Wavelet Tree based Dual Indexing Technique for Geographical Search

Arun Kumar Yadav[1] and Divakar Yadav[2]
[1]Department of Computer Science and Engineering, Ajay Kumar Garg Engineering College, India
[2]Department of Computer Science and Engineering, Madan Mohan Malaviya University of Technology, India

Abstract: *Today's information retrieval systems are facing new challenges in indexing the massive geographical information available on internet. Though in past, solutions for it, based on R-tree family and B-tree has been given, but due to increased size of index, they are found to be less efficient and time consuming. This paper presents a dual indexing technique for Geographical Information Retrieval. It uses wavelet tree data structure for both, textual and spatial indexing. It also allows dynamic insertion of Minimum Bounding Rectangle (MBR) in the wavelet tree during index construction. The proposed technique has been evaluated in terms of efficiency and time complexity. For pure spatial indexing, using this technique, the search time complexity is reduced and takes even less than one third time of that of spatial indexing performed using R-tree or R*-tree. Even in case of dual indexing (textual and spatial) using wavelet tree, the search time is reduced by half in comparison to other techniques such as B/R, B/R* when the search query length is larger than 2 keywords. In case the query is of 1 or 2 keywords, the search time remains approximately similar to that of other techniques.*

Keywords: *Information retrieval, wavelet tree, spatial search, indexing, Minimum bounding rectangles.*

Received May 28, 2016; accepted May 1, 2017

1. Introduction

Information Retrieval (IR) is the activity of obtaining information resources relevant to an information need from a collection of information resources. Searches can be based on full-text or other content-based indexing. Now-a-days improvements in the standard of computing resources have made the implementation of ontology based information retrieval systems very affordable. Such systems have plenty of data to be processed, which requires an efficient indexing technique. Over the past decades, several different indexing structures have been developed. Many of these indexing structures were designed for secondary memory due to small size and high cost of main memory. But, recently the drastic change in the price of main memory has made it possible to have indexes in main memory itself. This sets the requirement of developing a primary memory based indexing structure. This paper presents a compact structure known as wavelet tree for performing indexing, that facilitates to maintain a proper balance between search efficiency and the storage requirement.

The term 'spatial database system' has become popular during the last few decades. In 2006, Andrade and Silva [1] had presented an indexing and ranking architecture for a new geographical web retrieval system based on distances between users and the resources. Jones *et al*. [10] in 2004 had given a search engine named Spatially-Aware Information Retrieval on the Internet (SPIRIT) that was specialized for access to geographical information. In the very next year, 2005 Zhou *et al*. [18] had proposed a hybrid index structure which integrated inverted files and R^*-trees to handle both textual and location based queries. He concluded that the structure in which first inverted file is used and then R^* tree, is more efficient in terms of query time. In the very next year 2006 Brisaboa *et al*. [3] focused on a sub-category of spatial indexes, based on wavelet tree. It solved the problem of indexing points while maintaining a proper balance between the space and search efficiency. In the same year, he proposed a self-indexing method based on wavelet tree. It basically joined Huffman compression technique with wavelet trees. He presented three variants which together aimed at reducing the size while preserving the space requirement. In 2007, Hariharan *et al*. [9] presented a framework for geographical information retrieval system and proposed an approach that can process spatial keyword queries efficiently. Lieberman *et al*. [11] in 2007 had given a model named Spatio-Textual Extraction on the Web Aiding Retrieval of Documents (STEWARD) i.e., spatio-textual extraction on the web aiding retrieval of documents. It is a system for extraction, performing query and visualizing textual references. Lin *et al*. [12] in 2007 proposed a spreading activation network based model for geographical information retrieval. With the evolution in the hierarchy of memory, it has now become possible to have compact structures for representing the indexes. With this advancement, Brisaboa *et al*. [4] in 2010 proposed a wavelet tree based structure for representing the geographical data. They efficiently represented the Minimum Bounding Rectangles (MBR) for solving the spatial queries with the help of wavelet tree structure.

Thorough literature review of the existing techniques reveals that none of the above discussed schemes have used wavelet tree based dual indexing for spatial and textual search in case of geographical information retrieval [4]. In most of the cases researchers have used B-tree for textual index and R-tree family for spatial search. Common drawbacks of the existing schemes are that more space is required to store indexes. The time complexity is higher and overlapping exists in case of dynamic insertion of MBRs in R-tree family. In addition to this one more drawbacks in the existing methods is that they are not able to assign multiple geographical references for a web document. Scheme proposed in this paper overcomes most of the above said drawbacks of the existing methods.

This work extends the work proposed in [7] for spatial as well as textual search and proposed a dual indexing technique for Geographical Information Retrieval based on the same structure i.e., Wavelet Tree. The benefit of using dual indexing is that it supports parallel computing in case of large data including textual and spatial format. The proposed technique can be extended concurrently on different machine to search textual data on one machine and spatial data on another machine. This helps in reducing the search time as well the space required for storage. We have also proposed an algorithm for the dynamic insertion of MBRs in the wavelet tree during index construction which also reduces MBR overlapping problem.

Our novel contributions in this paper are as follows:

- Use of same data structures (wavelet tree) instead of different structures for spatial and textual search which reduces the search time and storage space.
- Minimized the problem of MBR overlapping while inserting it dynamically in the wavelet tree.

