Hybrid Index Structures for Location-based Web Search*

Yinghua Zhou[1], Xing Xie[2], Chuang Wang[3], Yuchang Gong[1], Wei-Ying Ma[2]

[1]Department of Computer Science, University of Sci. & Tech. of China, Hefei, Anhui, 230026, P.R. China
yhzhou@mail.ustc.edu.cn
ycgong@ustc.edu.cn

[2]Microsoft Research Asia, 5F, Sigma Center, No. 49, Zhichun Road, Beijing, 100080, P.R. China
{xingx,wyma}@microsoft.com

[3]Department of Computer Science, Huazhong University of Sci. & Tech., Wuhan, 430074, P.R. China
chwang@mail.hust.edu.cn

ABSTRACT

There is more and more commercial and research interest in location-based web search, i.e. finding web content whose topic is related to a particular place or region. In this type of search, location information should be indexed as well as text information. However, the index of conventional text search engine is set-oriented, while location information is two-dimensional and in Euclidean space. This brings new research problems on how to efficiently represent the location attributes of web pages and how to combine two types of indexes. In this paper, we propose to use a hybrid index structure, which integrates inverted files and R*-trees, to handle both textual and location aware queries. Three different combining schemes are studied: (1) inverted file and R*-tree double index, (2) first inverted file then R*-tree, (3) first R*-tree then inverted file. To validate the performance of proposed index structures, we design and implement a complete location-based web search engine which mainly consists of four parts: (1) an extractor which detects geographical scopes of web pages and represents geographical scopes as multiple MBRs based on geographical coordinates; (2) an indexer which builds hybrid index structures to integrate text and location information; (3) a ranker which ranks results by geographical relevance as well as non-geographical relevance; (4) an interface which is friendly for users to input location-based search queries and to obtain geographical and textual relevant results. Experiments on large real-world web dataset show that both the second and the third structures are superior in query time and the second is slightly better than the third. Additionally, indexes based on R*-trees are proven to be more efficient than indexes based on grid structures.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**] Content Analysis and Indexing *–Indexing methods* H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval *–Search process, Retrieval models.*

General Terms: Management, Design, Experimentation

Keywords: Location-based web search, spatial index, textual index, geographical ranking, geographical scope

* This work was done when the first and the third authors worked as interns at Microsoft Research Asia

CIKM'05, October 31–November 5, 2005, Bremen, Germany.

1. INTRODUCTION

Location-specific information is common on the Web. According to previous studies, nearly one fifth of web search tasks are related to a specific place or region [6,14], which is usually called location-based web search. Recently, more and more commercial search engines start to provide location based services, such as local search, local advertisements and map services. These services are particularly useful for mobile users.

Most commercial search engines, such as Google Local [5] and Yahoo! Local [17], only search business addresses in Yellow Pages or other kinds of paid lists. In this paper, we are interested in a more general form of local search, that is, to search local content on the Web. In our approach, each web page will be first assigned to a few geographical locations according to its content and then spatially indexed in the search engine. Therefore, it can be later retrieved by its locations.

How to efficiently index and search location-specific information is being a key problem for location based search engines. A straightforward approach is to treat geographical words which represent location information as common keywords, and to retrieve web pages with specified location names in the same way to keyword matching. However, simple keyword matching neglects underlying spatial relationships, therefore, does not support advanced spatial queries. To solve the problem, it is necessary to design an efficient index structure that considers both spatial and textual features of web pages.

In this paper, we studied and compared the performance of different hybrid index structures for location-based web search. In general, there are two main design issues: location representation and index combination scheme.

There are many types of location information on the Web. Our concern is the geographical scope of a web page, i.e. the geographical area that a creator of the page intends to reach [4], also can be intuitively explained as people think the page most relevant to. Recently, quite a few researchers have studied this problem [1,3,4,13,18]. Most of them use place names to represent the geographical scope which can be obtained by analyzing web textual content and/or geographical distribution of hyperlinks. In order to support spatial semantics, the scope should be, however, treated as a two-dimensional spatial object. After considering the trade-off between accuracy and computational cost, in this paper we use minimum bounding rectangle (MBR) based on longitude/latitude coordinates to represent a spatial object (in some applications, points may be sufficient for location representation. Our method can be easily adapted to support these applications.). The scope of a web page may include multiple spatial objects. Therefore, our scope

could be represented as multiple MBRs. In order to efficiently organize the two-dimensional data, we choose R*-tree [2], a popular and efficient spatial index for rectangles and points, to manage the scopes.

Next we consider the scheme for combining index structures of text and location information. We propose to use a hybrid index structure, which integrates inverted files and R*-trees, to handle both textual and location aware queries. Three different combining schemes are studied: (1) inverted file and R*-tree double index, (2) first inverted file then R*-tree, (3) first R*-tree then inverted file. The first structure includes two independent index structures, inverted list and R*-tree, to index web pages both textually and spatially. The second structure is designed based on the idea of spatially partitioning of each page list in the inverted files. The main idea of the third structure is, for each spatial object, we create inverted files for the web pages whose scope contains the spatial region. Experiments on large real-world web dataset show that these three structures have almost the same storage cost and both the second and the third structures are superior in query time and the second is slightly better than the third. Additionally, indexes based on R*-trees are proven to be more efficient than indexes based on grid structures.

