Citation in BibTeX format

RESEARCH-ARTICLE

Efficient retrieval of the top-k most relevant spatial web objects

GAO CONG, Aalborg University, Aalborg, Nordjylland, Denmark

CHRISTIAN S. JENSEN, Aalborg University, Aalborg, Nordjylland, Denmark

DINGMING WU, Aalborg University, Aalborg, Nordjylland, Denmark

Efficient Retrieval of the Top-k Most Relevant Spatial Web Objects

Gao Cong   Christian S. Jensen   Dingming Wu
Department of Computer Science
Aalborg University, Denmark
{gaocong, csj, dingming}@cs.aau.dk

ABSTRACT

The conventional Internet is acquiring a geo-spatial dimension. Web documents are being geo-tagged, and geo-referenced objects such as points of interest are being associated with descriptive text documents. The resulting fusion of geo-location and documents enables a new kind of top-k query that takes into account both location proximity and text relevancy. To our knowledge, only naive techniques exist that are capable of computing a general web information retrieval query while also taking location into account.

This paper proposes a new indexing framework for location-aware top-k text retrieval. The framework leverages the inverted file for text retrieval and the R-tree for spatial proximity querying. Several indexing approaches are explored within the framework. The framework encompasses algorithms that utilize the proposed indexes for computing the top-k query, thus taking into account both text relevancy and location proximity to prune the search space. Results of empirical studies with an implementation of the framework demonstrate that the paper's proposal offers scalability and is capable of excellent performance.

1. INTRODUCTION

Driven in part by the emergence of the mobile Internet, the conventional Internet is acquiring a geo-spatial dimension. On the one hand, many (geo-referenced) points of interest—e.g., stores, tourist attractions, hotels, entertainment services, public transport, and public services—are being associated with descriptive text documents. On the other hand, web documents are increasingly being geo-tagged.

This fusion of geo-location and documents enables queries that take into account both location proximity and text relevancy. One study has found that about one fifth of web search queries are geographical and have local intent, as determined by the presence of geographical terms such as place names and postal codes [25]. Indeed commercial search engines have started to provide location-based services, such as map services, local search, and local advertisements. For example, Google Maps supports location-aware text retrieval queries. Additional examples of location-based services include online yellow pages.

This paper considers a new kind of top-k query that takes into account both location proximity and text relevancy for points of interest with associated text. An example query may request a "good micro-brewery that serves pizza" and that is close to the user's hotel. We call this type of query a *location-aware top-k text retrieval* (LkT) query. The answer to such a top-k query is a list of k objects ranked according to a ranking function that combines their distances to the query location and the relevance of their textual descriptions to the query phrase. The LkT query is different from the query that retrieves relevant documents within a geographic range.

In the paper, we compute the text relevancy of a query result by means of language models and a probabilistic ranking function that have sound foundations in statistical theory and have performed well empirically in many information retrieval tasks [22, 30].

The LkT query poses new challenges for both existing spatial database and existing information retrieval techniques that have been developed separately. The research in spatial databases mainly focuses on highly structured, map-based geometric data and their attributes. In contrast, information retrieval research often treats location information as common keywords.

We are not aware of any published techniques that efficiently support the computation of the location-aware top-k text retrieval query considered in this paper. Some techniques [18, 28, 32] use an ad-hoc combination of nearest neighbor (NN) and keyword search techniques for location-aware query processing. For example, an R-tree is used to find the nearest neighbors and then for each neighbor, an inverted file is used to rank the objects according to text relevancy. This ad-hoc combination cannot easily be applied to process the LkT query since it is difficult to determine in advance the number of nearest neighbors needed to obtain the top-k results ranked by a combination of distance proximity and text relevancy. A recent proposal [7] integrates the R-tree with signature files. However, this proposal is not applicable to the LkT query (which is a ranking query, not a Boolean query) mainly due to the use of signature files, which cannot sensibly handle ranked text retrieval [33].

In this paper, we propose a new indexing framework for processing the location-aware top-k text retrieval (LkT) query. This framework integrates the inverted file for text retrieval and the R-tree for spatial proximity querying to obtain an Inverted file R-tree. Within the framework, an index approach called the IR-tree is proposed that is essentially an R-tree extended with inverted files. An associated algorithm is proposed for the processing of the LkT query that is able to prune the search space by simultaneously making use of both spatial proximity and text relevancy.

Each node of the IR-tree records a summary of the location information and the textual content of all the objects in the sub-tree rooted at the node. The query processing algorithm utilizes the location index information to estimate the spatial distance of a query

to the objects in the node's sub-tree, and it uses the text index to estimate the text relevancy scores for these objects.

We also explore a variant of the IR-tree that incorporates document similarity when computing *Minimum Bounding Rectangles* (MBR), yielding a new index, called the DIR-tree. While the IR-tree considers only location information when generating its MBRs, the DIR-tree takes into account both location information and document similarity. The IR-tree can be seen as a special case of the DIR-tree. To further improve the performance of the IR-tree and the DIR-tree, we cluster the documents attached to spatial objects. In an index node, tighter text relevancy scores can be estimated for a group of similar documents than for diverse documents that belong to different categories.

In summary, the paper's contribution is threefold. First, we introduce a new type of location-aware top-k text retrieval queries, LkT queries, that returns objects ranked according to a linear interpolation function that combines normalized location proximity and text relevancy.

Second, to efficiently process the query, we propose a new indexing framework that integrates location indexing and text indexing, and we develop an IR-tree and an associated algorithm for processing the LkT query. A variant of the IR-tree, the DIR-tree, is proposed to incorporate document similarity when computing Minimum Bounding Rectangles. We also exploit document clustering to improve the indexing framework.

Third, we conduct extensive experiments to evaluate the paper's proposals. Results of empirical studies with implementations of the proposed techniques demonstrate that the paper's proposals offer scalability and are capable of excellent performance.

The rest of this paper is organized as follows. Section 2 formally defines the location-aware top-k text retrieval problem. Section 3 presents the indexing framework for processing the LkT query. Section 4 proposes two methods for enhancing the framework. We report on a performance evaluation in Section 5. Finally, we cover related work in Section 6 and offer conclusions and research directions in Section 7.

2. PRELIMINARIES

We proceed to describe the problem addressed by the paper and then present baseline solutions that utilize existing techniques.

2.1 Problem Statement

Let D be a spatial database. Each spatial object O in D is defined as a pair $(O.loc, O.doc)$, where $O.loc$ is a location descriptor in multidimensional space and $O.doc$ is a document (e.g., a dining menu) that describes the object (e.g., an Italian restaurant). We assume a two-dimensional geographical space composed of latitude and longitude, but the paper's proposals generalize to other multidimensional spaces of low dimensionality. Document $O.doc$ is represented by a vector in which each dimension corresponds to a distinct term in the document. The value of a term in the vector is computed by a language model [22] as follows:

$$\hat{p}(t|\theta_{O.doc}) = (1-\lambda)\frac{tf(t, O.doc)}{|O.doc|} + \lambda\frac{tf(t, Coll)}{|Coll|}, \quad (1)$$

where $tf(t, O.doc)$ is the number of occurrences of term t in document $O.doc$ and $tf(t, Coll)$ is the count of term t in the document collection $Coll$ of D; $tf(t, O.doc)/|O.doc|$ is the maximum likelihood estimate of term t in document $O.doc$ and $tf(t, Coll)/|Coll|$ is the maximum likelihood estimate of term t in collection $Coll$; and λ is a smoothing parameter of the Jelinek-Mercer smoothing method.

Smoothing is a common practice for language models and is important for retrieval performance. While comparing a language model with the TF.IDF scheme, smoothing plays an IDF-like role. For ease of understanding, we use $tf(t, O.doc)$ to represent the weight of term t in the running example of the paper; however, we use language models in the experiments.

Intuitively, a *location-aware top-k text retrieval* (LkT) query retrieves k objects in database D for a given query Q such that their locations are the closest to the location specified in Q and their textual descriptions are the most relevant to the keywords in Q. Formally, given a query $Q = (loc, keywords)$ where $Q.loc$ is a location descriptor and $Q.keywords$ is a set of keywords, the objects returned are ranked according to a ranking function $f(D_\varepsilon, P(Q.keywords|O.doc))$, where D_ε is the Euclidian distance between Q and O and $P(Q.keywords|O.doc)$ is the probability of generating query $Q.keywords$ from the language models of the documents, which will be used to rank the objects.

Specifically, given a query Q and a document $O.doc$, the ranking function for the query likelihood language model is as follows:

$$P(Q.keywords|O.doc) = \prod_{t \in Q.keywords} \hat{p}(t|\theta_{O.doc}) \quad (2)$$

Problem Statement: We tackle the problem of efficiently answering LkT queries, i.e., given a query Q, we retrieve a ranked list of k objects according to their ranking scores as computed by the ranking function $f(\cdot, \cdot)$ introduced above. The paper's proposals are applicable to a wide range of ranking functions, namely all functions that are monotone with respect to distance proximity $f(D_\varepsilon)$ and text relevancy $P(Q.keywords|O.doc)$.

