, H_7, H_2 are returned next. □

IV. IR²-TREE

The IR²-Tree is a combination of an R-Tree and signature files. In particular, each node of an IR²-Tree contains both spatial and keyword information; the former in the form of a minimum bounding area and the latter in the form of a signature. An IR²-Tree facilitates both top-*k* spatial queries and top-*k* spatial keyword queries as we explain below.

More formally, an IR²-Tree *R* is a height-balanced tree data structure, where each leaf node has entries of the form *(ObjPtr, A, S)*. *ObjPtr* and *A* are defined as in the R-Tree while *S* is the signature of the object referred by *ObjPtr*. A non-leaf node has entries of the form *(NodePtr, A, S)*. *NodePtr* and *A* are defined as in the R-Tree while *S* is the signature of the node. The signature of a node is the superimposition (OR-ing) of all the signatures of its entries. Thus a signature of a node is equivalent to a signature for all the documents in its subtree. Figure 6 shows an IR²-Tree for the sample dataset of Figure 1. To simplify the following presentation we focus on the two-dimensional space.

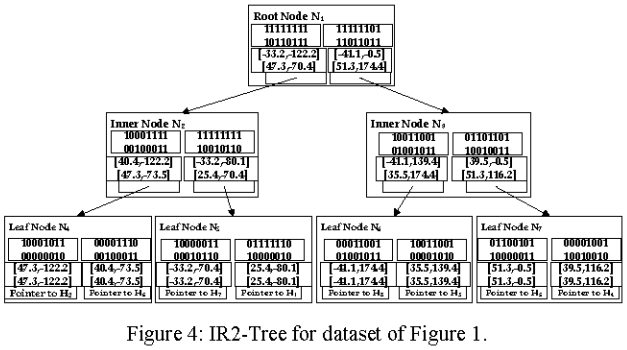


Figure 4: IR2-Tree for dataset of Figure 1.

The IR²-Tree is maintained through insert and delete operations, which are modifications of the corresponding R-Tree operations. Figures 7 and 8 show the `Insert` and `Delete` algorithms respectively.

The `Insert` algorithm uses a standard R-Tree implementation of `ChooseLeaf`, which can be found in [Gut84]. We use the standard Quadratic Split technique [Gut84] for node splitting. We modify the standard `AdjustTree` method to also maintain the signatures of the modified nodes. That is, if a new bit is set to 1 in a node *N*, then it must be also set to 1 for *N*'s ancestors. Finally, we assume that all tree related algorithms have implicit access to the root node of the IR²-Tree *R*.

The input of the `Insert` algorithm is a pointer to an object *T*, its MBR, and its signature. Line 1 retrieves a leaf node *N* which is best suited according to the MBR of *T*. Then *T*'s pointer, MBR, and signature are stored in *N*. If *N* has reached its maximum node capacity then it will split. If *N* is split into nodes *O* and *P*, on Line 4, and it is the root node, a new node *M* will be created. *M* becomes the parent *O* and *P* and stores their pointer, MBR, and signature. Finally, *M* is declared the new root node. If *N* is not the root then its parent node has to be updated as is the case on line 14 or 18. Finally, since we assume that the IR²-Tree is disk resident, the `StoreNode` function stores the node to the corresponding disk block(s).

Standard implementation of `FindLeaf` is used in the implementation of `Delete`. However, `CondenseTree` is modified to maintain the signatures of updated nodes, similarly to `AdjustTree` above. In Line 1 of Figure 8, a search for a leaf node *N* containing an unwanted object *T* is performed. If such *N* exists, *T* is removed from *N*, otherwise the algorithm stops. If *T* is removed, the tree is condensed and proper tree maintenance takes place.

```
Insert(ObjPtr,MBR,S)
1   N ← ChooseLeaf(MBR)
2   N.Add(ObjPtr,MBR,S)
3   if N needs to be split
4     {O,P} ← N.Split() /*nodes O and P are returned */
5     if N.IsRoot()
6       initialize a new node M
7       M.Add(O.Ptr,O.MBR,O.S)
8       M.Add(P.Ptr,P.MBR,P.S)
9       StoreNode(M)
10      StoreNode(O)
11      StoreNode(P)
12      R.RootNode ← M
13    else
14      AdjustTree(N.ParentNode,O,P)
15  else
16    StoreNode(N)
17    if not N.IsRoot()
18      AdjustTree(N.ParentNode,N,null)
```

Figure 5: Insert method for IR²-Tree.

Clearly, the complexity of the `Insert` and `Delete` algorithms is the same as in an R-Tree, since the only additional operation is the maintenance of the signatures of the updated nodes and their ancestors. Note that the updating of the signatures throughout a node and its ancestor is being done at the same time the tree would normally update the MBR of a node and its ancestors.

To account for the extra space needed to store the signatures in an IR²-Tree node, and in order to have the same number of children as in the corresponding R-tree, we allocate additional disk block(s) to an IR²-Tree node when needed.