Additionally, we design and implement a complete location-based web search engine to validate the performance of proposed index structures. The engine mainly consists of four parts: (1) an extractor which detects geographical scopes of web pages and represents geographical scopes as multiple MBRs based on geographical coordinates; (2) an indexer which builds hybrid index structures to integrate text and location information; (3) a ranker which ranks results by geographical relevance as well as non-geographical relevance; (4) an interface which is friendly for users to input location-based search queries and to obtain geographical and textual relevant results.

Our novel contributions include:

- We represented geographical scopes of web pages as multiple MBRs and discussed the corresponding index/search process;
- We studied and compared three hybrid index structures based on inverted files and R*-trees;
- We developed and introduced a complete location-based web search engine prototype based on proposed hybrid index structures;
- We carried out large-scale experiments based on real dataset to validate the performance of our index structures.

This paper is organized as follows. Section 2 describes related work. Section 3 is the introduction of the framework of our location-based web search engine and its main components. Section 4 is the analysis of the performance of hybrid index structures. Section 5 provides our experimental results, mainly on the index structures. Finally, we conclude the paper and discuss our future work in Section 6.

2. RELATED WORK

Location-based search has attracted a lot of attention in the research community. We will discuss the state of art from three aspects in the following subsections.

2.1 Location Representation

Generally, location information can be represented as either textual keywords (set space) or two-dimensional spatial objects (Euclidean space).

Textual keywords include postal codes, telephone numbers, place names, etc.[3,13]. Among them, place name is more convenient to express the location hierarchy and can be easily transformed to other representations. Place name is very useful for extracting and detecting the location information in web content. However, it cannot easily describe the detail shape of a place and spatial relationships among different places.

Two-dimensional spatial objects can be represented using vector model or raster model. Compared with textual keywords, they are more powerful in describing the region shapes. As to the raster model, the precision of the representation heavily depends on the size of grid cells. In [12], the authors superimposed a grid of 1024x1024 tiles on the total area of Germany. However, for an area like USA or the world which is relatively much larger than Germany, it is difficult to balance the storage requirement and the representation precision. In vector model, point locations are represented as points while region locations are described as polygons or minimum bounding rectangles (MBRs). Polygon representation is more accurate but the storage cost is large and the computation is complex. MBR is a simple approximation to a region's shape. Only two diagonal points are needed to represent the location information. Therefore, computation based on MBR is much simpler.

There exists much related work based on MBR in the literature, such as [8,9,11]. In this work, the scope of each page has only one MBR which is usually calculated as the bounding rectangle of all places mentioned in the page. While in our work, the scope of each page may have multiple MBRs, where each MBR corresponds to one focused region of the web page. As shown in Figure 1, if the scope of a web page includes South Dakota and Colorado, the representation of the scope is the solid rectangle for [9,11] while two dashed rectangles for our work. Apparently our approximation is more precise.

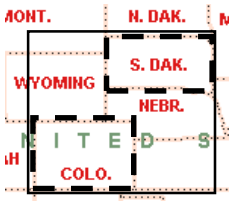

Figure 1. Geographical scope representation as MBRs.

2.2 Location Index

Different location representations lead to different index structures.

Place names can be organized as a flat list, or a hierarchy tree to represent the part-of relationships in the administrative hierarchy. The hierarchy structure is also very useful in detecting and extracting the main geographical scope of web pages, as shown in [4,18].

For representations based on spatial objects, using spatial index, for example R-tree family, quad-tree, or grid files, is a good choice. In [8], Global-Atlas stores the bounding rectangles of all regions in a database using Oracle Spatial Cartridge extension. In [9,11], their

work is based on an R-tree index of the MBRs of each page. The main difference between our work and their work is: their spatial index is based on the only MBR in each page and their datasets are about several thousands of web pages, while in our work the spatial index is based on multiple MBRs in a page and our work is based on over one million pages. Since the number of different MBRs (corresponding to different geographical scopes) is much smaller than the number of pages, the scale of spatial index will not become too large.

2.3 Index Combination Scheme

Existing work related to index combination schemes can be classified into two types.

The first type of approaches performs post-processing on general web search results. In [12], first textual web search is done in a way similar to conventional search and a set of pages are returned, then the location footprints of these pages are compared with the query location. Namely, conventional search is followed by a spatial filtering. In [9,11], spatial index is introduced to speed up the location filtering process. They build an R-tree dynamically on the search results from Google. However, the problem is that search engines only return the most relevant pages to users, so for those unpopular locations, the search results may contain very few correct results.