Rest of the paper is organized as follows. Next section describes the related concepts. Section 3 presents our proposed work. Section 4 presents experimental results. Finally, we conclude in section 5.

2. Related Concepts

The growing use of internet has increased the research in information retrieval by many folds. Information retrieval can be done in two ways: textual and spatial. These researches shows that it has been widely used in the implementation of document databases, digital libraries and web search engines. Many of the documents stored in databases include geographic references in form of text. The main goal is the retrieval of geographically relevant documents in response to queries of the form <theme, location>.

Spatial indexes are key components in Geographical Information System (GIS). Spatial indexes can be used to solve both range queries and point queries of which range queries have proven to be better than Point Access Methods (PAMs) [16]. Gaede and Gjnther [8] in 1998 classified these index structures into two categories, one based on point query and other on polygon query. Point query methods are used to improve the access time in collections of spatial points. Polygon (rectangle) query methods are more general and are used to improve the access time in collections of geographical objects.

Geographical Information Retrieval (GIR) system consists of both, thematic as well as geographical information [9]. For the better performance of the complete system, indexing is required to be done which facilitates better retrieval. In this paper, we focus mainly on geographical information retrieval which can be indexed in the following four diverse ways:

2.1. Index Based only on Keywords

In this, indexing is only based on the keyword [15]. It is basically done for the textual data. It does not differentiate the spatial keywords present in the query, rather treats it similar to textual keywords. So for spatial keywords, a separate index of these words is maintained. While solving a query having both parts i.e., textual as well as spatial, first it processes the query on the basis of textual keywords of the query using the inverted file index and thus obtained documents are checked whether they meet the spatial part of the query as well or not. In this strategy, it was found that the major computational time is consumed to perform textual followed by spatial searching to find the final query results. It is also not efficient when multiple geo footprints are available for same documents

2.2. Keyword-Spatial Dual Index

In this, both keyword as well as spatial indexing is performed. For the purpose of analysis, the complete process is divided into three parts. First is processing of the textual part using inverted file index. Second is query on the R-tree for finding the MBR's, and finally merging the two results. This indexing is somewhat better as compared to pure keyword based index but, it introduces large storage overhead.

2.3. Spatial-Keyword Hybrid Index

In this, initially, the spatial indexing is done and then we get the pointer to the inverted files, which contain textual indexes [2]. The only difference in this method is that in the contents of the leaf nodes. In the previous one leaf node contained the pointers to the spatial data which required greater storage whereas in this the leaf node have the pointer to the smaller inverted files containing the textual indices.

2.4. Keyword-Spatial Hybrid Index

In this, firstly the inverted file is built using the keywords for indexing all the keywords contained in the query. Then the spatial index is built and placed in between vocabulary and occurrence. This is done only to index the occurrences of the keyword according to their geographic approximation. At last, these occurrence tables are divided into several smaller parts and put in the leaf nodes, which reduces the storage overhead.

Apart from these indexing techniques, there are several data structures that has been used in past for solving both range queries as well as point queries using spatial indexes. Some of them are discussed here.

2.5. R-Tree

It is a depth-balanced tree in which each node corresponds to a disk page [14] (i.e., the number of entries in each node is limited). Two types of queries can be solved using R-tree: point query: "what object contains the query point" and window query: "what objects intersect the query window". R-Trees can organize any-dimensional data by representing the data with a minimum bounding box. Each node bounds its children. A node can have many objects in it. The leaves point to the actual objects (stored on disk). The major disadvantages associated with R-trees are:

1. The execution of a point location query on R-tree may require the traversal of several paths from the root to the leaf. This contributes to poor performance [6].
2. Presence of large rectangles increases the degree of overlapping which contributes to performance degradation.

2.6. R$^+$ Tree

It avoids multiple paths during searching; instead objects may be stored in multiple nodes. So, redundant information appears in two or more nodes in case of overlaps but MBR overlapping of internal nodes can be avoided. On performing insertion/deletion operation, the tree may change downward or upward in order to maintain the structure. The fact that multiple copies of an object's MBR may be stored in several leaf nodes has a direct impact on the deletion algorithm. All copies of an object's MBR must be removed from the corresponding leaf nodes.

2.7. R* Tree

The original R-tree only minimizes the area of MBRs but R* tree is a significant structure to minimizes area, overlap and maximizes space utilization. A reinsertion algorithm is used for tree rebalancing and significantly improving performance during query processing. Re-inserting data improves performance by 20-50% as has been proved experimentally. However, reinsertion is a costly operation. Therefore, only one application of reinsertion is permitted for each level of the tree. When overflow cannot be handled by reinsertion, node splitting is performed. The major drawback with R* tree is that it has no control over space utilization and achieves approximately 70% maximum utilization.

2.8. Hilbert R Tree

Several space filling curves can be used to impose a linear ordering on the rectangle- z-order curves, Hilbert curves, and gray code curves. Hilbert R tree is a combination of R-tree and B+tree. It is a structure based on Hilbert space filling curves and aims to reduce the area and perimeter of resulting MBRs for spatial searches. The Hilbert value of a rectangle is actually the Hilbert value of its centre. In this tree, similar nodes are grouped together to achieve as high as 100% space utilization depending on how "good" the grouping is. It behaves like an R tree on spatial search operation whereas for insertion it uses deferred splitting technique which uses the Hilbert value of the inserted data as the primary key.