In this paper, we follow existing work and use linear interpolation [18]. Specifically, we derive a ranking function as a linear interpolation of normalized factors for ranking an object O with regard to a query Q:

$$D_{ST}(Q, O) = \alpha\frac{D_\varepsilon(Q.loc, O.loc)}{maxD} +$$
$$(1-\alpha)(1 - \frac{P(Q.keywords|O.doc)}{maxP}), \quad (3)$$

where $\alpha \in (0, 1)$ is a parameter used to balance spatial proximity and text relevancy; the Euclidian distance between Q and O, $D_\varepsilon(Q.\,loc, O.loc)$, is normalized by $maxD$, which can be, e.g., the maximal distance between two objects in D; and $maxP$ is used to normalize the probability score into the range from 0 to 1 and is computed by:

$$\prod_{t \in Q.keywords} \max_{O' \in D} \hat{p}(t|O'.doc),$$

where the idea is to use the maximal language model of each word to compute an upper bound on the probability value. Note that the lower the score computed by ranking function, the better.

The parameter α in Equation 3 allows users to set their preferences between text relevancy and location proximity at query time. Note that this study focuses on efficient solutions, not on new effective ranking functions.

Example 2.1: Figure 1 describes eight spatial objects $O_1 \dots O_8$, and Equation 4 shows a term-by-document matrix M of the documents of the eight objects. In the matrix, rows and columns represent terms and documents, respectively. For example, the matrix shows that document $O_1.doc$ contains the term *Chinese* five times and the term *restaurant* five times.

$$M = \begin{matrix} & O_1.doc & O_2.doc & O_3.doc & O_4.doc & O_5.doc & O_6.doc & O_7.doc & O_8.doc \\ Chinese \\ Spanish \\ restaurant \\ food \end{matrix} \begin{pmatrix} 5 & 0 & 7 & 0 & 4 & 0 & 1 & 0 \\ 0 & 5 & 0 & 0 & 0 & 4 & 1 & 3 \\ 5 & 5 & 0 & 7 & 4 & 3 & 4 & 3 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \quad (4)$$
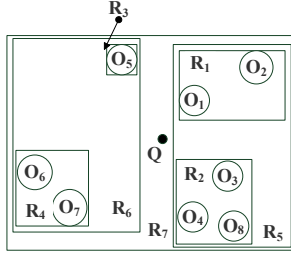
Figure 1: Eight Objects and Their Bounding Rectangles

Given a query Q with location $Q.loc$ as shown in Figure 1 and $Q.keywords$ = (*Chinese restaurant*), object O_1 is the result of the top-1 query according to Equation 4 ($\alpha = 0.3$). □

2.2 Baseline Algorithms

Before proceeding to present the proposed solution, we discuss how to exploit existing techniques for the processing of LkT queries.

No baseline algorithm exists for LkT queries. A straightforward baseline is to adapt an existing approach [18], thus computing the text relevancy using an inverted file and computing location proximity using an R-tree separately for all objects and then combining them to obtain the top-k objects. This is not efficient. The main difficulty is to benefit from both the inverted file and the R-tree.

The two new baseline algorithms presented next are named based on their underlying data structures: Inverted File Only (IFO) and R-tree plus Inverted File (RIF). Both are inspired by the Threshold algorithm [9].

Baseline 1: IFO. The idea is to utilize the inverted file to compute the text relevancy scores of all objects (corresponding to the right operand of the operator "+" in Equation 3), thus obtaining a list $IRRanking$ that ranks them in ascending order of their scores. The list is then scanned to compute the spatial proximity to the query until further scanning will not generate top-k results. This algorithm uses the inverted file only.

The tricky part is when to stop scanning. During a scan, the algorithm keeps track of the combined ranking score (defined in Equation 3; the lower the score, the better) of the current k'th object, denoted by $threshold$. For a new object T, if its $IRRanking$ score is bigger than $threshold$, the algorithm stops since all objects after T in $IRRanking$ will have a score that exceeds $threshold$; otherwise, we retrieve its location, compute its combined ranking score (Equation 3), and compare with $threshold$ to determine whether $threshold$ needs to be updated.

Baseline 2: RIF. This algorithm uses an R-tree and an inverted file in two stages. The inverted file is used for computing the list $IRRanking$ as for IFO. The algorithm then incrementally finds nearest neighbors [15] using the R-tree and checks the text relevancy scores of objects in $IRRanking$.

In the process, the algorithm keeps track of the minimum text relevancy score in $IRRanking$, denoted by $MinTR$, that has not been "seen" so far, and the combined ranking score (Equation 3) of the current k'th object, denoted by $threshold$.

For a newly "seen" object with spatial distance $dist$, if the combined score computed from $dist$ and the current $MinTR$ exceeds $threshold$, the algorithm stops since it is guaranteed that all "unseen" objects will not have lower scores than the current k'th object (and thus cannot be in the result).

3. HYBRID INDEXING FOR LOCATION-AWARE TEXT RETRIEVAL

We present a framework that integrates the R-tree and the inverted file into a new index, the *Inverted File R-tree* (IR-tree) and

that includes an algorithm for processing LkT queries using the IR-tree.

3.1 Hybrid Index Framework: The IR-Tree

The R-tree [13] is arguably the dominant index for spatial queries, and the inverted file is the most efficient index for text information retrieval [33]. These were developed separately and for different kinds of queries.

We aim to develop an approach that is able to leverage both techniques for the efficient processing of LkT queries. To achieve this goal, a simple approach is to use the inverted file (resp. the R-tree) to generate a number of top candidate objects based on text relevancy (resp. spatial proximity) and then compute the spatial distances (resp. text relevancy) of the candidate objects using the other index. However, this approach is not efficient since there is no sensible way to determine the number of candidate objects needed from the first step in order to ensure that k top-k objects are found in the end. Instead, we propose a hybrid indexing structure, the IR-tree, that utilizes both indexing structures in a combined fashion.

The IR-tree is essentially an R-tree, each node of which is enriched with reference to an inverted file for the objects contained in the sub-tree rooted at the node.

In the IR-tree, a leaf node N contains a number of entries of the form $(O, rectangle, O.di)$, where O refers to an object in database D, $rectangle$ is the bounding rectangle of object O, and $O.di$ is the identifier of the document of object O. A leaf node also contains a pointer to an inverted file for the text documents of the objects being indexed. The inverted file is stored separately, for two reasons: First, it is more efficient to store each inverted file contiguously, rather than as a sequence of blocks or pages that are scattered across a disk [33]. Second, the inverted file can be distributed across several machines while this is not easily possible for the R-tree [26].

An inverted file consists of the following two main components.

- A vocabulary for all distinct terms in a collection of documents.

- A set of posting lists, each of which relates to a term t. Each posting list is a sequence of pairs $\langle d, w_{d,t} \rangle$, where d refers to a document containing term t, and $w_{d,t}$ is the weight of term t in document d.

A non-leaf node R contains a number of entries of the form $(cp, rectangle, cp.di)$ where cp is the address of a child node of R, $rectangle$ is the Minimum Bounding Rectangle of all rectangles in entries of the child node, and $cp.di$ is the identifier of a pseudo document.

The pseudo document is an important concept in the hybrid index structure. It represents all documents in the entries of the child node, enabling us to estimate a bound of the text relevancy to a query of all documents contained in the subtree rooted at cp. The weight of each term t in the pseudo document referenced by $cp.di$ is the maximum weight of the term in the documents contained in the subtree rooted at node cp.

Example 3.1: Figure 2 illustrates the hybrid index for the eight objects in Figure 1. Table 1 shows the inverted files of the leaf nodes (*InvFile* 4, *InvFile* 5, *InvFile* 6, and *InvFile* 7 in Figure 2). Table 2 shows the content of the inverted files of the non-leaf nodes (*InvFile* 1, *InvFile* 2, and *InvFile* 3 in Figure 2). As a specific example, the weight of the term *restaurant* in entry $R2$ of node $R5$ is 7, which is the maximal weight of the term in the three documents in node $R2$. □

Vocabulary	InvFile 4 Posting lists	InvFile 5 Posting lists	InvFile 6 Posting lists	InvFile 7 Posting lists
Chinese	$\langle O_1.doc, 5\rangle$	$\langle O_3.doc, 7\rangle$	$\langle O_5.doc, 4\rangle$	$\langle O_7.doc, 1\rangle$
Spanish	$\langle O_2.doc, 5\rangle$	$\langle O_8.doc, 3\rangle$		$\langle O_6.doc, 4\rangle, \langle O_7.doc, 1\rangle$
restaurant	$\langle O_1.doc, 5\rangle, \langle O_2.doc, 5\rangle$	$\langle O_4.doc, 7\rangle, \langle O_5.doc, 4\rangle, \langle O_8.doc, 3\rangle$	$\langle O_6.doc, 3\rangle, \langle O_7.doc, 4\rangle$	
food		$\langle O_3.doc, 1\rangle, \langle O_4.doc, 1\rangle$		$\langle O_7.doc, 1\rangle$

Table 1: Posting Lists for InvFile 4, 5, 6, and 7

Vocabulary	InvFile 2 Posting lists	InvFile 3 Posting lists	InvFile 1 Posting lists
Chinese	$\langle R_1.doc, 5\rangle, \langle R_2.doc, 7\rangle$	$\langle R_3.doc, 4\rangle, \langle R_4.doc, 1\rangle$	$\langle R_5.doc, 7\rangle, \langle R_6.doc, 4\rangle$
Spanish	$\langle R_1.doc, 5\rangle, \langle R_2.doc, 3\rangle$	$\langle R_4.doc, 4\rangle$	$\langle R_5.doc, 4\rangle, \langle R_6.doc, 4\rangle$
restaurant	$\langle R_1.doc, 5\rangle, \langle R_2.doc, 7\rangle$	$\langle R_3.doc, 4\rangle, \langle R_4.doc, 4\rangle$	$\langle R_5.doc, 7\rangle, \langle R_6.doc, 4\rangle$
food	$\langle R_2.doc, 1\rangle$	$\langle R_4.doc, 1\rangle$	$\langle R_5.doc, 1\rangle, \langle R_6.doc, 1\rangle$

Table 2: Posting Lists for InvFile 1, 2, and 3

We proceed to present an important metric, the *minimum spatial-textual distance* $MIND_{ST}$, which will be used in the query processing. Given a query Q and a node N in the hybrid index, the metric $MIND_{ST}$ offers a lower bound on the actual spatial-textual distance between query Q and the objects enclosed in the rectangle of node N. This bound can be used to order and efficiently prune the paths of the search space in the hybrid index.