This fact has a minor impact on the performance of the IR^2-Tree algorithms as shown in Section 6.1.

```
Delete(ObjPtr)
1   N ← R.FindLeaf(ObjPtr)
2   if N was not found
3      return
4   else
5      N.Remove(ObjPtr)
6      CondenseTree(N)
7      if R.RootNode has only one child M
8         R.RootNode ← M
```

Figure 6: Delete method for IR2-Tree.

Multilevel IR^2-Tree

A drawback of the IR^2-Tree described above is that the same signature length is used for all levels which leads to more false positives in the higher levels, which have more 1's (since they are the superimpositions of the lower levels). To address this problem, we use varying signature lengths for different levels. This is achieved using multi-level superimposed coding [CS89,DR83,LKP95], which reduces the number of false positives, particularly in non-leaf nodes. In this case, we use the optimal signature length for each level (we use the optimal signature length formula from [MC94]), and superimpose the signatures of all objects in the subtree of each node, instead of the signatures of the children nodes as before. A drawback of this variant, called *Multi-level IR^2-Tree* (MIR^2-Tree), is that it significantly increases the complexity of the tree maintenance operations (`Insert` and `Delete`) since for each object inserted or deleted, we have to recompute the signatures of all ancestor nodes by accessing all underlying objects and not just by superimposing the children's signatures as before. We compare the performances of IR^2-Tree and MIR^2-Tree in Section 6.

V. Algorithms to Answer Top-*K* Spatial Keyword Queries

We consider two baseline algorithms, in Section 5.1, which are named based on the underlying data structures they use: the *R-Tree*, and the Inverted Index Only (*IIO*). In Section 5.2 we present the distance-first IR^2 algorithm which uses the IR^2-Tree structure to answer distance-first top-*k* spatial keyword queries. Then in Section 5.3 we present the general IR^2 algorithm which uses the IR^2-Tree structure to answer general top-*k* spatial keyword queries. Note that these last two algorithms can also operate on MIR^2-Trees with no modification.

A. Current Baseline Algorithms

For simplicity we describe the R-Tree and the IIO baseline algorithms for the simpler distance-first top-k spatial keyword queries, which are also used in the experiments (Section 6). Both algorithms can be extended to answer general top-k spatial keyword queries.

R-Tree Algorithm

The first baseline algorithm, R-Tree, makes use of only an R-Tree data structure. Given a distance-first top-*k* spatial keyword query, the algorithm first finds the top-1 nearest neighbor object to the query point *Q.p*. Then it retrieves that object (since the R-tree only contains object pointers) and compares that object's textual description with the keywords of the query. If the comparison fails then that object is discarded, and the next nearest object is retrieved. The incremental NN algorithm in Figure 3 ([HS99]) is used. This process continues until an object is found whose textual description contains the query keywords. Once a satisfying object is found it is returned and the process repeats until *k* objects have been returned.

The drawback of this algorithm is that it has to retrieve every object returned by the NN algorithm until the top-*k* result objects are found. This potentially can lead to the retrieval of many "useless" objects. In the worst case (when none of the objects satisfies the query's keywords) the entire tree has to be traversed and every object has to be inspected.

IIO Algorithm

The IIO baseline algorithm makes use of an inverted index. It first finds all the objects (object ids) whose text document contains the query keywords by intersecting the lists returned by the inverted index. Let *V* be the set of objects in this intersection. Then the objects in *V* are retrieved and the distance between the query point *Q.p* and each of the objects in *V* is computed. These objects are sorted and the top-*k* objects are returned. Figure 9 shows the IIO algorithm. The input parameters are the inverted index *I* and the distance-first top-*k* spatial keyword query *Q*.

Example 2: ***Consider the query top-2 hotels from point [30.5, 100.0] containing the keywords {"internet", "pool"} on the data of Figure 1. The trace of IIO algorithm is the following:***

1. *H_1, H_2, H_6, H_7 are returned by the inverted index for keyword "internet"*
2. *H_2, H_3, H_4, H_7, H_8 are returned by the inverted index for keyword "pool"*
3. *H_2, H_7 are the result after the intersection*
4. *Objects H_2, H_7 are accessed to get their coordinates*
5. *Add H_2 to list L={(H_2, 222.8)}*
6. *Add H_7 to list L={(H_7, 181.9), (H_2, 222.8)}*

return H_7, H_2 as the result. □

The performance of this algorithm deteriorates if many objects contain the query keywords. In this case the inverted index would return many objects, which are then retrieved and inspected. Notice that IIO is the only non-incremental algorithm presented in this paper. That is, IIO computes all the query results and its performance is independent of *k*, as shown in Section 6.

B. Distance-First IR^2-Tree Algorithm

In this section we present the distance-first version of the IR^2-Tree algorithm, which outputs the objects that contain all

query keywords ordered by their distance from the query point. In Section 5.3 we show how this algorithm is generalized to handle general top-*k* spatial keyword queries.

```
IIOTopK(I,Q)
     /* I is the inverted index */
1    for each word wi in Q.t do
2      Li ← I.RetrieveObjectPointersList(wi)
3    V ← intersection of object pointers in Li's
4    initialize a list L
5    for each ObjPtr in V do
6      T ← LoadObject(ObjPtr)
7      d ← Dist(Q.p,T.p)
8      L.Add(T,d)
9    sort items in L by distance
10   return first Q.k objects in L
```

Figure 7: Inverted Index Only (IIO) algorithm.