Generally, one use R- tree and its variants for spatial indexing which are based on MBRs but there are some major issues which may arise. They are:

- Use of different data structures is required for different types of indexing as there is no single data structure to identify both point and spatial accesses to memory. One keep using inverted file structure for textual indexing and R-tree family for spatial indexing and thus it makes implementation difficult.
- Another problem which can be attributed to the use of traditional R-trees is the space requirement. As the size of indexes grow, more and more secondary storage is required and the actual memory required becomes vast.
- As the number of geographic references in the users' query increases, the number of objects, to be accessed increases and so does the searching time.
- It is difficult to define spatial ordering for spatial objects.

2.9. Wavelet Tree

It is a compressed structure which facilitates the storage of indexed data in a compact way [5, 6, 13, 17]. A wavelet tree-based structure allows us to represent minimum bounding rectangles, solving geographic range queries in logarithmic time. It has efficient *rank* and *select* operations which can be used for document retrieval and traversal of the tree in constant time. For a given bitmap B of size n, rank (B, i) returns the frequency of 0/1 till position i and select (B, j) returns the position of *j*th bit set to 0/1 in B.

3. Proposed Work

This section describes the usage of wavelet trees to solve range queries using dual indexing which define a rectangular query window $[l_i^q, u_i^q]$ in response to a query given in the form <theme, location>. The location can be recognized practically with the help of gazetteer and rectangular query window is created to identify the overlapped documents. The goal is to retrieve all geographic objects that have at least one point in common with the query rectangle. This section is divided into three parts, creation of query rectangle Minimum Bounding Rectangle (MBR), insertion of the newly created MBR into the wavelet tree structure, and searching of the MBR. These three parts are discussed in detail along with example in the following sections.

3.1. Creation of Query Rectangle (MBR)

Through the literature, it is found that rectangular polygon is optimal in all the aspects of parameters including minimum data loss and implementation. Creation of rectangle is important part for spatial index generation and to define spatial query window of user.

For a given spatial search like hotels in Ghaziabad, a specific set of latitude and longitude related to location Ghaziabad is required. Query window is a collection of all these points on the 2D map as shown in Figure 1. The query window has its own MBR which is defined by the end points of the query window. There are two endpoints for any MBR, (l_1, u_1) and (l_2, u_2). These endpoints determine the range over which a region would be bounded. In similar way the query window end points are represented by (l_1^q, u_1^q) and (l_2^q, u_2^q). The values (l_1^q, u_1^q) and (l_2^q, u_2^q) are generated using following Algorithm:

Algorithm 1: MBR creation for GIR Document

- *Step 1: The left uppermost and right bottom most points for a MBR is found which is calculated on the basis of latitudes and longitudes (from gazetteer database) of locations described.*
- *Step 2: For the query window, the values are marked as follows:*

l_1^q = minimum value of latitude of the region surrounding.
u_1^q = minimum value of longitude of the region surrounding.
l_2^q = maximum value of latitude of the region surrounding.
u_2^q = maximum value of longitude of the region surrounding.

- *Step 3: A rectangle is made using the endpoints of the query window by intersecting the lines l_1^q with u_1^q and l_2^q with u_2^q.*

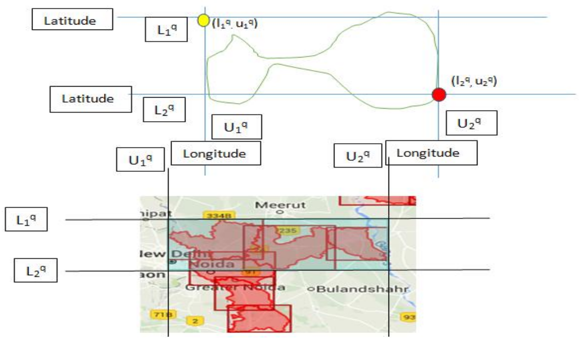

Figure 1. The MBR of different regions.

The rectangle so formed is the MBR for our query window and the same is used to determine query result as shown in Figure 2. Query window is formed based on collection of all the MBR's which satisfies the query. Here, we have query window defined in the region X [5, 6] and Y [7, 10].

The query window is scaled for convenience in calculation of overlapping regions.

For example, for the query "hotels in Ghaziabad", it is found in database that there are 3 locations related to Ghaziabad. All the locations have their specific location ids, latitude and longitude.

In Figure 3, Ghaziabad's location endpoint is based on these 3 locations and this defines the boundary of the query window as can be seen in Figure 3. The query window is shown in green colour and can be seen that it consists of a region formed by 3 locations falling in it. Now, this query window $[l_1^q, u_1^q] \times [l_2^q, u_2^q]$ is used for finding the results of the query. The MBR of a geographical object 'o is also defined in the similar manner.

$$MBR\ (o) = I_1(o) \times I_2(o),\ where,\ I_i(o) = [l_i, u_i] \qquad (1)$$

The final result is the overlapping areas $(u_1^q >= l_1 \wedge l_1^q <= u_1 \text{ and } u_2^q >= l_2 \wedge l_2^q <= u_2)$ which have at least one overlapping point with the query window.