Table 1. Comparison of previous work and our work.

Work	Location representation	Location index	Combination scheme
[12]	Raster model	Quad-tree	Textual index without spatial index
[9,11]	MBR	R-tree	Textual search followed by online spatial indexing
[15]	Place name	Hierarchy gazetteer	Textual index with query expansion
[7]	MBR/polygon	Regular grid	Two-stage hybrid index
Our work	MBR	R*-tree	Two-stage hybrid index

The other type of approaches tries to integrate text and location information during indexing. In the Global Atlas search engine [8], all spatial and textual information query processes are carried out by the Oracle database. This approach can not deal with very large scale data like the Web. The most relevant work to us is a technical report [7] on their preliminary research of spatio-textual index in the SPIRIT project [15]. Their analysis is based on a hybrid structure of regular grids and inverted files. The conclusion is that the best spatial search time can be achieved when the cell size is 5% of the total area. In our option, regular grid structure is a coarse spatial partitioning method, while in our work R*-tree is used, which is more fine granularity. The number of pages related to a typical MBR is much smaller than that to a cell of 5% size; therefore the cost for merging lists is reduced. Experimental results on large-scale real dataset in Section 5 will show that, to obtain the same results, our hybrid index structures outperform the structure in [7] in query time.

Table 1 summarizes the characteristics of previous work and our work according to the above three aspects.

3. A LOCATION-BASED WEB SEARCH ENGINE

In this section, we will introduce the framework and main components of our location-based web search engine.

3.1 Framework

A complete work flow of our engine comprises offline processing and online processing. Offline processing includes extracting geographic scopes and indexing web pages according to their scopes, while online processing includes retrieving location aware information, ranking and presenting the retrieved results. Our search engine has four main components: extractor, indexer, ranker and search interface, as shown in Figure 2.

In the following subsections, we will introduce the above components in detail except the indexer which we will leave to Section 4.

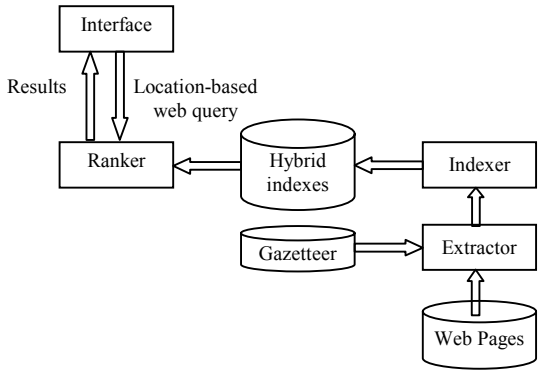

Figure 2. The system framework.

3.2 Extractor

The extractor module extracts geographical scope of pages and translates them to MBRs before sending them to the indexer.

The extraction and detection of scope is based on our previous work [18]. In that paper, web locations are divided into three types: provider location, content location and serving location. According to the type of location, web textual content and/or geographical distribution of hyperlinks and/or user logs are analyzed to extract the correct scope of web pages. Scope is represented as location names and a web page may have multiple location names as its scope.

A gazetteer is constructed to translate location names to MBRs based on geographical coordinates, which will be described in Section 5.1 in detail.

3.3 Search Interface

We provide users two types of search interfaces to input queries. One is based on maps and textual keywords, while the other is based on pure text input. For the former, a user can draw a region of interest on the map and input keywords in the text box. For the latter, a user can describe location names and spatial relationships textually. The latter type of interface will automatically detect the location information from search queries. Some related work on detecting location information in queries can be found in [19].

3.4 Ranker

The goal of the ranker is to return those important pages which are not only most relevant to text keywords but also most relevant to

query regions. Currently there is little work on how to combine geographical ranking and non-geographical ranking. Here we describe our preliminary study on the combination of two rankings and the computation of geographical relevance.

To combine two ranking values, a simple method is that the total ranking value of a web page is calculated as a weighted sum of the geographical and non-geographical ranking values.

In some cases, users may only care about whether pages' geographical ranking values are bigger than a threshold but do not care about their absolute values. We can first sort pages based on their geographical ranking values, then fetch those page whose value is bigger and sort them by their non-geographical ranking values.

Geographical ranking values lie heavily on the spatial query type. Our engine supports four spatial query types: contain, overlap, inside, and nearby. Their corresponding geographical ranking algorithms are described in the following subsections.

3.4.1 Contain

This type of queries tries to find pages whose scopes are contained by the spatial query region. We defined the geographical rank as:

$$grank(Q, R) = R / Q$$

Q is the extent of the spatial query region; R is the extent of the scope of a web page. A page is more geographical relevant if it has a larger scope.

3.4.2 Inside

This type of queries tries to find pages whose scopes contain the spatial query region. In this case, a page is less relevant if its scope is larger.

$$grank(Q, R) = Q / R$$

3.4.3 Overlap

This type of queries tries to find pages whose scopes overlap the query region. A page is more relevant if the overlap region is bigger.

$$grank(Q, R) = (Q \cap R) / (Q + R - Q \cap R)$$

$Q \cap R$ is the overlap extent of Q and R.