The distance-first IR^2-Tree algorithm exploits the structure of the IR^2-Tree to efficiently answer distance-first top-*k* spatial keyword queries. The tree traversal is based on the Incremental Nearest Neighbor algorithm (Figure 3). The key advantage of this algorithm is that it prunes whole subtrees if their root-node signature does not match the query signature *Signature(Q.t)*. This happens because the signature of an IR^2-Tree node is composed from all the signatures of its children. This pruning occurs in addition to the spatial pruning provided by the traditional Incremental Nearest Neighbor. By tightly integrating these two pruning mechanisms, the distance-first IR^2-Tree algorithm accesses a minimal set of IR^2-Tree nodes and objects to answer a distance-first top-*k* spatial keyword query.

Figure 10 shows the distance-first IR^2-Tree algorithm (`IR2TopK`). The key methods is `IR2NearestNeighbor(.)`, which is based on the `NearestNeighbor` algorithm but inputs an additional input parameter `W`, which is the signature of the query. The signatures of nodes and objects are compared against `W` and are skipped if their signatures do not match `W` (i.e. they are dropped from the search queue). Notice that each call to the `IR2NearestNeighbor(.)` method returns a candidate result object, which is then checked (Line 21) to ensure it is not a false positive.

Example 3: ***As an example we trace the execution of the algorithm on the IR^2-Tree of Figure 6 to answer the query: top-2 hotels from point [30.5, 100.0] containing the keywords {"internet", "pool"}.***

1. *Enqueue N_1; U={(N_1, 0.0)}*
2. *Dequeue N_1; Enqueue N_2; U={(N_2, 170.4)}*
3. *Dequeue N_2; Enqueue N_4, N_5; U={(N_5, 170.5), (N_4, 173.8)}*
4. *Dequeue N_5; Enqueue H_7; U={(N_4, 173.8), (H_7, 181.9)}*
5. *Dequeue N_4; Enqueue H_2; U={(H_7, 181.9), (H_2, 222.8)}*
6. *Dequeue and return H_7;*
7. *Dequeue and return H_2;*

Notice how the IR^2-Tree signature pruning ability starts to emerge in Line 2. Only one child of N_1 is enqueued. The other child is discarded as it fails the signature check. Objects H_1 and H_6 also get pruned when their parent is visited (Lines 4 and 5). □

C. IR^2-Tree Algorithm

In this section we present the general version of the IR^2-Tree algorithm, where objects are output ordered by a ranking function *f(distance(T.p, Q.p), IRscore(T.t, Q.t))* as defined in Section 2. The key differences to the distance-first version are that:

(i) We do not create a single signature *Signature(Q.t)* for the query, but instead we use the individual signatures, *Signature(w)*, $w \in Q.t$, of the query keywords. The reason is that we do not use AND semantics, that is, an object containing only some of the query keywords may be in the result.

(ii) We can no longer output an object as soon as we know it is the next closest and contains all query keywords, because a farther object may have a higher overall *f(.)* score. Hence, the nodes *v* in the queue *U* are ordered by the maximum score that an object *T* inside them may have, that is, by:

$Upper(v) = UpperBound_{T \in v}(f(distance(T.p, Q.p), IRscore(T.t, Q.t))$

Assuming that *f()* is decreasing with *distance()* and increasing with *IRscore()* we have:

$Upper(v) = LowerBound_{T \in v}(f(distance(v.MBR, Q.p), UpperBound_{T\text{-has-signature-}v.S}(IRscore(T.t, Q.t)))$

```
IR2NearestNeighbor(p,W,U)
1    while not U.IsEmpty()
2      E ← U.Dequeue()
3      if E is a non-Leaf Node
4        for each (NodePtr,MBR,S) in E
5          if S matches W
6            U.Enqueue(LoadNode(NodePtr),Dist(p,MBR))
7      else if E is a Leaf Node
8        for each (ObjPtr,MBR,S) in E
9          if S matches W
10           U.Enqueue(ObjPtr,Dist(p,MBR))
11     else /* E is an object pointer */
12       return E as next nearest object pointer to p
IR2TopK(R,Q)
13   initialize a list L
14   Initialize a priority queue U
15   U.Enqueue(R.RootNode,0)
16   W ← Signature(Q.t)
17   c ← 0
18   while c < Q.k
19     ObjtPtr ← IR2NearestNeighbor(Q.p,W,U)
20     T ← LoadObject(ObjPtr)
21     if T.t contains all keywords in Q.t
22       c ← c + 1
23       L.add(T)
24   return L
```

Figure 8: Distance-First IR2-Tree algorithm.