3.2. Insertion of Newly Created MBR in the Wavelet Tree

Wavelet tree is used for both textual searching and spatial searching techniques. The insertion of a MBR in the already defined wavelet tree involves its addition in appropriate position and then redesigning the whole tree again. The complete process is described through the following algorithms:

The variables used in the algorithms are: u_1 (latitude coordinate of top left endpoint of new node); i (used to store position of leaf node); s_i (used to store symbol at position i); count (used to count number of occurrence of symbols in root node); temp (stores the number of occurrence of a symbol in the node above it); fpos (final position of symbol in the root node); n (number of symbols in root node); newNode (new node); X (root node); and Flag (variable used to store overlapping flag condition).

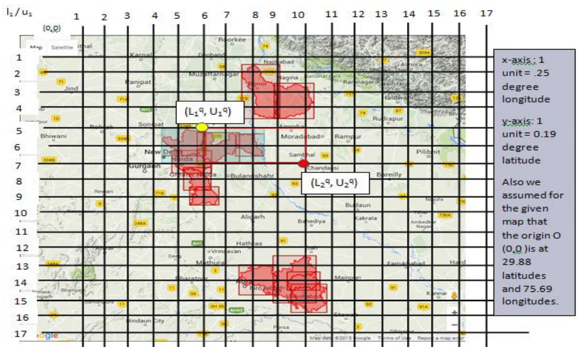
l_1/u_1
(0,0)
(L_1^q, U_1^q)
(L_2^q, U_2^q)
x-axis: 1 unit = .25 degree longitude
y-axis: 1 unit = 0.19 degree latitude
Also we assumed for the given map that the origin O (0,0) is at 29.88 latitudes and 75.69 longitudes.

Figure 2. Example of plotting query window.

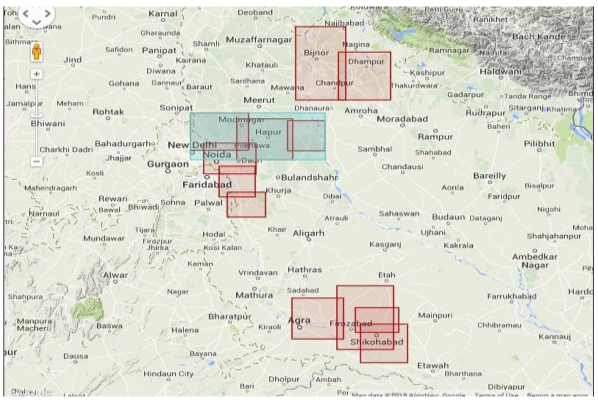

Figure 3. Map showing query window.

Algorithm 2: Create Wavelet Tree for MBR and Remove Overlapped MBR

```
Insert (new, pos)
1.  $u = u_1$ for new node // stores value in a temporary variable
2.  $i = 0$
3.  $s_i$ := Access(i)
4.  while $s_i.u_1 <= u$ // find the position of leaf node at which
    new node to be inserted by searching for the largest value of
    $u_1$ less than u for new node
5.  do i := i + 1
6.  $s_i$ := Access(i)
7.  $s = s_i$ // access the symbol at last value of i
8.  count = $Rank_s(X, n)$ // count the no. of occurrence of symbol
    s
9.  d = count
10.     d = $Select_s(X, d)$ // find the position of last occurrence of
        the symbol in the root node by repeating step 7 till we
        reach the root node
11.     fpos = temp //store the position in a variable
12.     for i = 0 to n
13.     if (Remove_complete_overlap(new, access(i))) // overlap
        from left to right
14.     then flag = 1 // new node completely overlaps an old node
15.     else if (Remove_complete_overlap(access(i), new))
        // overlap from right to left
16.     then flag = 2 // old node completely overlaps new
        defined node
17.     else new = Remove_partial_overlap(new, access(i))
        //remove the overlapping boundaries of new node by
        comparing it with each node in the root
18.     if flag == 1
19.     then Remove old node from the root list
20.     Insert(newNode, fpos)
21.     else if flag == 2
22.     then do nothing
23.     else Insert(newNode, fpos) //insert the new node at the
        position.
```

```
• Remove_partial_overlap (newNoder, oldNode)
1.  if new.l1 < old.l2 and new.l1 > old.l1 and new.l2 > old.l2
2.  then new.l1 = old.l2 //remove the left intersecting part and
    change the left coordinate
3.  if new.l2 < old.l2 and new.l2 > old.l1 and new.l1 < old.l1
4.  then new.l2 = old.l1 //remove the right intersecting part and
    change the right coordinate
5.  if new.u1 < old.u2 and new.u1 > old.u1 and new.u2 >
    old.u2
6.  then new.u1 = old.u2 //remove the top intersecting part and
    change the upper coordinate
7.  if new.u2 < old.u2 and new.u2 > old.u1 and new.u1 <
    old.u1
8.  then new.u2 = old.u1 //remove the bottom intersecting part
    and change the lower coordinate
9.  return new
```

```
• Remove_complete_overlap (A, B) //check if A completely
  covers boundary of B
1.  if $A.l_1 <= B.l_1$ and $A.l_2 >= B.l_2$ and $A.u_1 >= B.u_1$ and $A.u_2$
    $<= B.u_2$)
2.  then return true
3.  else return false
```

3.2.1. Time Complexity Analysis of Proposed Algorithm

Time Complexity for the insertion of a new MBR in the wavelet tree can be calculated by adding the following individual complexities:

1. Complexity for search operation of node using while loop = O(n).
2. Complexity for access operation = O(1).
3. Complexity for rank operation = O(log n).
4. Complexity for select operation = O(log n).
5. Complexity for remove-partial-overlap = O(1).
6. Complexity for remove-complete-overlap = O(1).
7. Complexity for removal operation = O(n).