Definition 1. The distance of a query point Q from a node N in the hybrid index, denoted as $MIND_{ST}(Q, N)$, is defined as follows:

$$MIND_{ST}(Q, N) = \alpha \frac{MIND_\varepsilon(Q.loc, N.rectangle)}{maxD} + (1-\alpha)(1 - \frac{\text{P}(Q.keywords|N.doc)}{maxP}), \quad (5)$$

where α, $maxD$, and $maxP$ are the same as in Equation 3; $\text{P}(Q.keywords|N.doc)$ is computed by Equation 2 replacing $O.doc$ by $N.doc$ (the pseudo document of node N); and $MIND_\varepsilon(Q.loc, N.rectangle)$ is the minimum Euclidian distance between $Q.loc$ and $N.rectangle$. $\square$

A salient feature of the proposed hybrid indexing structure is that it inherits the nice properties of the R-tree for query processing.

Theorem 3.1: Given a query point Q and a node N whose rectangle encloses a set of objects $SO = \{O_i, 1 \le i \le m\}$, the following is true:

$$\forall O \in SO \ (MIND_{ST}(Q, N) \le D_{ST}(Q, O))$$

Proof: Since object O is enclosed in the rectangle of node N, the minimum Euclidian distance between $Q.loc$ and $N.rectangle$ is no larger than the Euclidian distance between $Q.loc$ and $O.loc$:

$$MIND_\varepsilon(Q.loc, N.rectangle) \le D_\varepsilon(Q.loc, O.loc)$$

For each term t, $w_{N.doc,t}$ (the weight of the term in $N.doc$, which is the pseudo document of node N) is the maximum value $w_{O.doc,t}$ of all the documents in node N. Thus:

$$P(Q.keywords|N.doc) \ge P(Q.keywords|O.doc)$$

According to Equations 3 and 5, we have:

$$MIND_{ST}(Q, N) \le D_{ST}(Q, O),$$

thus completing the proof. $\square$

When searching the hybrid index for the k objects nearest to a query Q, one must decide at each visited node of the hybrid index which entry to search first. Metric $MIND_{ST}$ offers an approximation of the distance to every entry in the node and, therefore, can
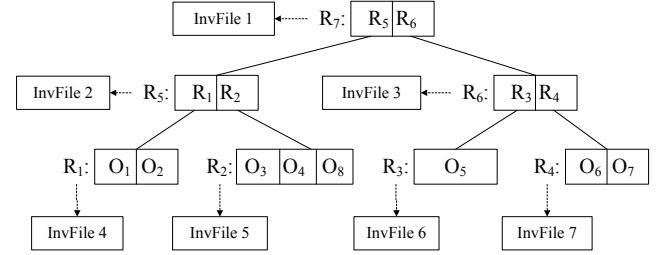

Figure 2: Hybrid Index in the Framework

be used to direct the search. Note that only the posting lists of keywords in query Q, but not all posting lists, are loaded into memory at a node to compute $MIND_{ST}$.

We next present an algorithm for building the IR-tree. The IR-tree is constructed by means of an insert operation that is adapted from the corresponding R-tree operation [13]. Algorithm 1 shows the Insert algorithm. It takes in two arguments, the MBR and the *document* of an object. It uses a standard implementation of the R-tree [13] with operations ChooseLeaf and Split.

Algorithm 1 Insert(MBR, *document*)

1: $N \leftarrow$ ChooseLeaf(MBR);
2: Add MBR to node N, add *document* to the inverted file of N;
3: **if** N needs to be split **then**;
4:     $\{O, P\} \leftarrow N.$Split();
5:     **if** N is root node **then**
6:         Initialize a new node M;
7:         Add O and P to node M and update the inverted file of O, P and M;
8:         Set M to the root node;
9:     **else**
10:         Ascend from N to the root, adjusting covering rectangles, updating the inverted file and propagating node splits as necessary;
11: **else if** N is not root **then**
12:     Update the covering rectangles and inverted files of the ancestor nodes of N;

We may characterize the additional disk storage required by the IR-tree by comparing with the original R-tree and the inverted file. The number of nodes in the IR-tree is the same as that of the original R-tree, and the size of the inverted files contained in all leaf nodes of the IR-tree is comparable with the original inverted file. The IR-tree needs additional space to store the inverted files in its non-leaf nodes, the sizes of which depend on the number of non-leaf nodes and the storage utilization of nodes. If the capacity of each node is 100 entries, the length of the posting list for one word is at most 100 in a non-leaf node, which is independent of the number of objects contained in the subtree rooted at the non-lead node.

The size of the inverted file at a non-leaf node is thus much smaller than that of the original inverted file. The paper's experimental study covers storage space.

3.2 Processing of LkT Queries

To process LkT queries with the hybrid index framework, we exploit the best-first traversal algorithm (e.g., [15]) for retrieving the top-k objects. With the best-first traversal algorithm, a priority queue is used to keep track of the nodes and objects that have yet to be visited. The values of D_{ST} and $MIND_{ST}$ are used as the keys of objects and nodes, respectively.

When deciding which node to visit next, the algorithm picks the node N with the smallest $MIND_{ST}(Q, N)$ value in the set of all nodes that have yet to be visited. The algorithm terminates when k nearest objects (ranked according to Equation 3) have been found. Algorithm 2 shows the pseudo-code.

Algorithm 2 LkT($Query, Index, k$)

1: $Queue \leftarrow$ NewPriorityQueue();
2: $Queue$.Enqueue($Index.RootNode$, 0);
3: **while not** $Queue$.IsEmpty() **do**
4:     $Element \leftarrow Queue$.Dequeue();
5:     **if** $Element$ is an object **then**
6:         **if not** $Queue$.IsEmpty() **and** $D_{ST}(Query, Object) >$ $Queue$.First().Key **then**
7:             $Queue$.Enqueue($Object, D_{ST}(Query, Object)$);
8:         **else**
9:             Report $Element$ as the next nearest object;
10:            **if** k nearest objects have been found **then**
11:                **break**;
12:    **else if** $Element$ is a leaf node **then**
13:        **for each** entry($Object$) in leaf node $Element$ **do**
14:            $Queue$.Enqueue($Object, D_{ST}(Query, Object)$);
15:    **else**
16:        **for each** entry($Node$) in node $Element$ **do**
17:            $Queue$.Enqueue($Node, MIND_{ST}(Query, Node)$);

We proceed to explain the algorithm and the use of the priority queue in the algorithm with an example.

Example 3.2: Consider the query Q ($Q.keywords$ = (*Chinese restaurant*) in Figure 1. We want to find the top-1 object. We give the Euclidean distances and D_{ST} (Equation 3) between query Q and all objects, as well as $MIND_{ST}$ (Equation 5) between Q and all bounding rectangles in Table 3. Note that Algorithm LkT only computes the distances between Q and the objects or rectangles traversed by the algorithm, not all the distance in Table 3. The algorithm uses a priority queue that contains the objects (resp. bounding rectangles) listed together with their D_{ST} (resp. $MIND_{ST}$) scores, in increasing order of the scores, with ties broken by the alphabetical ordering. The algorithm starts by enqueueing R_7 and then executes the following steps:

(1) Dequeue R_7, enqueue R_5 and R_6.
    Queue:{(R_5,0.05119),(R_6,0.269)}
(2) Dequeue R_5, enqueue R_1 and R_2.
    Queue:{(R_2,0.1048),(R_1,0.238),(R_6,0.269)}
(3) Dequeue R_2, enqueue O_3, O_4 and O_8.
    Queue:{(R_1,0.238),(R_6,0.269),(O_3,0.481),(O_4,0.517), (O_8,0.686)}
(4) Dequeue R_1, enqueue O_1 and O_2.
    Queue:{(O_1,0.238),(R_6,0.269),(O_3,0.481),(O_2,0.512), (O_4,0.517),(O_8,0.686)}
(5) Dequeue O_1. Report O_1 as the nearest object. Terminate. □

Objects	Dist.	D_{ST}	Rectangles	Dist.	$MIND_{ST}$
O_1	2	0.238	R_1	2	0.238
O_2	5	0.512	R_2	2	0.1048
O_3	6	0.481	R_3	4	0.368
O_4	7	0.517	R_4	5	0.42
O_5	3	0.53	R_5	0.5	0.05119
O_6	9	0.58	R_6	1	0.269
O_7	8	0.55	R_7	0	0
O_8	8	0.686			

Table 3: Distances Between Query Q and Objects and Bounding Rectangles in Figure 1

Observe that in the example, the algorithm does not traverse the entire tree in Figure 2 since the search space is being pruned. However, the algorithm still visits some nodes that contain no results. Recall that before reporting O_1, nodes R_7, R_5, R_2, and R_1 are visited. Since O_1 is the nearest object, ideally we would visit R_7, R_5, and R_1, but not R_2. The reason why R_2 is being visited is that the value $MIND_{ST}$ of R_2 is less than that of R_1.