Sometimes a user only submit a query for local information while does not state his query types. In this case, we will execute three queries of type contain, inside and overlap, and the geographical ranking value is decided by the following equation:

$$grank(Q, R) = (Q \cap R) / (Q + R - Q \cap R)$$

3.4.4 Nearby

This type of queries tries to find pages whose scope is close to the query region. We transform this query type to an overlap query. The query region is a circle whose center is the query point or the center of the query region. Users can specify the radius of the circle, i.e. the distance between the results and the query region, or specify the number of results for dynamically adjusting the radius. The relevance of results will lie on the distance between the query region and the geographical scopes, nearer means more relevant.

4. INDEXER

The indexer aims to build hybrid index structures to integrate text and location information of web pages. To textually index web pages, inverted files are a good choice as shown in conventional search engines. To spatially index web pages, two-dimensional spatial indexes are used, for example, R-tree family, quad-tree and grid structure. R-tree uses the minimum bounding rectangle (MBR) as an approximation to a spatial object, which is similar to our approximation of the geographical scope. R*-tree is a variant of R-tree that can further improve search performance, so we choose R*-tree as the spatial index. Additionally, considering that the index is built while offline processing and the collection of geographical scopes is stable with time, we use Sort-Tile-Recursive (STR) algorithm [10] as the packing algorithm to pre-process the spatial datasets before building R*-trees.

We study three hybrid methods: (1) inverted file and R*-tree double index, (2) first inverted file then R*-tree, (3) first R*-tree then inverted file. We will describe the hybrid index structures and present cost models for each structure. The symbols used in the cost models are listed in Table 2.

Table 2. The description of symbols.

Symbol	Description
M	The number of MBRs in the gazetteer
G	The number of geo-keywords in the datasets
K	The number of keywords in the lexicon
$g(Q)$	The number of geo-keywords for a query Q
$P_K(k)$	*The length of the page list of a keyword k
$P_M(m)$	*The length of the page list of an MBR m
$P_G(g)$	*The length of the page list of a geo-keyword g
B_{List}	Storage of page lists
$B_R(x)$	Storage of an R*-tree of x elements
$T_{I/O}$	The time cost of disk accesses
T_{disk}	The time cost of one disk access
$T_R(x)$	The time cost to retrieve an R*-tree of x elements
$T_{mg}(x)$	The time cost to merge x elements

4.1 Inverted File and R*-tree Double Index

In this structure, web pages are indexed separately twice, once by R*-tree and once by inverted files. All MBRs are indexed by an R*-tree. The difference from conventional R*-tree is that each leaf node of the MBR tree points to a page list whose scope includes this MBR, as shown in Figure 3. Inverted files are the same to conventional search engines. Thus we have two kinds of page lists whose entry is either an MBR or a keyword.

A location-based web search comprises non-spatial keywords and query region and/or specified spatial query types. Non-spatial query keywords are retrieved similar to conventional inverted files, while query region and spatial query type are passed to the R*-tree. The final results are the merge of page lists from two indexes.

The storage in disk comprises the two kinds of page lists and the R*-tree. Therefore,

$$Storage_1 = B_R^1 + B_{List}^1$$

The storage of an R*-tree that has x leaf nodes [16] is $B_R(x) = O(x)$

The storage of page lists depends on the length of each list, whose unit is the identifier of a page Assuming the length of the list whose entry is keyword k is $P_K(k)$ and the length of the list whose entry is MBR m is $P_M(m)$, the total length of all lists is $\sum_{m=1}^{M} P_M(m) + \sum_{k=1}^{K} P_K(k)$.

Then

$$B^1_{List} = O(\sum_{m=1}^{M} P_M(m) + \sum_{k=1}^{K} P_K(k))$$

Now, $Storage_1 = B^1_R + B^1_{List}$

$$= O(M) + O(\sum_{m=1}^{M} P_M(m) + \sum_{k=1}^{K} P_K(k))$$

$$= O(\sum_{m=1}^{M} P_M(m) + \sum_{k=1}^{K} P_K(k))$$

So, the main cost of storage in disk is caused by two kinds of page lists above. For the storage of the identifier of pages is about a fixed value, the storage is mainly determined by the total length of all page lists.

Assume there is a query including m keywords and a query region. The online computation has three parts: (1) the retrieval of the m keywords in inverted files and the loading of corresponding page lists from the disk; (2) the retrieval of the R*-tree, assuming n MBRs are got, and the loading of corresponding page lists of these MBRs from the disk; (3) the merge of these $(m+n)$ page lists.

The retrieval of keywords is implemented by a hashing function, so we think the time can be ignored. The time of loading page lists is determined mainly by the number and total length of lists. The merge processing depends on the total length of these page lists.