To compute the maximum possible IR score $UpperBound_{T\text{-has-signature-}v.S}(IRscore(T.t, Q.t))$ of an object in the MBR of *v* we can assume that *v* has an imaginary object *T* that contains all keywords of *Q* specified by the signature of *v.S* exactly once (term frequency *tf*=1), that is, we assume no false positives. Hence, the document length (*dl*) of *T.t* is the number of such keywords. Then, we can use a traditional *tfidf* IR ranking function [Sin01]. This method facilitates outputting result-objects as early as possible. Note that it is not possible to estimate a tight maximum possible IR score if

the IR function uses advanced features like thesaurus or ontology.

We make the following specific changes to the distance-first version of Figure 10:

1. Replace Line 16 with:

```
for each keyword w_i in Q.t do
  W_i ← Signature(w_i)
```

2. Replace Lines 21-23 with:

```
Score ← f(distance(T.p,Q.p),IRscore(T.t,Q.t))
if Score ≥ Upper(U.top())
/* check if actual score of T is greater or equal to the max possible score
of the objects in the queue */
  c ← c + 1
  L.add(T)
else
  U.Enqueue(T,Score) /*to be considered later*/
```

3. Replace Lines 5-6 with:

```
Score ←UpperBound_T-has-signature-s(IRscore(T.t,Q.t))
if Score > 0
  U.Enqueue(LoadNode(NodePtr),Score)
/* check if there can be an object T with non-zero IR score. The "if"
condition can be removed if results with 0 IR score are acceptable*/
```

4. Replace Lines 9-10 with:

```
Score ←UpperBound_T-has-signature-s(IRscore(T.t,Q.t))
if Score > 0
  U.Enqueue(ObjPtr,Score)
/* check if T has non-zero IR score. The "if" condition can be removed if
results with 0 IR score are acceptable*/
```

VI. EXPERIMENTS

To measure the performance of the IR^2-Tree, MIR^2-Tree, and baseline algorithms, we have implemented all algorithms and underlying data structures in Java. All index structures (R-Tree, IR^2-Tree, MIR^2-Tree and inverted index) are disk-resident. We focus on the distance-first version of the top-k spatial keyword query, since its results are easier to comprehend and analyze. The spatial objects are stored in a plain text file and the leaf nodes of the tree data structures store pointers to the object locations in the file. We make comparisons based on the disk accessed required to satisfy a query and the execution time. An Athlon 64 3400+ (NewCastle) with 2GB of RAM and 74GB 10000RPM drive was used for the experiments.

We present the results of two datasets provided by the High Performance Database Research Center (http://hpdrc.fiu.edu/). Both datasets are plain text files (tab delimited) where each spatial object occupies a row. The first dataset contains objects that represent hotels and will be referred as the Hotels dataset. The second dataset will be referred as the Restaurants dataset and contains restaurant data. Table 1 shows more details of the two datasets.

In all experiments the disk block size is 4,096 KB. Also, the number of children of a node of the R-Tree is computed given the fact the each node is a disk block. This translates to 113 children per node in our implementation. We use this same number of children for the IR^2- and MIR^2-Trees, which typically requires two disk blocks per node. As the experiments show, the extra disk block overhead adds to the size of the IR^2- and MIR^2-Trees but has little effect on the execution time.

We compare the performance of the IR^2-Tree and MIR^2-Tree algorithm with that of the R-Tree and IIO algorithms. Three sets of experiments were carried out. The first measures the performance of the algorithms for varying values of requested results k (top-k). The second set measures the effect of the number of query keywords. Finally, the third set of experiments shows the effect of the signature length r.

TABLE 1: DATASET DETAILS

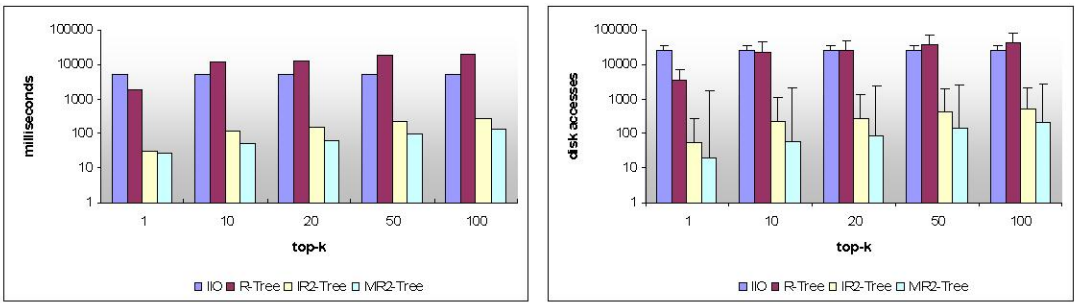
Dataset	Size (MB)	Total # of objects	Average # unique words per object	Total # unique words in dataset	Average # disk blocks per object
Hotels	55.2	129,319	349	53906	2
Restaurants	61.3	456,288	14	73855	1

Varying k (top-k)

In this experiment we fix the number of query keywords to 2 and the signature length to 189 bytes (for Hotels dataset) and 8 bytes (for Restaurants dataset). Note that this signature length is longer at the top levels of the MIR^2-Tree, which uses variable signature sizes. These signature lengths were chosen to balance space requirements and performance. The results of this experiment are shown in Figures 9 and 12 for Hotels and Restaurants respectively. The graph y-axes have logarithmic scale to illustrate the difference more clearly.