Adding all these we get the time complexity for insertion of a node in wavelet tree and is given as follows:

$$T(n) = O(\log n) \tag{2}$$

This is better than the existing complexities of R-tree family algorithms. It also removes the problem of MBR overlapping.

3.3. Searching of Query Rectangle in the Wavelet Tree

This section presents a hybrid spatial index to solve geographical queries. MBRs of different geographical points are considered and the data structure used is Wavelet tree. This is same as R-tree except that it increases the response time and reduces the space complexity during execution. The same data structure i.e. wavelet tree is used for both geographical as well as textual searching. Other existing technique uses R-tree family for geographical index and B-tree for inverted index. Through theoretical as well as experimental evaluation it has been proved that proposed hybrid index technique outperforms the existing techniques. This paper does not discuss about the way to find geographic locations from user query because our main focus is to optimize spatial indexes. The pseudo code for index construction and searching geographical document using dual index technique is discussed as given below.

Algorithm 3: Searching Dual Indexes

Suppose user query q has two parts: q1 and q2. i.e. q = q1+q2 where, q1 belongs to textual part of query and q2 belongs to spatial part of query and C[k] is bit map array of wavelet tree index.

```
Search_dual_index (q1, q2)
1.  k ← extract_overlap_id()  //this function will return the id of overlapped MBR.
2.  While(tree!=NULL)
     do
3.      if(C[k]==0)
4.          tree=tree→left
5.          k =Rank(C, tree)
6.      else
7.          tree=tree→right
8.          k=rank(C, tree)
9.  L1=search_mbr_inverted_file(k)    //returns document related to MBR location
10. L2=search_invertedfile(q2) //return documents related to textual query
11. L = L1 ∩ L2
```

The entire process in flowchart form is shown in Figure 4.

3.3.1. Time Complexity of Algorithm

Let, n = Total number of MBRs; and m=number of MBRs intersecting query window; then Time to search the node in the root using binary search = O (log n); and Time to reach leaf of 'm' number of MBRs = m*O(log n). Similarity let q = number of unique keywords; and p = number of documents; then Time to search a keyword = O(log p); and Time for searching the documents for all keywords = q*O(log p). Therefore:

a) Spatial Complexity = (1 + m) O(log n).
b) Textual Complexity = q O(log p).

$$\text{Total Complexity} = (1 + m)\, O(\log n) + q\, O(\log p) \tag{3}$$

Complexity given above is retrieval of spatial followed by textual data using dual indexing and wavelet tree data structure. We have compared the complexity of this dual index with the complexity of existing indexing structures and it is found that our dual index structure out performs in case of larger database and long queries.

The query window extends from [3, 8] to [8, 10] as shown in Figure 5. We can find MBRs 'f' and 'g' to overlap with 'q' using the relation mentioned in the algorithm. latitude wise where row L_1 representing lower and row U_1 represents upper latitudes of each MBRs. Similarly, in array Y, all the MBRs are arranged longitude wise.

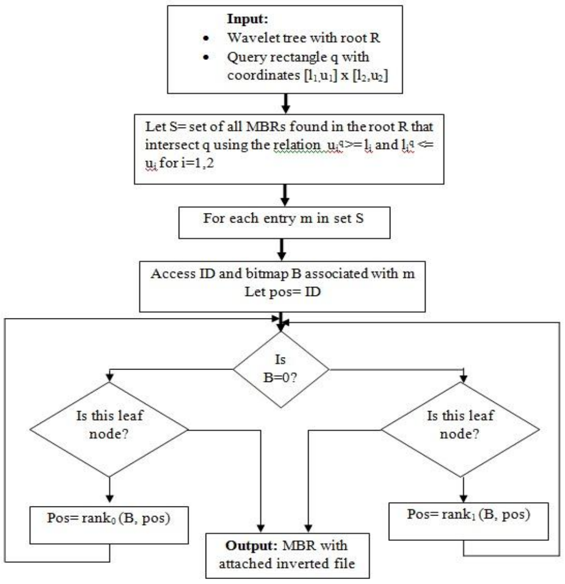
Input:
• Wavelet tree with root R
• Query rectangle q with coordinates $[l_1,u_1] \times [l_2,u_2]$
Let S= set of all MBRs found in the root R that intersect q using the relation $u_i^q >= l_i$ and $l_i^q <= u_i$ for i=1,2
For each entry m in set S
Access ID and bitmap B associated with m
Let pos= ID
Is B=0?
Is this leaf node?
Is this leaf node?
Pos= $rank_0$ (B, pos)
Pos= $rank_1$ (B, pos)
Output: MBR with attached inverted file

Figure 4. Process of searching in a wavelet tree.