For the R*-tree that in this structure has M leaf nodes, assuming that the query time is $T_R(M)$

The merge time for x elements in memory is: $T_{mg} = O(x)$

The time to read a page list whose length is x from disk is

$$T_{I/O} = T_{disk} \cdot O(x / B_{section})$$

In the above equation, $B_{section}$ is the size of a section of the disk which depends on the file system of a computer. In our system it is 4Kbytes.

Then, $Time_1 = T^1_R + T^1_{I/O} + T^1_{mg}$

$$= T_R(M) + (\sum_{i=1}^{m} T_{disk} \cdot O(P_M(m_i) / B_{section})$$

$$+ \sum_{i=1}^{n} T_{disk} \cdot O(P_K(k_i) / B_{section})) + O(\sum_{i=1}^{m} P_M(m_i) + \sum_{i=1}^{n} P_K(k_i))$$

For different queries, there are two factors that effect the query time. One is the merge operations of m page lists whose entry is a keyword and n page lists whose entry is an MBR, which depends on the total length of these $(m+n)$ page lists. The other factor is the time to read these $(m+n)$ page lists from disk.

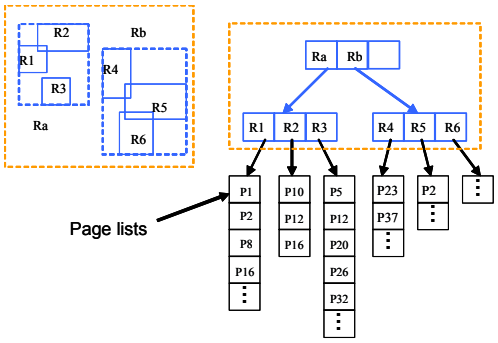

Figure 3. The structure of R*-tree in the hybrid structure of inverted file and R*-tree double index.

4.2 First Inverted File Then R*-tree

As shown in Figure 4, each keyword points to an R*-tree. For each page list whose entry is a keyword in the first hybrid structure, these pages in the list are assigned to different MBRs according to their geographical scopes, and an R*-tree is built on these MBRs, then we can get a set of page lists whose entry is determined by a pair of a keyword and an MBR. A pair of a keyword and an MBR is named a **geo-keyword** if there is a page which includes the keyword and whose scope includes the MBR.

Assuming $P_G(g)$ is the length of a page list whose entry is a geo-keyword g.

The storage in disk includes these page lists and R*-trees pointed by K keywords. So

$$Storage_2 = B^2_R + B^2_{List}$$

$$= K \cdot O(M) + O(\sum_{g=1}^{G} P_G(g))$$

In fact, an R*-tree in this structure may not index all M MBRs as in the first structure, so the scale of R*-trees is smaller. Thus we can see that the cost of storage in disk is mainly caused by the total length of page lists whose entry is a geo-keyword.

Assuming the number of geo-keywords for a query Q of m keywords and n MBRs is $g(Q)$. The online computation includes: (1) first to retrieve the m query keywords; (2) to search in the corresponding R*-trees whose number is m and the average leaf node is $\overline{M}$, and to find some MBRs and their corresponding page lists, the number of lists got from m R*-trees is $g(Q)$; (3) to merge these $g(Q)$ page lists. The retrieval for m keywords is implemented by a hashing function, and the time is ignored. So,

$$Time_2 = T^2_R + T^2_{I/O} + T^2_{mg}$$

$$= m \cdot T_R(\overline{M}) + \sum_{i=1}^{g(Q)} T_{disk} \cdot O(P_G(i) / B_{section}) + O(\sum_{i=1}^{g(Q)} P_G(i))$$

Besides the retrieval of m R*-trees, there are also two main factors for online search. One factor is caused by the total length of the page lists whose entry is a geo-keyword, the number of lists is $g(Q)$. The other factor is the time to read these $g(Q)$ page lists from disk. The pages in a page list whose entry is a geo-keyword is a subset of pages in the page list whose entry is the corresponding keyword or MBR, so the length of a page list whose entry is a geo-keyword is greatly reduced.

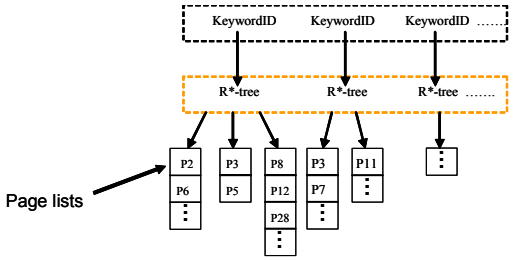

Figure 4. The illustration of first inverted file then R*-tree index structure.

4.3 First R*-tree Then Inverted File

As shown in Figure 5, an R*-tree is built on all MBRs included in scopes of all web pages. And web pages are assigned to MBRs according to their scopes. After all pages of each MBR are textually indexed by keywords, we can get a set of page lists whose entry is a geo-keyword.