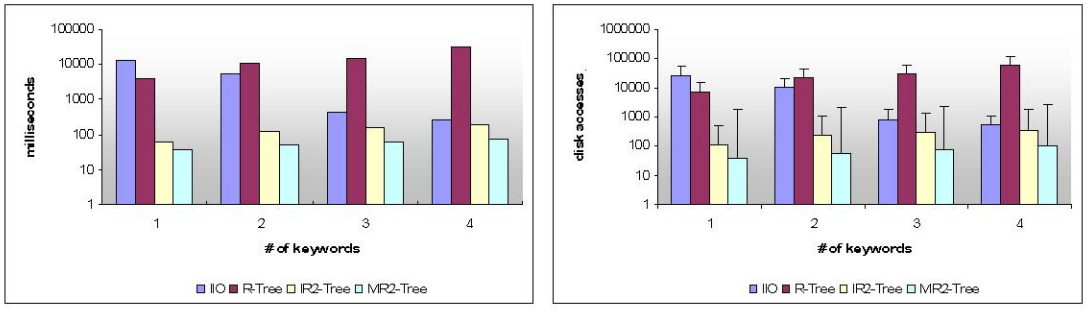
The graph shows that IR^2-Tree and MIR^2-Tree perform better than R-Tree for all values of k. This is expected since the R-Tree approach will have to access more objects and potentially more tree nodes as well. In contrast, the IR^2-Tree and MIR^2-Tree use the signatures to prune whole subtrees. In particular, the MIR^2-Tree does a better job filtering inner nodes since the optimal signature length is used for each tree level, as described in Section 4.

Figures 9b and 12b show the disk block accesses for the algorithms. The thick bars illustrate the number of random disk block accesses while the thin lines on top of the thick bars show the numbers of sequential disk block accesses. As expected, the execution time is primarily proportional to the random access numbers. Note that MIR^2-Tree performs fewer random disk block accesses than the IR^2-Tree because of the pruning effect, but performs more sequential disk block accesses. This is mainly because the nodes of the top levels of the MIR^2-Tree occupy more disk blocks due to their longer signatures. The IIO algorithm is insensitive to the k value since it has to examine all objects that contain all keywords.



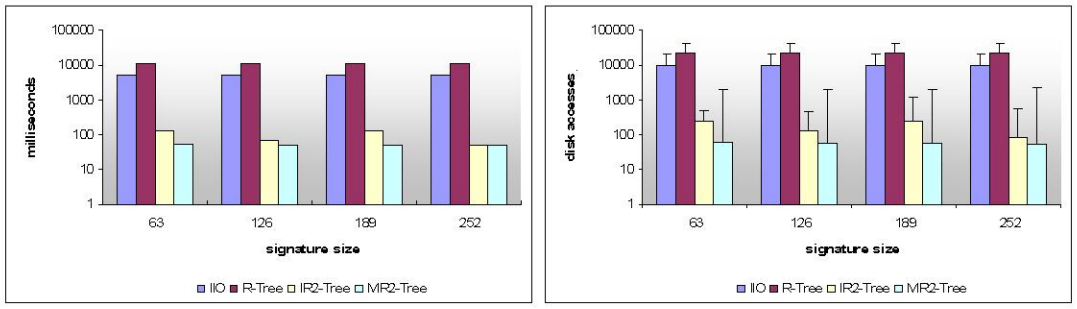
(a) Execution Time (b) Object Accesses

Figure 9: Results for varying k (top-k) searches for the Hotels dataset



(a) Execution Time (b) Object Accesses

Figure 10: Results for varying number of keywords for the Hotels dataset



(a) Execution Time (b) Object Accesses

Figure 11: Results for varying signature lengths (in bytes) for the Hotels dataset

Vary number of keywords

In this experiment we fix the number of requested objects k to 10 and the signature lengths as above. Refer to Figures 10 and 13 for the results of this experiment. By increasing the number of keywords we reduce the number of objects that contain all of them (since distance-first top-k spatial keyword queries are conjunctive). We note that the IIO algorithm performs better as the number of keywords increases, since the intersection of the inverted lists then becomes shorter and hence the object accesses fewer.

Vary signature length

In this experiment we fix k to 10 and the number of keywords to 2. Refer to Figures 11 and 14 for the results of this experiment. First note, that the signatures chosen for the Hotels dataset are different than those for the Restaurants dataset. This is because a Hotel object contains more unique words than a Restaurant object, as shown in Table 1. Note that the displayed signature lengths are used for the leaf nodes of MIR2-Tree. Longer signatures are used for the top nodes. There is a trade-off in increasing the signature lengths for IR2-Tree and MIR2-Tree. Increasing the signature length decreases the false positives but increases the occupied space of the tree structures, which can lead to more disk accesses. Hence, there is no clear trend for varying the signature lengths.