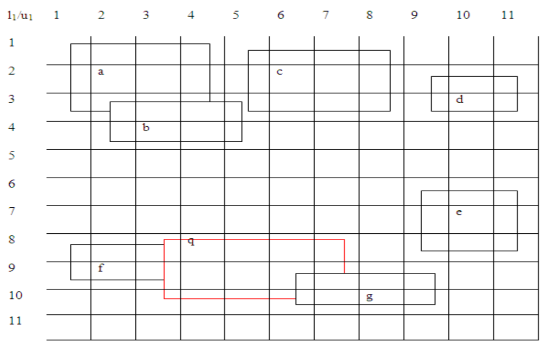

Figure 5. Representation of MBR's.

Array X:

	a	c	d	b	e	f	g
L_1	1	5	9	2	9	1	6
U_1	1	1	2	3	6	8	9

Similarly, Array Y:

	a	c	d	b	e	f	g
L_2	4	8	11	5	11	4	9
U_2	3	3	3	4	8	9	10

Now, we use array X to create wavelet tree for Figure 5. In similar way wavelet tree can also be created on the basis of array Y but while performing the search operation any one of these two-wavelet tree is sufficient enough to provide the complete information.

In the tree (Figure 6), bitmap 0/1 is assigned to each of the entries. To retrieve MBR 'f' we search the root R of the wavelet tree. Since bitmap associated with 'f' is 1, the search space is subdivided and now searching is done in the right child of the tree. In the right sub-tree, since bitmap associated with 'f' is 0, we search in the left sub-tree. We keep on searching in the left and right sub-tree depending on the bitmap associated with MBR until we finally reach the leaf node where we can access all documents in the MBR. Each node has an inverted file attached to it which has all the documents contained in that node.

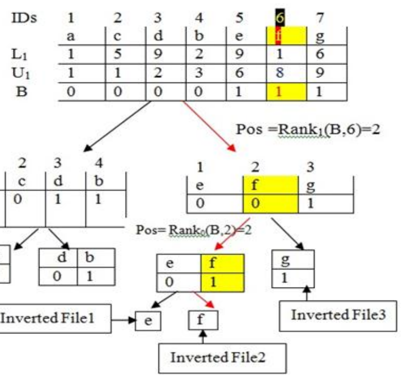

Figure 6. Initial wavelet tree.

- *Example* 1-a: Suppose we want to resolve the query hotels in Ghaziabad. Algorithm form query window for spatial location Ghaziabad and then search for all MBRs overlapping this query window to retrieve documents related to Ghaziabad as shown in Table 2.

Table 1. Vocabulary.

Vocabulary	Inverted File 2	Inverted File 3
Ghaziabad	<doc1,doc2,doc5,doc7>	<doc2,doc5,doc7>
Delhi	<doc6,doc8>	<doc2,doc3,doc5>
Noida	<doc2,doc8,doc9>	<doc1,doc2,doc7>
Gurgaon	<doc5>	<doc6,doc8,doc9>

Let, while searching the location Ghaziabad the MBRs 'f' and 'g' are overlapped with the query window formed for it. As explained above, these MBRs produces two inverted indexes, Inverted File 1 and Inverted file 2 respectively. As shown in Table 1 these inverted files contain the documents <doc1, doc2, doc5, doc7> and <doc2, doc5, doc7> respectively. So merging is performed to find the common documents as shown below:

<doc1, doc2, doc5, doc7> && <doc2, doc5, doc7> = {doc2, doc5, doc7} ....(Result-1)

- *Example* 1-b: Now suppose we have an inversion list (as shown in Table 2) which contains a list of keywords along with their ids and the documents which contain these keywords.

Table 2. Inverted list.

Keyword No.	Keyword	Document
1	We	1
2	Flower	1, 3
3	Hotels	1, 2
4	Restaurants	2
5	Our	2
6	Home	2, 4
7	I	3, 4, 5
8	My	3
9	You	4
10	Dog	4
11	Book	4, 5
12	Coffee	5
13	Tea	5

The wavelet tree for all documents shown in Table 2 is created as shown in Figure 7. Thereafter it is needed to retrieve the documents that contain the keyword hotels and finally intersection on documents, retrieved by result-1 and result-2 (explained below) is performed to get the final result for the query.

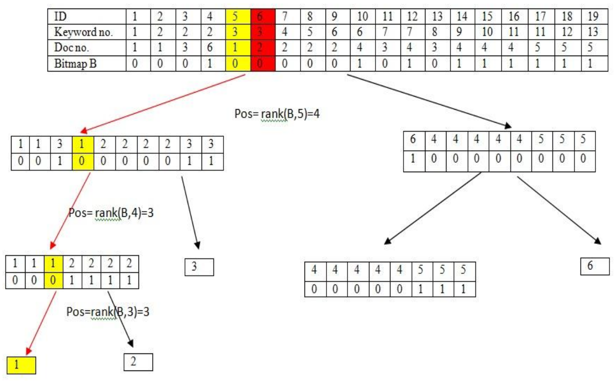

Figure 7. Complete wavelet tree.