In Table 1, we can see that objects O_3 and O_4 are very different. The term *Chinese* occurs seven times in $O_3.doc$, but does not appear in $O_4.doc$. On the other hand, the term *restaurant* occurs seven times in $O_4.doc$, but does not appear in $O_3.doc$. Since R_2 contains the two objects, the term frequencies of *Chinese* and *restaurant* in $R_2.doc$ are both seven (see Table 2). The reason for R_2 being highly relevant to the query is that it mixes objects of different types such that the weights of many terms are high in the pseudo document for R_2.

Based on this observation, we proceed to discuss how to enhance the query processing performance offered by the framework.

4. ENHANCED HYBRID INDEXING

We extend the hybrid indexing framework by incorporating document similarity, yielding an index called the DIR-tree (Document similarity enhanced Inverted file R-tree). The IR-tree can be viewed as a special case of the DIR-tree. This section also presents a clustering enhanced method for improving the indexing framework.

4.1 Incorporating Document Similarity

Like the R-tree, the IR-tree is built based on the heuristic of minimizing the area of each enclosing rectangle in the inner nodes. Thus, the tree aims to place nodes that are spatially close in the same higher-level node. However, the spatial objects considered in this paper also have associated documents, and the LkT query takes into account both location proximity and text relevancy.

Unlike the IR-tree, the DIR-tree aims to take both location and text information into account during tree construction, by optimizing for a combination of minimizing the areas of the enclosing rectangles and maximizing the text similarities between the documents of the enclosing rectangles.

We present an algorithm for building the DIR-tree in Section 4.1.1, and we cover query processing in Section 4.1.2.

4.1.1 The DIR-Tree

The leaf nodes of a DIR-tree have the same format as those of an IR-tree. A non-leaf node R contains a number of entries of the form ($cp, rectangle, cVectorId$) where cp is the address of a child node in the index, $rectangle$ is the MBR of the child node, and $cVectorId$ is the identifier of the centroid vector of all the vectors enclosed in the subtree rooted at cp. Let $V_c = (w_1, \ldots, w_T)$ be the centroid vector of a set of vectors $V_1, \ldots, V_n$. Then $V_c.w_i = max(V_1.w_i, \ldots, V_n.w_i)$.

To choose an appropriate insertion path for an object, the DIR-tree takes into account both the area parameter and the similarity

between the document of the object and the document vector representing the centroid of the documents associated with the objects in a node. We next describe how to incorporate document similarity.

Let $E_1, \ldots, E_p$ be the entries in the current node, and let O be the object to be inserted. In the R-tree, the area cost of inserting O into E_k, $1 \le k \le p$, is defined as follows:

$$AreaCost(E_k) =$$
$$area(E'_k.rectangle) - area(E_k.rectangle), \quad (6)$$

where $E'_k.Rectangle$ is the (possibly enlarged) version of rectangle E_k after inclusion of O.

Definition 2. The area cost extended with document similarity is defined as follows:

$$SimAreaCost(E_k, O) = (1 - \beta)\frac{AreaCost(E_k)}{maxArea}$$
$$+ \beta(1 - cosine(E_k.cVector, O.vector)) \quad (7)$$

$\square$

In this definition, document similarity is measured by the cosine similarity between two document vectors[1], $maxArea$, which is the area of the minimum bounding rectangle enclosing all objects, is used for normalization, and β is a parameter. Observe that if we set $\beta = 0$, the DIR-tree reduces to the IR-tree. In the other extreme, setting $\beta = 1$ means that we consider only document similarity when building a DIR-tree.

The insertion algorithm of the DIR-tree follows that of the IR-tree, with the exception of the specifics of functions ChooseSubtree and Split. Function ChooseSubtree is given in Algorithm 3.

Algorithm 3 ChooseSubtree

1: $N \leftarrow$ the root;
2: **loop**
3:     **if** N is a leaf node **then**
4:         Return N;
5:     **else**
6:         Choose the entry in N with the minimum value for $SimAreaCost(E_k, O)$, resolving ties by choosing the entry with the minimum value for $AreaCost(E_k)$;
7:         $N \leftarrow$ the child node pointed by the child pointer of the chosen entry;

Beginning at the root, the function finds at every level the most suitable subtree to accommodate the new entry until a leaf node is reached. Specifically, it chooses subtrees that minimize the value of $SimAreaCost$.

If ChooseSubtree reaches a leaf node with the maximum number of entries M, function Split must distribute the $M + 1$ rectangles between two nodes. We incorporate document similarity into the standard Quadratic Split algorithm [13]. Function Split is given in Algorithm 4.

4.1.2 *Query Processing*

We use Algorithm 1 for the processing of LkT queries on the DIR-tree. An example illustrates its benefits.

Example 4.1: We build a DIR-tree on the eight objects in Figure 1. The result is shown in Figures 3 and 4. Let a query $Q = (loc, keywords)$, where $Q.keywords = $ *Chinese restaurant* be given.

---

[1]Unlike in query processing, we do not use language models here since this would introduce asymmetry.

Algorithm 4 Split(N)

1: **for each** pair of entries E_i and E_j in node N **do**
2:     $R_{ij} \leftarrow$ compose a rectangle including E_i and E_j;
3:     $d_{ij} \leftarrow area(R_{ij}) - area(E_i) - area(E_j)$
4:     $cosSim_{ij} \leftarrow$ cosine similarity between documents of E_i and E_j
5:     $ineffi \leftarrow (1 - \beta)d + \beta(1 - cosSim)$;
6: Choose the pair with the largest $ineffi$ value to be the first elements of the two groups;
7: **loop** $\triangleright$ assign all entries in N to the two groups
8:     **if** all entries in N have been assigned to two groups **then**
9:         **break**
10:     **if** one group needs to include all remaining entries **then**
11:         Assign all remaining entries to it and **break**;
12:     Choose the next entry and add it to the group with the smaller $SimAreaCost()$, resolving ties by adding the entry to the group with the smaller $AreaCost()$;

If we apply Algorithm 1 to retrieve the top-2 most relevant objects, the results are O_1 and O_5, and the nodes R_7, R_5, and R_1 are visited. If we use instead the corresponding IR-tree in Figure 2, nodes R_7, R_5, R_6, R_1, and R_3 are visited. The DIR-tree thus yields better performance than the IR-tree for query Q. $\square$
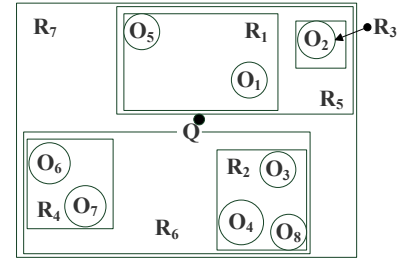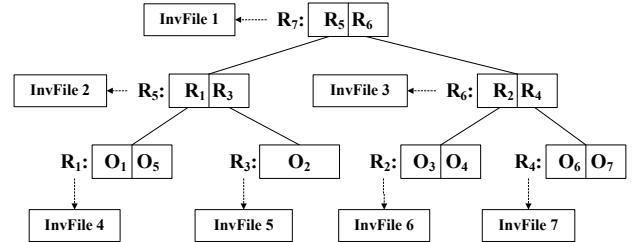

Figure 3: Objects and Bounding Rectangles

Figure 4: Structure of the DIR-Tree

4.2 Cluster Enhanced Method

We propose to enhance the hybrid indexing framework with clustering, which makes it possible to estimate tighter bounds at each tree node, thus improving query performance.

Spatial web objects often belong to different categories. For example, the geo-referenced points of interest may belong to specific categories, such as retail, accommodations, restaurants, and tourist attractions. Points of interest from different such categories may appear in the same node of the hybrid index, and thus two objects in the same node can be very different in terms of their document similarities.