A. Space Requirements

Table 2 shows the total size (in MB) of each structure. For IR^2-Tree and MIR^2-Tree we consider the instance used for the "varying top-k" and "varying keywords" experiments above,

i.e., signature lengths of 189 and 8 bytes for the Hotels and Restaurant dataset respectively.

Notice that the size of the IIO structure is significantly less for the Restaurants dataset. This is because the Restaurant dataset contains far less unique keywords per object than the Hotels dataset as shown in Table 1. On the other hand, the sizes of the tree structures are larger for the Restaurants dataset because it has more objects than the Hotels dataset.

B. Discussion

As shown in the "vary signature length" experiment above, by increasing the signature size we achieve fewer accesses to spatial objects and inner nodes by eliminating false positives. On the other hand, larger signatures also increase the size of the IR^2-Tree and MIR^2-Tree. The larger signatures impact the size of the IR^2-Tree more than that of the MIR^2-Tree, since the signature size is consistent thought all nodes of an IR^2-Tree. The signature size only impact the leaf nodes of the MIR^2-Tree because the inner nodes are recalculated based on the object that the subtree can reference.

TABLE 2: SIZES (MB) OF INDEXING STRUCTURES

Dataset	IIO	R-Tree	IR^2-Tree	MIR^2-Tree
Hotels	31.4	6.9	34.5	44.9
Restaurants	7.2	23.9	47.2	68.2

Also, in the rare case where every query keyword appears in very few objects, the IIO method will be faster since the inverted lists would be very short. On the other extreme, if the query keywords appear in almost all objects, the R-Tree will excel. Finally, the MIR^2-Tree generally performs better than the IR^2-Tree; however, the MIR^2-Tree is expensive to maintain. Hence, for frequently updated datasets, IR^2-Tree is the choice.

VII. RELATED WORK

Nearest Neighbor Queries

Answering *k*-nearest neighbor queries on a spatial database is a classical database problem. Most methods use indices built on the data to assist the k-NN search. Perhaps the most widely used algorithm is the branch-and-bound algorithm [RKV95] which traverses an R-tree [Gut84] while maintaining a list of *k* potential nearest neighbors in a priority queue. There have also been attempts to use range queries to solve the k-NN search problem, such as the one proposed by Korn et al. [KSF+96]. The basic idea is to use a range query to retrieve the potential k-NNs. This algorithm is further extended by improving the region estimation [CG99], and by a better search technique of the k-NN in the region [SK98].

Before the incremental NN algorithm by Hjaltason and Samet [HS99] for R-Trees, used in this paper was developed, the problem of incremental NN was tackled for three different data structures: [Bro90,Hen94,HS95] operated on k-d trees, LSD-Tree, and PM quadtree respectively. All algorithms have similar principles and mainly differ in the data structures used during execution. Recently there has also been work on continuous k-NN queries [TPS02,XMA05] which find continuously the k nearest objects to a query point. Park and Kim [PK03] independently developed a technique similar to ours to answer NN queries using a combination of an R-tree and multiple S-trees, one for every non-spatial attribute.

Signature Files

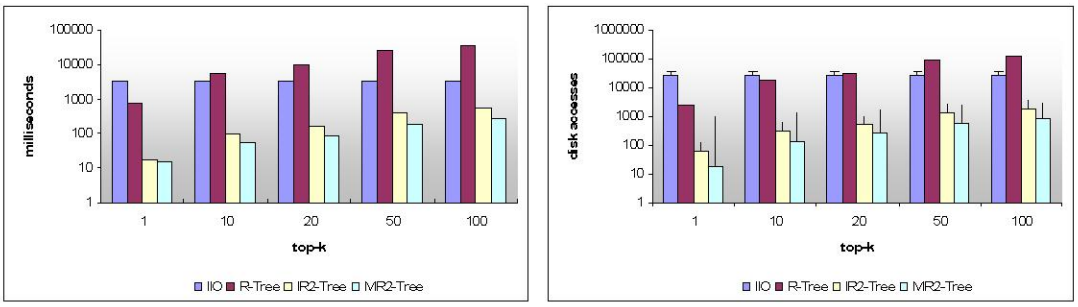
Signature files were introduced by Faloutsos and Christodoulakis [FC84,FC85,Fal85] as a method to efficiently search a collection of text documents. Lee et al. [LKP95] present methods to build structures on top of a signature file. In this work we view the document describing a spatial object as a text block in their notation and build similar structures on top of this set of objects. In particular, we adopt the idea of an indexed descriptor file structure [PBC80] (S-Tree [Dep86] is a variant of an indexed descriptor), which is a tree where the lowest level consists of block signatures. These are superimposed codes obtained from the text blocks. A group of *b* signatures at the *i*-th level is superimposed together to form a signature at the *(i-1)*-th level. The signatures of each level have the same length. Similarly, in our IR^2-Tree, the parent has a signature that superimposes (binary ORs) the signatures of the children.