We store a unique ID with each entry in the root R to solve the rank operation efficiently. In the above constructed wavelet tree, it is clearly visible that keyword hotels with keyword number 3 occurs in document 1 and 2. For keyword number 3, we access the bitmap for 3. Since it is 0, we move to the left child of the tree and pos=rank0(B, ID)=rank$_0$(B, 5)=4. So, we again access position 4 in the left child. Here B=0, so we move to the left child and find pos=rank$_0$(B, 4)= 3. This goes on recursively until we reach the leaf node where we get our required document. In the above example, documents containing keyword hotels are, Doc1 and Doc2 (result-II). After performing merge operation on result-I and result-II i.e. {doc2, doc5, doc7} && {doc1, doc2} = {doc2}. Therefore, doc2 resolves the query hotels in Ghaziabad.

- **Time Complexity**

For spatial indexing, suppose we have n MBRs in the root R of the wavelet tree and out of these 'n' MBRs, 'm' MBRs intersect our query window 'q'. For rank operation the time required is constant and equal to O(1).

In the above Example 1-a: we get 'f', and 'g' as a result of overlapping MBRs. Hence, the total time complexity is O(log n) + mO(log n) = (1+m)O(log n).

In example 1(b), binary search in the root requires q*O(log p) time for 'p' keywords which are contained in 'q' documents and traversal of tree using rank operation only takes O(1). Therefore, the total complexity of dual indexing is (1+m)*O(log n)+q*O(log p).

4. Experimental Results

We have implemented our proposed algorithm in PHP with database MYSQL 5.6.12 and server Apache 2.4.4, executed on windows 8-64bit, AMD A8 Quad core 1333 MHz processor with 4GB RAM. The results obtained are compared to the existing structures for spatial indexes. In this paper, we have compared the wavelet tree based technique with R-tree and R*-tree based techniques. The result is shown in Table 3 and Figure 8.

We have also proposed dual indexing technique using common structure for both textual and spatial i.e., wavelet tree. This dual indexing technique is also compared to other existing dual pairs. The result is shown in Table 4 and Figure 9.

Table 3. Searching time (ms) of different algorithms.

Query Length	R-tree	R*-Tree	Wavelet tree
2	122	98	24
3	139	120	38
4	149	189	50
5	195	208	56
6	254	265	78
7	320	310	112
8	446	412	137
9	522	473	146
10	698	545	155
11	770	603	166
12	872	691	187
13	950	762	195
14	1098	835	211
15	1190	895	241

Table 4. Searching time of dual indexing techniques.

Query length	B/R	B/R*	Dual Wav/Wav
2	155	139	161
3	229	184	182
4	311	278	193
5	399	362	221
6	526	443	234
7	645	548	279
8	877	712	318
9	945	832	402
10	1022	906	469
11	1290	1011	534
12	1401	1265	599
13	1506	1381	690
14	1720	1500	783
15	1795	1605	916

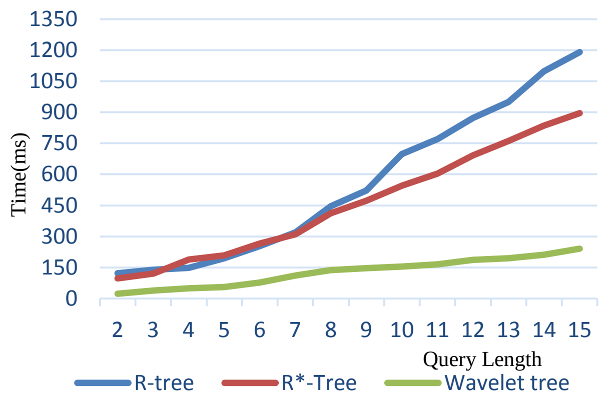

Figure 8. Graph showing the variation in searching time.

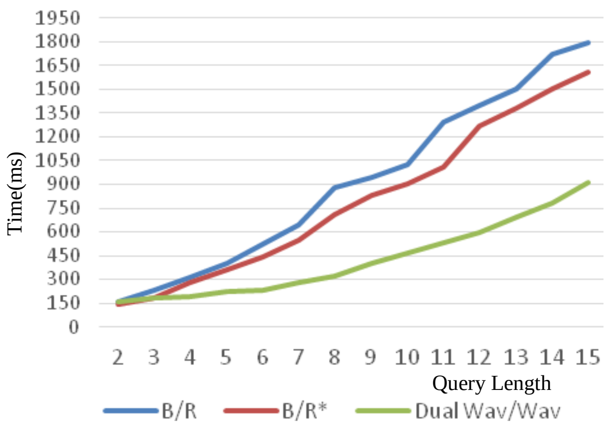

Figure 9. Graph showing the variation in searching time of dual indexing techniques.

5. Conclusions and Future Work

In this paper, we have proposed and implemented an spatial indexing technique based on wavelet tree and have also given a dual indexing technique using common indexing structure i.e. wavelet tree both for textual as well as spatial indexing. We have also designed an algorithm for the dynamic insertion of MBR in the wavelet tree. The experiments result presented shows that algorithms outperform the existing algorithms both in terms of simplicity in implementation and searching time. For example, for pure spatial indexing, using wavelet tree, the search time complexity is reduced and it takes even less than one third time of that of spatial indexing performed using R-tree or R*-tree (Table 4). Even in case of dual indexing (textual and spatial) using wavelet tree, the search time is reduced by half in comparison to other existing techniques used for the dual indexing such as B/R, B/R* when the search query length is more than 2 keywords. In case the query is of 1 or 2 keywords, the search time remains approximately similar to that of other techniques.