The main storage in disk includes the page lists whose entry is a geo-keyword and the R*-tree.

$$storage_3 = B_R^3 + B_{List}^3$$
$$= O(M) + O(\sum_{g=1}^{G} P_G(g)) = O(\sum_{g=1}^{G} P_G(g))$$

The main cost of storage in disk is caused by the total length of page lists whose entry is a geo-keyword.

The online computation includes: (1) first to search the R*-tree and get n MBRs, (2) for each MBR, to retrieve which of the m keywords are pointed to, then to load corresponding page lists. The number of lists in total is $g(Q)$; (3) to merge these page lists. So, $Time_3 = T_R^3 + T_{I/O}^3 + T_{mg}^3$

$$= T_R(M) + \sum_{i=1}^{g(Q)} T_{disk} \cdot O(P_G(i)/B_{section}) + O(\sum_{i=1}^{g(Q)} P_G(i))$$

Similar to the second structure, besides the retrieval of the R*-tree, there are also two main factors, the total length of these $g(Q)$ page lists and the time to read these $g(Q)$ page lists from disk.

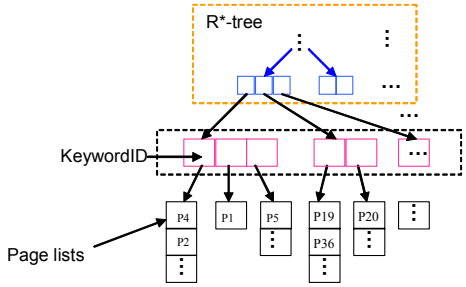

Figure 5. The illustration of first R*-tree then inverted file index structure.

In summary, the online performance of the first structure depends on the total length of page lists of m keywords and n MBR, also on the time to read these $(m+n)$ page lists from disk. Besides the retrieval of R*-trees, the second and third structure mainly depends on the total length of $g(Q)$ page lists, also on the time to read these $g(Q)$ page lists from disk, so the two have similar performance. And in real case $g(Q)$ is relatively smaller, which can be found on real large-scale dataset in Section 5, so the performance of the second and third is better than that of the first. Additionally, there are m times of R*-tree searching for the second while one time of R*-tree searching for the third, but the R*-tree in the second has $\overline{M} = G/K$ leaf nodes on average while the R*-tree in the third has M leaf nodes. In fact, for each keyword, the number of MBRs to make a geo-keyword with this keyword is limited, i.e. $\overline{M}$ is much smaller than M, the experiments in Section 5 will prove it. Thus the scale of R*-trees in second is much smaller than of the R*-tree in third. Based on the analysis of [2], for the same query, the scale of an R*-tree is most important factor for query time; in the worst case all leaf nodes have to be accessed. So for a query, the query time in any of R*-trees of the second is much smaller than in the R*-tree of the third. Considering that m is very small too, the second may be slightly better than the third in query time.

5. EXPERIMENTS

To evaluate the performance of three hybrid index structures, we implemented them in our system and compared them with existing grid based indexes. In the following we will first describe the experimental settings and dataset, then discuss the results of the experiments.

5.1 Settings and Dataset

We use .GOV data as our benchmark dataset. It is a collection of real web resources of major USA government sites whose top domain is .gov. These data are mainly crawled in the year 2002 and used by TREC2003. The dataset covers a wide geographical range of USA.

To spatially index web pages, we should first get geographical scope of web pages. In the work of [18], geographical scope has been extracted as place names, so a gazetteer must be constructed in advance to translate place names to MBRs. In the absence of MBR representation for each place, we get the standard longitude/latitude coordinates of a place through Microsoft MapPoint Service. However, the borders of such an MBR are not straight lines. To solve the problem, we use Gauss-Kruger reference frame, which has little deformation in angle, length and extent, to transform the coordinates. Thus, our gazetteer can map location names to MBRs based on Gauss-Kruger coordinates. For textual index, the inverted files are created based on the work of MSRA's Web search platform for TREC2004.

Table 3. Statistics of our dataset.

Statistics	Value
The number of all pages	1,053,111
The number of local pages	197,775
The occurrences of MBRs in local pages	197,988
The number of MBRs	26,090
The number of all keywords L	2,684,633
The number of geo-keywords G	3,535,505
The number of MBRs included in geo-keywords M	4,246
The number of keywords included in geo-keywords K	758,717

Our experimental environment is a machine with an Intel Xeon 3.06 GHz CPU, 2 GB RAM, and running Microsoft Windows Server 2003.

After analyzing the dataset, we found that about 18.78% pages are local pages, i.e. have at least one geographical scope. Our experiments are mainly carried out on these local pages to emphasize the indexing performance for location-based web query. The total number of keywords K is 2,684,633; the number of MBRs in the gazetteer is 26,090; and the total number of occurrences of MBRs in all pages is 197,988; the number of geo-keywords is 3,535,505;the number of MBRs included in geo-keywords is 4,246; the number of keywords included in geo-keywords is 758,717, as shown in Table 3.