Finally, when building an indexed descriptor file, we expect the top levels to have more 1's due to the larger number of words in their subtrees, which in turn leads to more false positives. The principle of the multi-level superimposed coding was proposed [CS89,DR83] as a solution to this problem, where higher levels have longer signatures. This principle allows fewer false positives by incurring a space overhead. However, this makes updates on the underlying documents expensive to maintain.

Top-k queries

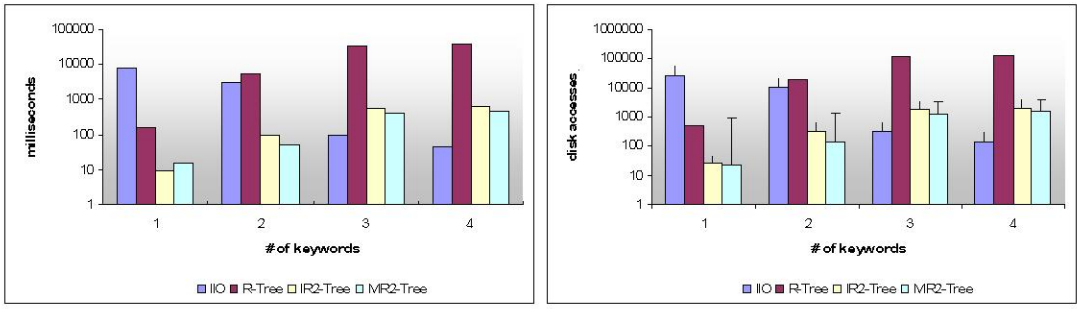
Top-*k* query works [Fag01, BGM02] handle the aggregation of attribute values of objects in the case where the attribute values lie in different sources. For example [BGM02] consider the problem of ordering a set of restaurants by distance and price. They present an optimal sequence of random or sequential accesses on the sources (e.g., Zagat for price and Mapquest for distance) in order to compute the top-*k* restaurants. They view the sources as black boxes in contrast to our work where we assume full access which allows us to build an IR^2-Tree.

Zhou et al. [ZXW+05] present techniques to combine an inverted index with an R* tree to answer Web queries with spatial constraints on pages with spatial information. Their experiments show that an inverted index whose keyword lists are organized as R* trees has the best performance. However, their algorithms are not top-*k*, that is, they require the spatial area as an input.



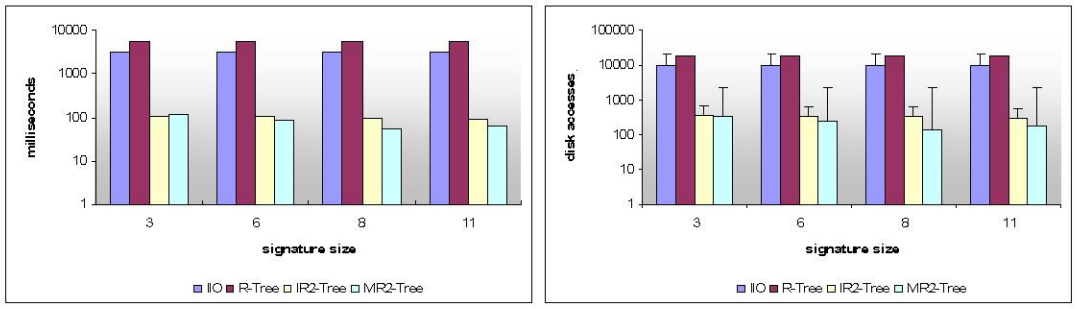
(a) Execution Time (b) Object Accesses

Figure 12: Results for varying k (top-k) searches for the Restaurants dataset



(a) Execution Time (b) Object Accesses

Figure 13: Results for varying number of keywords for the Restaurants dataset



(a) Execution Time (b) Object Accesses

Figure 14: Results for varying signature lengths (in bytes) for the Restaurants dataset

Furthermore, they do not scale well for multiple keywords since multiple R*-trees must be traversed and intersected (a combining algorithm is not presented). Vaid et al. [VJJS05] and Martins et al. [MSA05] present techniques to combine the output of a text and a spatial index to answer a spatial keyword query. These techniques are very similar to the baseline algorithms we use. However, they do not consider combining these indexes in a single structure like our IR^2-tree. Further, Vaid et al. [VJJS05] use a grid-based distribution of the spatial objects.

VIII. CONCLUSIONS

In this paper we introduced the problem of spatial keyword search and explained the performance limitations of current approaches. We proposed a solution which is dramatically faster than current approaches and is based on a combination of R-Trees and signature files techniques. In particular we introduced the IR^2-Tree and showed how it is maintained in the presence of data updates. An efficient incremental algorithm was presented that uses the IR^2-Tree to answer spatial keyword queries. We experimentally evaluated our technique, which proved its superior performance.

IX. REFERENCES

[Bro90] A. J. Broder. Strategies for efficient incremental nearest neighbor search. In Pattern Recognition, 23(1–2):171–178, January 1990.