The idea is to cluster objects into groups according to their corresponding documents. Each index node may then contain objects from different clusters. Instead of constructing a single pseudo document for each node, we construct a pseudo document for each

Vocabulary	InvFile 2 Posting lists	InvFile 3 Posting lists
Chinese	$\langle R_1.C_1.doc, 5\rangle, \langle R_2.C_3.doc, 7\rangle$	$\langle R_3.C_1.doc, 4\rangle, \langle R_4.C_1.doc, 1\rangle$
Spanish	$\langle R_1.C_2.doc, 5\rangle, \langle R_2.C_2.doc, 3\rangle$	$\langle R_4.C_2.doc, 4\rangle, \langle R_4.C_1.doc, 1\rangle$
restaurant	$\langle R_1.C_1.doc, 5\rangle, \langle R_1.C_2.doc, 5\rangle, \langle R_2.C_4.doc, 7\rangle, \langle R_2.C_2.doc, 3\rangle$	$\langle R_3.C_1.doc, 4\rangle, \langle R_4.C_1.doc, 4\rangle, \langle R_4.C_2.doc, 3\rangle$
food	$\langle R_2.C_3.doc, 1\rangle, \langle R_2.C_4.doc, 1\rangle$	$\langle R_4.C_1.doc, 1\rangle$

Table 4: Posting Lists for InvFile 2 and 3 in Figure 5

cluster in each node. Since objects within the same group are more similar than objects in different groups, the bounds estimated using clusters in a node will be tighter than those estimated for whole nodes. Therefore, we may expect the use of clustering to improve the query performance of both the IR-tree and the DIR-tree. We name the cluster enhanced IR-tree the CIR-tree and the cluster enhanced DIR-tree the CDIR-tree.

The CIR-tree. For convenience of presentation, we describe the proposed cluster enhanced method in the context of the CIR-tree; the method is equally applicable to the DIR-tree. Given any clustering of objects into n groups $C_1, \ldots, C_n$, we will form an inverted file incorporating the cluster information. We next present how to incorporate the cluster information into the leaf and non-leaf nodes in the hybrid index.

For a leaf node LN, we add a cluster label to each entry in the node. Each entry in a leaf node is then of the form $(O, O.rectangle, O.doc, O.C)$, where $O.C$ is the cluster label that indicates which cluster object O belongs to and the other items are as explained earlier. The inverted files for leaf nodes are organized by clusters, and we denote the set of documents belonging to cluster C_i in leaf node LN by $LN.C_i$.

Next, for a non-leaf node R, we need to construct a pseudo document for each cluster; the entries of such a document are quadruples: $(cp, rectangle, \{cp.Cluster\}, \{cp.Cluster.doc\})$, where cp and $rectangle$ are as in the IR-tree, $\{cp.Cluster\}$ is the set of cluster identifiers in the child node pointed to by cp, and $\{cp.Cluster.doc\}$ is a set of identifiers of pseudo documents of the clusters represented by $\{cp.Cluster\}$. Each node also contains a pointer to an inverted file that indexes all the pseudo documents referenced by document identifiers $cp.Cluster.doc$ of all entries of the non-leaf node.

The pseudo document of a cluster at a node is constructed in a bottom-up manner similarly to how we generate the pseudo documents in the IR-tree. Specifically, for each cluster C_i of a leaf node LN, we first construct a pseudo document denoted by $LN.C_i.doc$. For each term t, we choose the maximum value $w_{O.doc,t}$ of all the documents in each cluster C_i of a leaf node LN as the weight of the term in $LN.C_i.doc$, i.e.:

$$w(LN.C_i.doc, t) = \max_{O \in LN.C_i} w(O.doc, t).$$

Having constructed the pseudo documents of the clusters of the leaf nodes, we can construct pseudo documents of clusters of nodes at the upper levels from bottom to top. When we use pseudo documents at a lower level to construct pseudo documents at a higher level, identical clusters from different child nodes should be combined.

For instance, assume that a non-leaf node R_t contains two entries R_a and R_b (i.e., child nodes), where R_a includes two clusters $\{R_a.C_1, R_a.C_2\}$ and R_b includes two clusters $\{R_b.C_2, R_b.C_3\}$. The entry for parent node R_t should include three, not four, clusters, namely $\{R_t.C_1, R_t.C_2, R_t.C_3\}$. We combine $R_a.C_2$ and $R_b.C_2$ to obtain $R_t.C_2$. For each term t:

$$w_{R_t.C_2.doc,t} = \max\{w_{R_a.C_2.doc,t}, w_{R_b.C_2.doc,t}\}.$$

Example 4.2: Consider again the eight objects in Figure 1. We cluster these documents into four clusters: $C_1 = \{O_1, O_5, O_7\}$, $C_2 = \{O_2, O_6, O_8\}, C_3 = \{O_3\}, C_4 = \{O_4\}$. Figure 5 shows the CIR-tree for the eight objects. The contents of $InvFile$ 4, $InvFile$ 5, $InvFile$ 6, and $InvFile$ 7 are the same as in Table 1. The contents of $InvFile$ 1, $InvFile$ 2, and $InvFile$ 3 are given in Tables 4 and 5. □

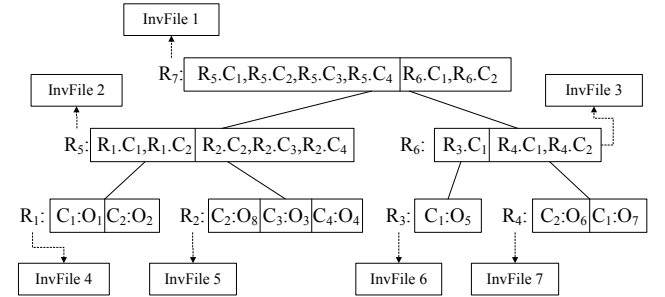| Vocabulary	InvFile 1 Posting lists
Chinese	$\langle R_5.C_3.doc, 7\rangle, \langle R_5.C_1.doc, 5\rangle, \langle R_6.C_1.doc, 4\rangle$
Spanish	$\langle R_5.C_2.doc, 5\rangle, \langle R_6.C_2.doc, 4\rangle, \langle R_6.C_1.doc, 1\rangle$
restaurant	$\langle R_5.C_4.doc, 7\rangle, \langle R_5.C_1.doc, 5\rangle, \langle R_5.C_2.doc, 5\rangle,$ $\langle R_6.C_1.doc, 4\rangle, \langle R_6.C_2.doc, 3\rangle$
food	$\langle R_5.C_3.doc, 1\rangle, \langle R_5.C_4.doc, 1\rangle, \langle R_6.C_1.doc, 1\rangle$

Table 5: Posting Lists of InvFile 1 in Figure 5

Figure 5: Structure of the CIR-Tree

Query processing. The algorithm for processing LkT queries on the CIR-tree is almost the same as Algorithm 2, the exception being that line 17 in Algorithm 2 is replaced by the following lines.

Replacement to Line 17 in Algorithm 2:

(20.1) $minD = min_{(1 \leq i \leq m)} MIND_{ST}(Query, Node.C_i);$
(20.2) $Queue.$Enqueue$(Node, minD);$
　　　　(/*m is the number of clusters in each entry*/)

The key of an entry in the priority queue is the minimum value of $MIND_{ST}(Query, Node.C_i)$ among all clusters.

Example 4.3: Consider the query in Example 4.2. The algorithm for processing the query using the CIR-tree starts by enqueueing R_7 and then executes the following steps:

(1) Dequeue R_7, enqueue R_5 and R_6.
    Queue:$\{(R_5, 0.1845), (R_6, 0.296)\}$
(2) Dequeue R_5, enqueue R_1 and R_2.
    Queue:$\{(R_1, 0.238), (R_6, 0.296), (R_2, 0.338)\}$
(3) Dequeue R_1, enqueue O_1 and O_2.
    Queue:$\{(O_1, 0.238), (R_6, 0.269), (R_2, 0.338), (O_2, 0.512)\}$
(4) Dequeue O_1. Report O_1 as the nearest object. Terminate.

Observe that before reporting O_1, nodes R_7, R_5, and R_1 are visited. Compared with the algorithm using the IR-tree, fewer nodes

are visited (recall that the algorithm using the IR-tree visited R_2) because the CIR-tree provides tight bounds. $\square$

Clustering objects. Ideally, we would find clusters that are optimal for the running time of Algorithm 2. Consider a query Q for the top-k objects on dataset D and a set of document clusters $C_1, \ldots, C_n$ on the documents in D. Let ScanTime($C_1, \ldots, C_n$, D, k, Q) be the number of objects checked by Algorithm 2 before returning the top-k objects. Given (D, k, Q), one would like to find n clusters $C_1, \ldots, C_n$ such that ScanTime($C_1, \ldots, C_n$, D, k, Q) is minimized.

It is well-known that finding clusters of points that minimize the diameter within a metric space is NP-hard. Although minimizing the diameter of a cluster in our problem usually results in a tight upper bound of each cluster for a query, this does not immediately imply that finding a clustering solution that minimizes ScanTime() is NP-hard, since ScanTime() does not directly correspond to the diameters of clusters.

Theorem 4.1: The problem of finding a clustering solution that minimizes ScanTime($C_1, \ldots, C_n$, D, k, Q) is NP-hard.

Proof: The proof is by reduction from the *bin packing problem*, which is NP-hard [12]. Consider $k = 1$ and let objects have the same location information. If the *IRScore* computed based on pseudo documents (i.e., upper bounds) is smaller than the *IRScore* of the top-1 document, Algorithm 2 will first scan the clusters containing the top-1 document to find the top-1 document, and it will then prune the clusters whose upper bound *IRScore* is lower than the *IRScore* of the top-1 document.