5.2 Comparison of Three Hybrid Index Structures

We compared the disk storage requirement and the query time of hybrid index structures.

As we can see from Table 4, the storage of the first structure is 140.00+0.83=140.83 Mbytes, approximately equaling to that of the other two which is about 138.95 Mbytes.

Additionally, the number of lists in the first structure is much smaller than that of the other two (758,717 + 4,246 << 3,535,505), while the total length of page lists in the first structure is more than the other two (One unit of the length of a page list is the identifier of a page). So the average length of each list in the first structure is much longer.

There are K smaller R*-trees in the second structure and only one bigger R*-tree in the third structure. However, the storage space for R*-trees is relatively small compared with the size of page lists. So the difference between the second and the third structures is very small.

Table 4. Disk storage for three hybrid index structures.

	Page lists	The number of lists	Total length of page lists	Average length	Physical size (Mbytes)
The 1st structure	entry is a keyword	758,717	33,481,669	44.13	140.00
	entry is an MBR	4,246	197,988	46.63	0.83
The 2nd and 3rd structures	entry is a geo-keyword	3,535,505	27,666,384	7.83	138.95

To test the query time of the three structures, we used a query set comprising 2000 queries which were randomly generated. 1000 queries were input by drawing a query region on the map, and another 1000 queries were input by adding location keywords. Four spatial query types were randomly assigned to each query. In the test set, there are 551 contain queries, 517 overlap queries, 514 inside queries and 418 nearby queries. All these queries were submitted to three hybrid index structures.

As discussed in Section 4, the query time has three main parts: the time for retrieving relevant MBRs from R*-trees, the time for disk access and the time for merging page lists: $Time = T_R + T_{I/O} + T_{mg}$.

The results in Table 5 indicate the second and the third structures have obvious advantages over the first. This is because that the first spends too much time on $T_{I/O}$ and T_{mg}. These two factors are both determined by the length and number of page lists. We can see from Table 4, the average length of page lists in the first structure is much longer than that of the other two. The situation is the same for the number of lists to read. It is because that more lists are required to find the correct results for the first structure. Additionally, there are m R*-trees to search in the second while one R*-tree to search in the third; but on average each R*-tree in the second has 3,535,505/758,717=4.6 leaf nodes, while the R*-tree in the third has M=4,246 leaf nodes. Considering that there are 2.4 keywords per query on average i.e. m=2.4 in our query sets, the second structure spends less time in searching in R*-trees than the third. This verifies our analysis in Section 4.

Table 5. Average query time for three hybrid index structures.

	The 1st structure	The 2nd structure	The 3rd structure
Total length of page lists per query	38,868.91	122.42	122.42
Number of page lists per query	72.04	4.36	4.36
T_R*(ms)*	2.34	0.16	2.34
$T_{I/O}$*(ms)*	30.83	7.91	7.91
T_{mg}*(ms)*	17.01	0.73	0.73
Query time*(ms)*	50.18	8.80	10.98

5.3 Comparison of Hybrid Index Structures Based on R*-tree and Grid Structure

In this subsection, we compared the query time of our second and third structures with grid based structures [7]. In [7], their hybrid index structures are implemented based on regular grid structures and inverted files. The comparison was analyzed on the same query set as the previous subsection.

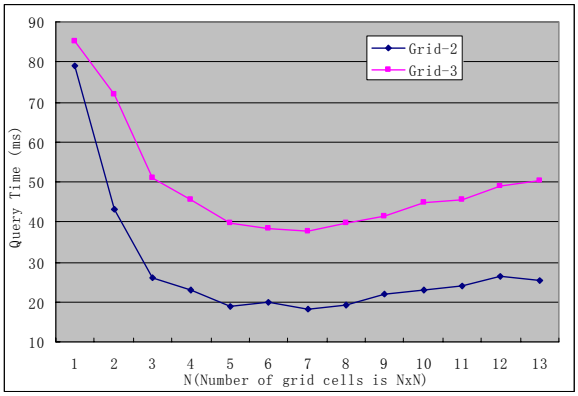

Figure 6. Average query time for hybrid index structures based on grids of different sizes.

In Figure 6, Grid-2 and Grid-3 stand for the second and the third structures based on grid. The x-axis shows the number of grid cells. The results show that grid based structures achieved the best performance when the number of cells is 7x7=49.

The results in Table 6 indicate that our structures based on R*-tree are superior to a regular grid with 7x7 cells which are shown to be the best parameters. Since the regular grid is a coarse granularity spatial division, additional comparisons should be done to judge whether the results from searching in the grid structure really match the query regions. This can be seen from Table 6, the spatial searching time in our structures is much less than that in the grid based structures.

Table 6. Average query time for hybrid index structures based on R*-trees and a 7x7 grid structure.