[BGM02] Nicolas Bruno, Luis Gravano, Amélie Marian. Evaluating Top-*k* Queries over Web-Accessible Databases., ICDE 2002.

[CG99] S. Chaudhuri and L. Gravano. Evaluating top-*k* selection queries. In VLDB, 1999.

[CS89] W. W. Chang, Hans-Jörg Schek: A Signature Access Method for the Starburst Database System. VLDB 1989: 145-153

[CSM06] Yen-Yu Chen, Torsten Suel, Alexander Markowetz. Efficient Query Processing in Geographic Web Search Engines. SIGMOD 2006

[Dep86] U. Deppisch. S-Tree: A dynamic balanced signature index for office retrieval. In Proc. of the ACM Conf. on Research and Development in Information Retrieval, Pisa, 1986.

[DR83] Ron Sacks-Davis, Kotagiri Ramamohanarao: A two level superimposed coding scheme for partial match retrieval. Inf. Syst. 8(4): 273-289 (1983)

[Fag01] Ronald Fagin, Amnon Lotem, Moni Naor: Optimal Aggregation Algorithms for Middleware. In PODS 2001

[Fal85] Christos Faloutsos: Signature files: Design and Performance Comparison of Some Signature Extraction Methods. In SIGMOD Conference 1985

[FC84] Christos Faloutsos, Stavros Christodoulakis: Signature Files: An Access Method for Documents and Its Analytical Performance Evaluation. In ACM Trans. Inf. Syst. 2(4): 267-288(1984)

[FC85] Christos Faloutsos, Stavros Christodoulakis: Design of a Signature File Method that Accounts for Non-Uniform Occurrence and Query Frequencies. In VLDB 1985: 165-170

[FO95] C. Faloutsos, D. W. Oard. A survey of information retrieval and filtering methods. Technical Report. UMI Order Number: CS-TR-3514., University of Maryland at College Park, 1995

[Gut84] A. Guttman. R-Trees: a dynamic index structure for spatial searching. In SIGMOD Conference, 1984.

[Hen94] A. Henrich. A distance-scan algorithmfor spatial access structures. In Proceedings of the Second ACM Workshop on Geographic Information Systems, pages 136–143, Gaithersburg,MD, December 1994.

[HS95] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In Advances in Spatial Databases — Fourth International Symposium, pages 83–95, Portland, ME, August 1995.

[HS99] G.R. Hjaltason and H. Samet. Distance browsing in spatial databases. In ACM Transactions on Database Systems, Vol. 24, No. 2, 1999

[KSF+96] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast nearest neighbor search in medical image databases. In VLDB, 1996.

[LKP95] Dik Lun Lee, Young Man Kim, Gaurav Patel: Efficient Signature File Methods for Text Retrieval. Pages 423-435. TKDE Vol 7, Number 3, June 1995

[MC94] Malcolm Campbell. The Design of Text Signatures for Text Retrieval Systems. Technical Reports 1994

[MSA05] B. Martins, M. Silva, and L. Andrade. Indexing and ranking in Geo-IR systems. In Proc. of the 2nd Int. Workshop on Geo-IR (GIR), November 2005.

[NMN+00] G. Navarro, E. Silva de Moura, M. S. Neubert, N. Ziviani, R. A. Baeza-Yates: Adding Compression to Block Addressing Inverted Indexes. Inf. Retrieval 3(1): 49-77 (2000), 2000

[PBC80] John L. Pfaltz, William J. Berman, Edgar M. Cagley: Partial-Match Retrieval Using Indexed Descriptor Files. In Commun. ACM 23(9): 522-528 (1980)

[PK03] D. Park, H. Kim: An Enhanced Technique for k-Nearest Neighbor Queries with Non-Spatial Selection Predicates. In Multimedia Tools and Applications archive, Volume 19 , Issue 1 (January 2003), Pages: 79 – 103

[RKV95] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In SIGMOD Conference, 1995.

[Sal97] D. Salomon. Data Compression. The Complete Reference. Springer, New York, 1997.

[Sin01] A. Singhal: Modern Information Retrieval: A Brief Overview, Google, IEEE Data Eng. Bull, 2001

[SK98] T. Seidl and H.-P. Kriegel. Optimal multi-step k-nearest neighbor search. In SIGMOD Conference, 1998.

[TPS02] Yufei Tao, Dimitris Papadias, and Qiongmao Shen. Continuous Nearest Neighbor Search. In VLDB, 2002.

[VJJS05] S. Vaid, C. B. Jones, H. Joho, and M. Sanderson. Spatio-textual indexing for geographical search on the web. SSTD 2005.

[XMA05] Xiaopeng Xiong, Mohamed F. Mokbel, Walid G. Aref: SEA-CNN: Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-temporal Databases. In ICDE 2005

[ZMR98] J. Zobel, A. Moffat, K. Ramamohanarao: Inverted Files Versus Signature Files for Text Indexing. In ACM Trans. Database Syst. 23(4): 453-490 (1998)

[ZXW+05] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W. Ma. Hybrid index structures for location-based web search. ACM CIKM 2005