Assume that a parameter B is computed as the sum of the logarithmic value of the term language model of the top-1 result. Minimizing the scan time corresponds to the decision problem of assigning all objects to k clusters, such that for each cluster, $Score(C) \leq B$. Given an instance of the bin packing problem, each item corresponds to one object whose document contains a distinct term, each bin corresponds to a cluster, and the size of each item corresponds to the logarithmic value of the term language model. The query Q will contain all terms in documents in D. There is then a solution for the bin packing problem that packs all the items in k bins of size B if and only if there is a solution for our problem. $\square$

Given the above result, we must use a heuristic method for the clustering. One natural and simple approach is to use the k-means clustering algorithm.

Reducing the size of the cluster based indexes. Although the cluster enhanced CIR-tree and CDIR-tree are capable of returning tighter estimation bounds during query processing than their counterparts, the clustering requires additional space. We next introduce a mechanism for reducing the sizes of the CIR-tree and CDIR-tree.

The basic idea is that if the upper bound weight of a term is similar in each cluster, the algorithm will not benefit from the different weights of each cluster for the term. Instead, we will store the maximal weight of the term across all clusters. Put differently, if a term facilitates distinguishing among clusters, we record weights for all clusters as in the CIR-tree; otherwise, we record a single weight for all clusters as in the IR-tree. Specifically, for each term we compute the variance of the weights of the clusters in a node. If the variance is smaller than a pre-defined threshold ξ, we store a single weight for the term; otherwise, we store a weight for each cluster in the node.

Additional disk storage is needed by the CIR-tree (resp. the CDIR-tree) when compared with the IR-tree (resp. the DIR-tree). The size of the inverted files in the leaf nodes is the same; the additional storage is due to the non-leaf nodes. Let c be the number of

clusters. Each entry in a non-leaf node in the IR-tree corresponds to a pseudo document, while each entry in a non-leaf node in the CIR-tree corresponds to at most c (less than c when some clusters are empty) pseudo documents; a pseudo document in the CIR-tree may contain less words than the corresponding pseudo document in the IR-tree due to the clustering. Hence, the size of the inverted file in a non-leaf CIR-tree node is no more than c times of that of a non-leaf IR-tree node.

5. EXPERIMENTAL STUDY

We proceed to evaluate the LkT query performance of the four proposed hybrid indexes, including the IR-tree, the CIR-tree, the DIR-tree, and the CDIR-tree.

Algorithms. In addition to the proposed algorithms, we implemented the two baseline algorithms presented in Section 2.2. As Baseline 2 (RIF) outperforms Baseline 1 (IFO) significantly, we only report results for Baseline 1, making the figures more presentable. Note that both baseline algorithms need to compute the text relevancy score for every object. In our implementation, we use an accumulator for each object document, and all accumulators are memory resident. However, in our hybrid approaches, we only need accumulators for the objects in a tree node (100 at most in our experiments, to be explained). This means that the baseline algorithms will need more memory than the hybrid algorithms.

Data and queries. We use 7 datasets. We use a real spatial dataset containing 131,461 objects located in LA streets[2] and five categories of a real document dataset of 20 Newsgroups[3] to generate a dataset DATA1 by randomly selecting a document for a spatial object. The 20 Newsgroups consists of short user-generated content that aim to resemble text attached to points of interest in application like Google Maps (e.g., restaurant reviews).

DATA2 includes a real spatial dataset modeling the roads in California and documents from WEBSPAM-UK2007[4] that consists of a large number of real web documents. Table 6 lists additional properties of these two datasets.

Property	DATA1	DATA2
Total number of objects	131,461	2,249,727
Average number of unique words per object	112	429
Total number of unique words in dataset	30,616	2,899,175
Total number of words in dataset	14,809,845	965,132,883

Table 6: Dataset Properties

To evaluate scalability, we generate 5 datasets containing from 2 to 10 million data points: the locations are generated randomly in the space of DATA1, and a document selected at random from 20 Newsgroups is attached to a location.

We generate 4 query sets, in which the number of keywords is 1, 2, 3, and 4, respectively, in the space of DATA1, and we generate 4 similar query sets for the space of DATA2. Each set comprises 200 queries, and each query is randomly generated. We report average costs of the queries in each query set.

Setup. All index structures (IR-tree, CIR-tree, DIR-tree, CDIR-tree, baseline) are disk resident, and the page size is 4KB. The number of children of a node in the R-tree is computed given the fact that each node occupies a page. This translates to 100 children per node in our implementation. For the DIR-tree and the CDIR-tree, the default value for parameter β is set at 0.1. For the CIR-tree

---

[2] http://www.rtreeportal.org

[3] http://people.csail.mit.edu/jrennie/20Newsgroups

[4] http://barcelona.research.yahoo.net/webspam/datasets/uk2007

and the CDIR-tree, the default number of clusters is set at 5. All algorithms were implemented in Java, and an Intel(R) Core(TM)2 Duo CPU T7500 @2.20GHz with 2GB RAM was used for the experiments.

Three sets of experiments are carried out. The first evaluates the performance when varying the number k of requested results. The second evaluates the effect of the number of query keywords. The third evaluates the effect of the parameter α in the ranking function. We also evaluate the effect of the buffer, scalability, number of clusters, and the effects of other parameters. Unless stated otherwise, DATA1 us used.

Varying k in LkT. In this experiment we fix the number of query keywords at 2 and α (in Equation 3) at 0.3. Figures 6(a) and 6(b) show the results. All the four hybrid indexes significantly outperform the baseline approach for all values of k in terms of both runtime and I/O. The DIR-tree performs better than the IR-tree, and the cluster enhancement (CIR-tree and CDIR-tree) improves the performance of both the IR-tree and the DIR-tree. This is expected since the DIR-tree accounts for the similarity among documents of objects while the IR-tree does not. Because the LkT query references both location and keywords, the DIR-tree will prune the search space more effectively than the IR-tree. As expected, the runtime is proportional to the number of page accesses.
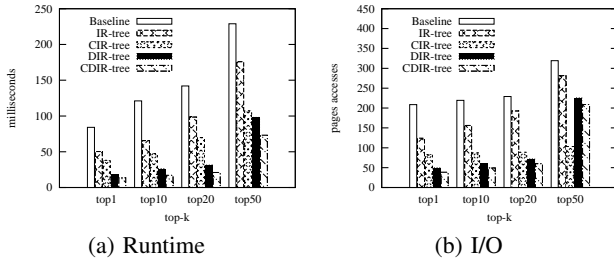

(a) Runtime   (b) I/O

Figure 6: Varying k

Varying the number of keywords. Here, we fix k at 10 and α at 0.3. In Figures 7(a) and 7(b), we can see that the hybrid indexes outperform the baseline approach. We note that the DIR-tree performs better than the IR-tree when the number of keywords is one and two, while the DIR-tree performs worse than IR-tree when the number of keywords is four. This may be because our document dataset contains five different topics. Queries may consist of keywords from different topics. Thus, when a query contains more keywords, the text relevancy of a query with each node of the DIR-tree is similar, which makes the text relevancy pruning less effective. It also can be seen from Figure 7(b) that the I/O cost of the DIR-tree is high for 4 keywords. The results also demonstrate that the cluster enhancements can improve the query performance of both the IR-tree and the DIR-tree for different numbers of keywords.

Varying α. Parameter α in Equation 3 allows users to set their preferences between text relevancy and spatial proximity. We fix k at 10 and the number of keywords at 2. Figures 8(a) and 8(b) show the results. A large α means that the spatial distance is more important, while a small α means that the keywords are more important. As expected, the IR-tree performs better for large α while the DIR-tree performs better for small α, since the DIR-tree takes into account document similarity and the benefit is significant when the text relevancy is given higher weight. As in the previous experiment, the cluster enhancement is effective.
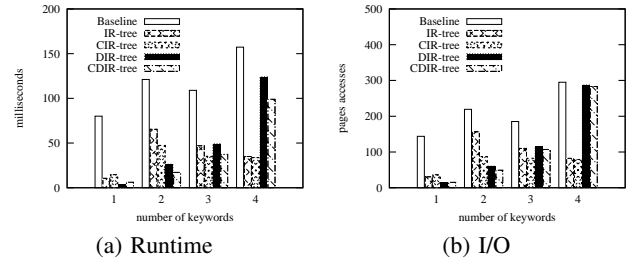

(a) Runtime   (b) I/O

Figure 7: Varying the Number of Keywords
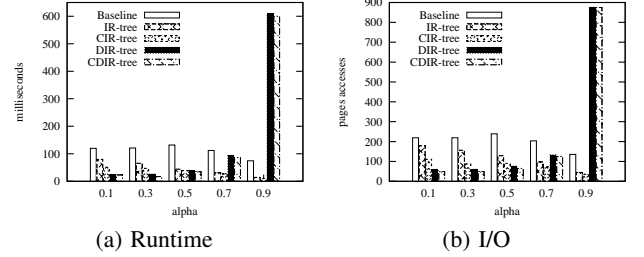