Our future work is to develop hybrid technique based on common structure for both textual and spatial data and assign multiple geographical references.

References

[1] Andrade L. and Silva M., "Indexing Structures for Geographic Web Retrieval," *in Proceedings of the Conference on Mobile and Ubiquitous Systems*, Guimarães, pp. 33-39, 2006.

[2] Bliujute R., Jensen C., Saltenis S., and Slivinskas G., "R-Tree Based Indexing of Now-Relative Bitemporal Data," *in Proceedings of the 24rd International Conference on Very Large Data Bases*, San Francisco, pp. 345-356, 1998.

[3] Brisaboa N., Luaces M., Navarro G., and Seco D., "A New Point Access Method Based on Wavelet Trees," *Advances in Conceptual*

Modeling-Challenging Perspectives, Berlin, pp. 297-308, 2009.

[4] Brisaboa N., Luaces M., Navarro G., and Seco D., "A Fun Application of Compact Data Structures to Indexing Geographic Data," *in Proceedings of the 5th International Conference on Fun with Algorithms*, Iscia, pp. 77-88, 2010.

[5] Claude F., Navarro G., and Ordónez A., "The Wavelet Matrix: An Efficient Wavelet Tree for Large Alphabets," *Information Systems*, vol. 47, pp. 15-32, 2015.

[6] Elabd E., Alshari E., and Abdulkader H., "Semantic Boolean Arabic Information Retrieval," *The International Arab Journal of Information Technology*, vol. 12, no. 3, pp. 311-316, 2015.

[7] Gagie T., Navarro G., and Puglisi S., "New Algorithms on Wavelet Trees and Applications to Information Retrieval," *Theoretical Computer Science*, vol. 426-427, pp. 25-41, 2012.

[8] Gaede V. and Gjnther O., "Multidimensional Access Methods," *ACM Computing Surveys*, vol. 30, no. 2, pp. 170-231, 1998.

[9] Hariharan R., Hore B., Li C., and Mehrotra S., "Processing Spatial-Keyword (SK) Queries in Geographic Information Retrieval (GIR) Systems," *in Proceedings of 19th International Conference on Scientific and Statistical Database Management*, Alta, pp. 16, 2007.

[10] Jones C., Abdelmoty A., Finch D., Fu G., and Vaid S., "The SPIRIT Spatial Search Engine: Architecture, Ontologies and Spatial Indexing," *in Proceedings of Geographic Information Science*, Berlin, pp. 125-139, 2004.

[11] Lieberman M., Samet H., Sankaranarayanan J., and Sperling J., "STEWARD: Architecture of A Spatio-Textual Search Engine," *in Proceedings of the 15th Annual ACM International Symposium on Advances in Geographic Information Systems*, Seattle, pp. 186-193, 2007.

[12] Lin X., Yu B., and Ban Y., "On Indexing Mechanism in Geographical Information Retrieval System," *in Proceedings of 10th AGILE International Conference on Geographic Information Science*, Denmark, pp. 1-3, 2007.

[13] Navarro G., "Wavelet Trees for All," *Journal of Discrete Algorithms*, vol. 25, pp. 2-20, 2014.

[14] Papadias D., Sellis T., Theodoridis Y., and Egenhofer M., "Topological Relations in The World of Minimum Bounding Rectangles: A Study With R-Trees," *in Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Jose, pp. 92-103, 1995.

[15] Su Q. and Widom J., "Indexing Relational Database Content Offline for Efficient Keyword-Based Search," *in Proceedings of 9th International Database Engineering and Application Symposium*, Montreal, pp. 297-305, 2005.

[16] Yadav A. and Yadav D., "Wavelet Tree based Hybrid Geo-Textual Indexing Technique for Geographical Search," *Indian Journal of Science and Technology*, vol. 8, no. 33, pp. 1-7, 2015.

[17] Yadav A., Yadav D., and Prasad R., "Efficient Textual Web Retrieval using Wavelet Tree," *International Journal of Information Retrieval Research*, vol. 6, no. 4, pp. 16-29, 2016.

[18] Zhou Y., Xie X., Wang C., Gong Y., and Ma W., "Hybrid Index Structures for Location-Based Web Search," *in Proceedings of the 14th ACM International Conference on Information and Knowledge Management ACM*, Bremen, pp. 155-166, 2005.

Arun Kumar Yadav is Associate Professor in the department of Computer Science and Engineering at Ajay Kumar Garg Engineering College, Ghaziabad (UP) India. He received PhD. degree in Computer Science and Engineering 2016. He guided 1 M. Tech. dissertation and published more than 10 research papers in reputed international/national journals and presented at conferences. His areas of interest are Information retrieval and Database Management System.

Divakar Yadav is Associate Professor in the department of Computer Science and Engineering at Madan Mohan Malaviya University of Technology, Gorakhpur (UP) India. He received PhD. degree in Computer Science and Engineering 2010. He spent one year, from Oct 2011 to Oct 2012, in Carlos III University, Leganes, Madrid, Spain as a postdoctoral fellow. He guided four PhD students, 14 M.Tech. dissertation and published more than 65 research papers in reputed international/national journals and presented at conferences. His areas of interest are Information retrieval and soft computing.