	The 2nd structure		The 3rd structure	
	Grid	R*-tree	Grid	R*-tree
Spatial search time*(ms)*	9.63	0.16	28.09	2.34
$T_{I/O}$*(ms)*	7.91	7.91	7.91	7.91
T_{mg}*(ms)*	0.73	0.73	0.73	0.73
Query time*(ms)*	18.27	8.80	37.73	10.98

6. CONCLUSIONS

In this paper, we have studied the performance of hybrid index structures that integrate text indexes and spatial indexes for location based web search. In our approach, we represented the geographical scopes of web pages as multiple MBRs and compared three hybrid index structures based on inverted files and R*-trees. We have also developed a complete location based search engine and carried our large scale experiments to validate the proposed structures. Experiments showed the structure of first inverted file then R*-tree is the most efficient in query time.

In our future work, we will continue to improve the performance for location indexing. Geographical ranking is also an important problem to study, which is critical for improving the performance of location based web search.

7. REFERENCES

[1] Amitay, E., Har'El, N., Sivan, R., and Soffer, A. Web-a-where: geotagging web content. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (ACM 2004)*, ACM Press, Sheffield, UK, 2004, 273-280

[2] Beckmann, N., Kriegel, H., Schneider, R. and Seeger B. The R*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data (SIGMOD 1990)*, Atlantic City, NJ, USA, 1990, 322-331

[3] Buyukkokten, O., Cho, J., Garcia-Molina, H., and Shivakumar, N. Exploiting geographical location information of web pages. In *ACM SIGMOD Workshop on The Web and Databases (WebDB 1999)*, Philadelphia, Pennsylvania, USA, 1999, 91-96

[4] Ding, J., Gravano L., and Shivakumar N. Computing geographical scopes of web resources, In *Proceedings of 26th International Conference on Very Large Data Bases (VLDB 2000)*, Cairo, Egypt., 2000, 545-556

[5] Google Local http://local.google.com

[6] Gravano, L., Hatzivassiloglou, V., and Lichtenstein, R. Categorizing web queries according to geographical locality. In *Proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management (CIKM 2003)*, ACM Press, New Orleans, Louisiana, USA, 2003, 325-333

[7] Jones., C.B. and Vaid., S. *Report on spatial indexing methods*. Technical report D12 2201, SPIRIT Project, 2004

[8] Lee, F., Bressan, S., and Ooi, B.C. Global atlas: calibrating and indexing documents from the internet in the cartographical paradigm. In *Proceedings of the 1st International Conference on Web Information Systems Engineering(WISE 2000)*, IEEE Computer Society 2000, Hong Kong, China, 2000, 125-132

[9] Lee, R.,et al. Optimization of geographic area to a web page for two-dimensional range query processing. In *Proceedings of Fourth International Conference on Web Information Systems Engineering Workshops(WISEW 2003)*, IEEE Computer Society 2003, Roma, Italy, 2003,9-17

[10] Leutenegger, S.T., Edgington, J.M., and Lopez, M.A. STR: a simple and efficient algorithm for R-tree packing. In *Proceedings of the Thirteenth International Conference on Data Engineering (ICDE 1997)*, IEEE Computer Society 1997, Birmingham, U.K., 1997, 497-506

[11] Ma, Q. and Tanaka, K. Retrieving regional information from web by contents localness and user location. In *Asia Information Retrieval Symposium (AIRS 2004)*, Lecture Notes in Computer Science, Beijing, China, 2004, 301-312

[12] Markowetz, A., Chen, Y., Suel, T., Long, X. and Seeger, B. *Design and implementation of a geographic search engine*. Technical Report TR-CIS-2005-03, Polytechnic University, Brooklyn, New York, 2005. http://cis.poly.edu/tr/tr-cis-2005-03.shtml

[13] McCurley, K.S. Geospatial mapping and navigation of the web, In *Proceedings of the Tenth International World Wide Web Conference (WWW 10)*, ACM Press, Hong Kong, China, 2001, 221-229

[14] Sanderson, M. and Kohler, J. Analyzing geographic queries, In *Proceedings of SIGIR 2004 Workshop on Geographic Information Retrieval*, ACM Press, Sheffield, UK, 2004

[15] SPIRIT project. http://www.geo-spirit.org

[16] Theodoridis, Y., Stefanakis, E. and Sellis, T. Efficient cost models for spatial queries using R-Trees. *IEEE Transactions on Knowledge and Data Engineering*, Volume 12, Issue 1, 2000, 19-32

[17] Yahoo Local http://local.yahoo.com

[18] Wang, C., Xie, X., Wang, L., Lu, Y., and Ma, W.Y. Web resource geographic location classification and detection. In *Proceedings of the 14th International World Wide Web Conference(WWW2005)*, Chiba, Japan, 2005, poster, 1138-1139

[19] Wang, L., Wang, C., Xie, X., Forman, J., Lu, Y., Ma, W.Y. and Li, Y. Detecting dominant locations from search queries, In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2005)*, ACM Press, Salvador, Brazil, 2005