(a) Runtime   (b) I/O

Figure 8: Varying α

Effect of buffering. We use an LRU buffer, and we vary the buffer size from 0 to 36M, where 36M corresponds to about 20% of the size of the IR-tree. As shown in Figure 9, extra buffer space improves the I/O performance of all hybrid indexes. As expected, the improvement decreases as the buffer size increases. The runtime improvement is similar and so is omitted.
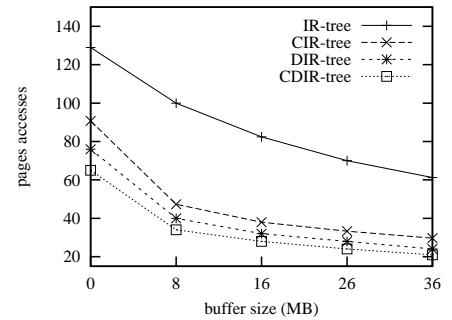

Figure 9: Varying the Buffer Size

Space requirements. Table 7 shows the total size of each index structure. The difference between the RIF and the IR-tree (resp. the DIR-tree) in the table shows the overhead introduced by the IR-tree (resp. the DIR-tree). As discussed in Section 3.1, the inverted file built in RIF is roughly the inverted files in the leaf nodes of the hybrid indexes. The overhead occurs because in the hybrid indexes, each non-leaf node also has an inverted file. The difference between the IR-tree (DIR-tree) and the CIR-tree (CDIR-tree) shows the storage overhead due to the cluster refined inverted file in non-leaf nodes in the CIR-tree. Note that the sizes of the inverted files in their leaf nodes are the same. We also note that the DIR-tree is smaller than the IR-tree. The possible reasons are twofold: 1) we find that the number of nodes in DIR-tree is smaller and the utilization of its nodes is higher, and 2) a non-leaf node in the DIR-tree is more likely to contain homogeneous documents and thus the size

of the inverted file attached to the non-leaf node will be relatively smaller.

RIF	IR-tree	CIR-tree	DIR-tree	CDIR-tree
88	157	234	109	162

Table 7: Index Structure Sizes (MB)

Varying threshold ξ**.** Parameter ξ, the variance threshold introduced in Section 4.2, enables reduction of the storage needed for the CIR-tree (and the CDIR-tree) at the expense of query efficiency. The CIR-tree (CDIR-tree) outperforms the IR-tree (DIR-tree) in terms of runtime, but needs more storage. Figure 10 shows the relative performance of the CIR-tree to the IR-tree in terms of runtime, the number of page accesses, and storage when we vary the variance threshold ξ (the value of the y-axis is the runtime, the number of page accessed and the storage of the CIR-tree divided by the runtime, the number of page accessed and the storage of the IR-tree, respectively). As expected, the bigger the parameter ξ is, the smaller the required storage and the lower the runtime are. When we increase ξ, the performance approaches that of the IR-tree ($\xi = \infty$); when we decrease ξ, the performance approaches that of the CIR-tree ($\xi = 0$). This enables balancing space and time.
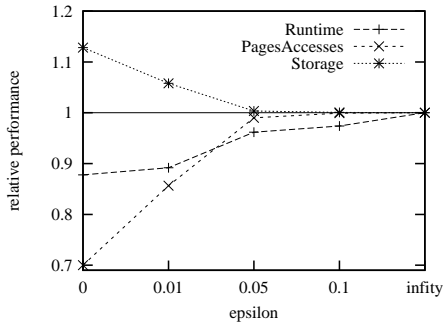

Figure 10: Varying ξ

Scalability. We run experiments on five datasets containing from 2 to 10 million objects. As shown in Figures 11(a) and 11(b), the runtime and page accesses increase with the size of the dataset, but the indexes scale (approximately) linearly.
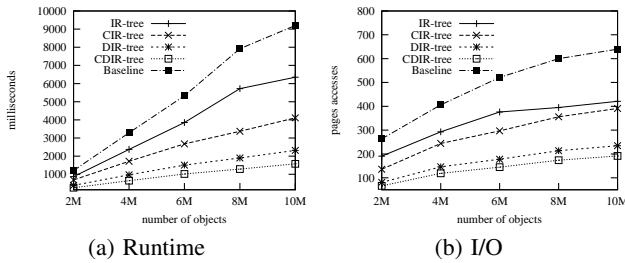

(a) Runtime   (b) I/O

Figure 11: Varying the Dataset Size

Varying the importance of document similarity when building DIR-trees. We vary the parameter β to build different DIR-trees. We set k at 10, α at 0.3, and the number of keywords at 2. Figures 12(a) and 12(b) show the performance of the different DIR-trees. In the extreme case of $\beta = 0$, the tree is actually an IR-tree. In the other extreme case of $\beta = 1$, only document similarity is

considered when building the DIR-tree. The performance is best at $\beta = 0.1$. For a specific application, we can find a good parameter value empirically. We have also found that if we set α at smaller values, i.e., giving text relevancy a higher weight, DIR-trees with large β perform better. We omit the details due to space limitations.
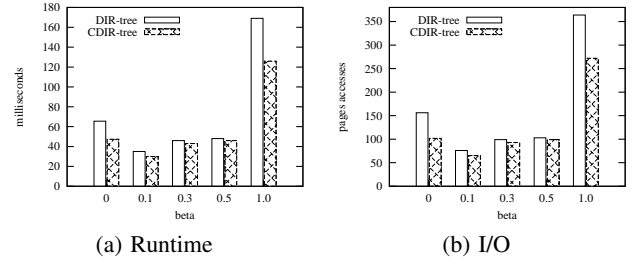

(a) Runtime   (b) I/O

Figure 12: Varying β

Varying the number of clusters. We fix the number of query keywords at 2 and α at 0.3. CIR-trees with 3, 5, 7, 9, 11, 15, 21, and 50 clusters are built. We use the k-means algorithm for clustering. The sizes of the resulting CIR-trees are shown in Table 8. The storage increases slightly with the number of clusters.

Figures 13(b) and 13(a) show the results of top-10, top-20, and top-50 queries. I/O decreases when the number of clusters increases from 3 to 50. Runtime decreases from 3 clusters to 9 clusters, but then increases from 11 clusters to 50 clusters. This behavior occurs because the time needed for processing clusters in non-leaf nodes counteracts the time saved by fewer page accesses due to more clusters. Page accesses decrease as the number of clusters increases. That is because more clusters offer tighter bounds, which are more effective for pruning. The results for the CDIR-tree are similar to those for the CIR-tree and are thus omitted.
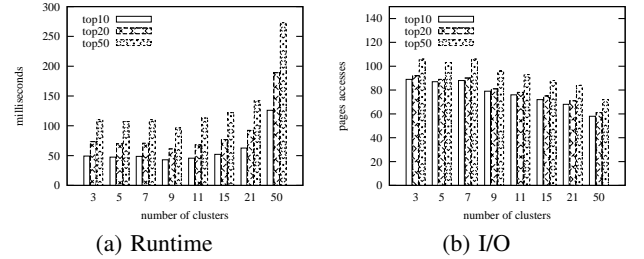

(a) Runtime   (b) I/O

Figure 13: Varying the Number of Clusters

# Clusters	3	5	7	9	11	15	21	50
Size	229	234	231	238	240	244	248	259

Table 8: Sizes of Different CIR-Trees (MB)

Experiments on DATA2. We have conducted the same extensive experiments on the DATA2 dataset as on DATA1. We observe qualitatively similar results on both datasets. Due to space limitations, we only report a subset of our results for DATA2. Figures 14(a) and 14(b) show the results of varying k while fixing the number of query keywords at 2 and α (in Equation 3) at 0.3. Figures 15(a) and 15(b) show the results of varying the number of keywords while fixing the number of requested objects k at 10 and α at 0.3. The results are consistent with those for DATA1.
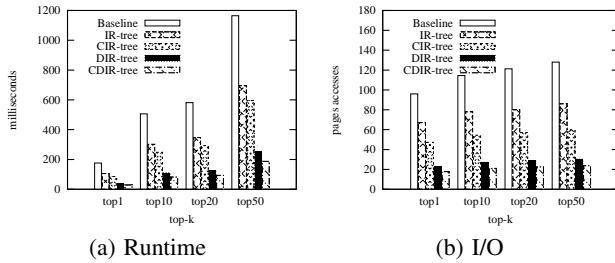
(a) Runtime   (b) I/O

Figure 14: Varying k
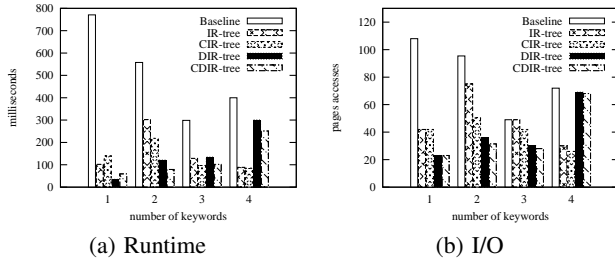

(a) Runtime   (b) I/O

Figure 15: Varying the Number of Keywords

To summarize, the experimental study shows that the proposed hybrid indexes significantly outperform the baseline. It also shows that the cluster enhancement indeed improves the performance of the hybrid indexes.

6. RELATED WORK

Nearest Neighbor and Top-k Queries. The processing of k-nearest neighbor queries (kNNs) in spatial databases is a classical subject. Most proposals use index structures to assist in the kNN processing. Perhaps the most influential kNN algorithm is due to Roussopoulos et al. [24]. In this solution, an R-tree [13] indexes the points, potential nearest neighbors are maintained in a priority queue, and the tree is traversed according to a number of heuristics. Other branch-and-bound methods modify the index structures to better suit the particular problem addressed [16, 29]. Hjaltason and Samet [15] propose an incremental nearest neighbor algorithm based on an R*-tree [4].

Our work is also related to top-k query processing [9, 17]. Fagin et al. [9] propose a class of algorithms known as threshold algorithms. These algorithms, like the ones proposed for information retrieval [2, 21], enable efficient computation of aggregate functions over multiple sorted lists. These algorithms can be easily integrated into the leaf-nodes in our framework (we need to process all entries in non-leaf nodes, so the threshold algorithm does not apply there); however, using them does not improve performance in our framework. The possible reason is that the posting lists in a leaf-node are short (limited by the capacity of a node).

Text Retrieval Queries. A variety of retrieval models have been proposed to meet different information retrieval needs, such as probabilistic models, the vector space model, the probabilistic similarity measure often referred to as the Okapi measure (or BM25) [23], and language models. The latter represent a relatively new approach, and they offer the best or competitive performance in many settings [6, 22].

Many different types of indexes have been proposed. The most efficient index structure for text retrieval is the inverted file [33], and many state-of-the-art large-scale IR systems such as web search engines employ inverted files for ranking-query evaluation. Alternative indexing techniques for text documents also exist, including suffix arrays [3] and signature files [10]. Zobel et al. [34] empirically show that signature files are not competitive with the inverted file for information retrieval queries.

To improve efficiency, a host of works develop effective heuristics for reducing query evaluation costs by reordering the inverted file according to frequency or contribution [2, 21]. There are also other techniques (e.g., [27]) that aim to increase query efficiency and compression techniques that aim to reduce storage costs (e.g., [11, 20]). These techniques can be applied to our framework; however, they are beyond the scope of this study.

Location-Aware Text Retrieval Queries. Commercial search engines such as Google and Yahoo! have introduced local search services that appear to focus on the retrieval of local content, e.g., related to stores and restaurants. However, the algorithms used are not publicized.

Much attention has been given to the problem of extracting geographic information from web pages (e.g., [1, 8, 19]). The extracted information can be used by search engines. McCurley [19] covers the notion of geo-coding and describes geographic indicators found in pages, such as zip codes and location names.

Recent studies that consider location-aware text retrieval constitute the work most closely related to this study. Zhou et al. [32] tackle the problem of retrieving web documents relevant to a keyword query within a pre-specified spatial region. They propose three approaches based on a loose combination of an inverted file and an R*-tree. The best approach according to their experiments is to build an R*-tree for each distinct keyword on the web pages containing the keyword. As a result, queries with multiple keywords need to access multiple R*-trees and to intersect the results. Building a separate R*-tree for each keyword also requires substantial storage.

Our hybrid indexing framework differs substantially from this indexing approach, although inverted files and R-trees are used in both approaches. Our approach incorporates the inverted file at each node of an R-tree such that both location and text information can be utilized to prune the search space at query time, while the approaches of Zhou et al. [32] adopt combinations that require query processing to occur in two stages: One type of indexing is used to filter web document in the first stage, and then the other index is employed in the second stage. This is similar in spirit to the baseline approach used in our experiments.

Next, our approach and their approaches target different kinds of queries: we focus on top-k queries while Zhou et al. aim to retrieve relevant documents within a given geographic region. We know of no way of adapting these approaches to process the top-k queries considered in this paper (without scanning all objects). In contrast, our framework can easily be extended to process the queries considered by Zhou et al.

Another study [5] addresses a problem similar to that of Zhou et al. [5]. In this study, a separate inverted file and a spatial indexing structure are built and used in two stages. Vaid et al. [28] also present techniques to combine the output of a text and a spatial index to answer a spatial keyword query in two stages. The aforementioned two differences between our approach and that of Zhou et al. also apply here.

Next, Hariharan et al. [14] address the problem of finding objects containing query keywords within a region. They present a hybrid indexing structure called the KR*-tree that consists of an R*-tree and an inverted file for the nodes of the R*-tree. The nodes of

the KR*-tree are virtually augmented with the sets of keywords that appear in the subtrees rooted at the nodes. At query time, the KR*-tree based algorithm finds the nodes that contain the query keywords and then uses these as candidates for subsequent search. This approach suffers from unnecessary overhead when there are many candidates.

Felipe et al. [7] propose an index structure that integrates signature files and the R-tree to enable keyword search on spatial data objects that each have a limited number of keywords. This approach needs to load the signature files of all words into memory when a node is visited, which incurs substantial I/O. Signature files are generally inferior to inverted files for general text retrieval [33]. The fact that there is no practical way of using signature files for handling ranked queries [33] renders it infeasible for this approach to support LkT queries that need to compute text relevancy scores (using language models).

Another hybrid indexing structure [31] combines the R*-tree and bitmap indexing to process the m-closest keyword query that returns the spatially closest objects matching m keywords. This approach exhibits the same problems as do signature-file based indexing [7].

Finally, Martins et al. [18] compute text relevancy and location proximity independently and then combine the two ranking scores. The baseline algorithms we investigate in this paper appear to be better than this approach.

7. CONCLUSIONS AND FUTURE WORK

This paper proposes a new indexing framework for location-aware top-k text retrieval. The framework integrates the inverted file for text retrieval and the R-tree for spatial proximity querying in a novel manner. Several hybrid indexing approaches are explored within the framework. The framework encompasses algorithms that utilize the proposed indexes for computing the top-k query, and it is capable of taking into account both text relevancy and location proximity to prune the search space at query time. Results of empirical studies with an implementation of the framework demonstrate that the paper's proposal offers scalability and is capable of excellent performance.

This work opens to a number of promising directions for future work. First, it is worth adapting existing optimization techniques developed for the inverted file (e.g., compression) and R-trees to the paper's setting. Second, it is of interest to develop algorithms for other type of queries, e.g., range queries, based on the hybrid index. Third, it would be interesting to understand how the top-k queries considered can best be processed if the spatial objects are constrained to a road network.

Acknowledgments

8. REFERENCES

[1] E. Amitay, N. Har'El, R. Sivan, and A. Soffer. Web-a-where: geotagging web content. In *SIGIR*, pp. 273–280, 2004.

[2] V. N. Anh, O. de Kretser, and A. Moffat. Vector-space ranking with effective early termination. In *SIGIR*, pp. 35–42, 2001.

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

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

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

[6] G. Cong, L. Wang, C.-Y. Lin, Y.-I. Song, and Y. Sun. Finding question-answer pairs from online forums. In *SIGIR*, pp. 467–474, 2008.

[7] I. De Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *ICDE*, pp. 656–665, 2008.

[8] J. Ding, L. Gravano, and N. Shivakumar. Computing geographical scopes of web resources. In *VLDB*, pp. 545–556, 2000.

[9] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[10] C. Faloutsos and S. Christodoulakis. Signature files: an access method for documents and its analytical performance evaluation. *ACM TODS*, 2(4):267–288, 1984.

[11] C. Faloutsos and H. V. Jagadish. Hybrid index organizations for text databases. In *EDBT*, pp. 310–327, 1992.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

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

[14] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems. In *SSDBM*, p. 16, 2007.

[15] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Trans. Database Syst.*, 24(2):265–318, 1999.

[16] N. Katayama and S. Satoh. The SR-tree: an index structure for high-dimensional nearest neighbor queries. In *SIGMOD*, pp. 369–380, 1997.

[17] N. Bruno, L. Gravano, and A. Marian. Evaluating top-k queries over web-accessible databases. In *ICDE*, pp. 369–380, 2002.

[18] B. Martins, M. J. Silva, and L. Andrade. Indexing and ranking in geo-IR systems. In *GIR*, pp. 31–34, 2005.

[19] K. S. McCurley. Geospatial mapping and navigation of the web. In *WWW*, pp. 221–229, 2001.

[20] A. Moffat and J. Zobel. Coding for compression in full-text retrieval systems. *Data Compression Conference*, pp. 72–81, 1992.

[21] M. Persin, J. Zobel, and R. Sacks-Davis. Filtered document retrieval with frequency-sorted indexes. *J. Am. Soc. Inf. Sci.*, 47(10):749–764, 1996.

[22] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *SIGIR*, pp. 275–281, 1998.

[23] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *TREC*, 19 pages, 1994.

[24] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD*, pp. 71–79, 1995.

[25] M. Sanderson and J. Kohler. Analyzing geographic queries. In *SIGIR Workshop on Geographic Information Retrieval*, 2 pages, 2004.

[26] B. Schnitzer and S. Leutenegger. Master-client R-trees: a new parallel R-tree architecture. In *SSDBM*, pp. 68–77, 1999.

[27] T. Strohman, H. Turtle, and W. B. Croft. Optimization strategies for complex queries. In *SIGIR*, pp. 219–225, 2005.

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

[29] D. A. White and R. Jain. Similarity indexing with the SS-tree. In *ICDE*, pp. 516–523, 1996.

[30] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM TOIS*, 22(2):179–214, 2004.

[31] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *ICDE*, pp. 688–699, 2009.

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

[33] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2), 56 pages, 2006.

[34] J. Zobel, A. Moffat, and K. Ramamohanarao. Inverted files versus signature files for text indexing. *ACM TODS*, 23(4):453–490, 1998.