International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2015): 78.96 | Impact Factor (2015): 6.391

# A Survey on Nearest Neighbor Search with Keywords

**Shimna P. T[1], Dilna V. C[2]**

[1, 2]AWH Engineering College, KTU University, Department of Computer Science & Engineering, Kuttikkatoor, Kozhikode, India

**Abstract:** ***Many applications require finding objects closest to a specified location that contains a set of keywords. Today, many modern applications call for novel forms of queries that aim to find objects satisfying both a spatial predicate, and a predicate on their associated texts. The problems of nearest neighbor search on spatial data and keyword search on text data have been extensively studied separately. In this work, we present an efficient method to answer top-k spatial keyword queries. To do so, we introduce an indexing structure called IR2-Tree (Information Retrieval R-Tree) which combines an R-Tree with superimposed text signatures. To increase the efficiency of nearest neighbor search we develop a new access method called the spatial inverted index that extends the conventional inverted index to cope with multidimensional data, and comes with algorithms that can answer nearest neighbor queries with keywords in real time. To answer mCK m-closest keywords queries efficiently, we introduce a new index called the bR*-tree, which is an extension of the R*-tree.***

**Keywords:** Nearest Neighbor Search, Keyword Search, Spatial Index, bR*-tree, IR2-Tree

## 1. Introduction

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

We present a method to efficiently answer top-k spatial keyword queries, which is based on the tight integration of data structures and algorithms used in spatial database search and Information Retrieval (IR). In particular, our method consists of building an Information Retrieval R Tree (IR2-Tree), which is a structure based on the R-Tree [Gut84]. At query time an incremental algorithm is employed that uses the IR2-Tree to efficiently produce the top results of the query.

The IR2-tree, however, also inherits a drawback of signature files: *false hits*. That is, a signature file, due to its conservative nature, may still direct the search to some objects, even though they do not have all the keywords. The penalty thus caused is the need to *verify* an object whose satisfying a query or not cannot be resolved using only its signature, but requires loading its full text description, which is expensive due to the resulting random accesses. In this paper, we design a variant of inverted index that is optimized for multidimensional points, and is thus named the *spatial inverted index* (SI-index). This access method successfully incorporates point coordinates into a conventional inverted index with small extra space, owing to a delicate compact storage scheme. Meanwhile, an SI-index preserves the spatial locality of data points, and comes with an R-tree built on every inverted list at little space overhead.

Current research on queries goes well beyond pure spatial queries such as nearest neighbor queries [29], range queries [25], and spatial joins [24], [28], [27], [26]. Queries on spatial objects associated with textual information represented by sets of keywords are beginning to receive significant attention from the spatial database research community and the industry. This paper focuses on a novel type of query called the ***m*closest keywords** (*m*CK) query: given *m* keywords provided by the user, the *m*CK query aims at finding the closest tuples (in space) that match these keywords. While such a query has various applications, our main interest lies in that of a **search by document**.

## 2. IR² tree

The IR2-Tree is a combination of an R-Tree and signature files. In particular, each node of an IR2-Tree contains both spatial and keyword information; the former in the form of a minimum bounding area and the latter in the form of a signature. An IR2-Tree facilitates both top-k spatial queries and top-k spatial keyword queries as we explain below. More formally, an IR2-Tree R is a height-balanced tree data structure, where each leaf node has entries of the form (ObjPtr, A, S). ObjPtr and A are defined as in the R-Tree while S is the signature of the object referred by ObjPtr. A non-leaf node has entries of the form (NodePtr, A, S). NodePtr and A are defined as in the R-Tree while S is the signature of the node. The signature of a node is the superimposition (OR-ing) of all the signatures of its entries. Thus a signature of a node is equivalent to a signature for all the documents in its subtree.


Licensed Under Creative Commons Attribution CC BY

### 2.1 IR2-Tree Algorithm

In this section we present the general version of the IR2-Tree algorithm, where objects are output ordered by a ranking function f(distance(T.p, Q.p), IRscore(T.t, Q.t)) as defined. The key differences to the distance-first version are that:

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

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

Upper(v)=UpperBoundT∈ v(f(distance(T.p,Q.p), IRscore(T.t, Q.t))

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

Upper(v)=LowerBoundT∈ v(f(distance(v.MBR,Q.p), UpperBoundT-hassignature-v.S(IRscore(T.t, Q.t)))

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

## 3. Spatial Inverted List

The IR2-tree is the first access method for answering NN queries with keywords. As with many pioneering solutions, the IR2-tree also has a few drawbacks that affect its efficiency. The most serious one of all is that the number of false hits can be really large when the object of the final result is faraway from the query point, or the result is simply empty. In these cases, the query algorithm would need to load the documents of many objects, incurring expensive overhead as each loading necessitates a random access.

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

**Algorithm 1:** Distance-First IR2-Tree algorithm

The spatial inverted list (SI-index) is essentially compressed version of an I-index with embedded coordinates as described. Query processing with an SI index can be done either by merging, or together with R-trees in a distance browsing manner. Furthermore, the compression eliminates the defect of a conventional Iindex such that an SI-index consumes much less space.

Compression is already widely used to reduce the size of an inverted index in the conventional context where each inverted list contains only ids. In that case, an effective approach is to record the *gaps* between consecutive ids, as opposed to the precise ids. For example, given a set S of integers {2, 3, 6, 8}, the gap-keeping approach will store {2, 1, 3, 2} instead, where the i-th value (i _ 2) is the difference between the i-th and (i – 1)-th values in the original S. As the original S can be precisely reconstructed, no information is lost. The only overhead is that decompression incurs extra computation cost, but such cost is negligible compared to the overhead of I/Os. Note that gap-keeping will be much less beneficial if the integers of S are not in a sorted order. This is because the space saving comes from the hope that gaps would be much smaller (than the original values) and hence could be represented with fewer bits. This would not be true had S not been sorted.

Compressing an SI-index is less straightforward. The difference here is that each element of a list, a.k.a. a point p, is a triplet (idp, xp, yp), including both the id and coordinates of p. As gap-keeping requires a sorted order, it can be applied on only one attribute of the triplet. For example, if we decide to sort the list by ids, gap-keeping on ids may lead to good space saving, but its application on the x- and y-coordinates would not have much effect .

To attack this problem, let us first leave out the ids and focus on the coordinates. Even though each point has 2


Licensed Under Creative Commons Attribution CC BY

coordinates, we can convert them into only one so that gap keeping can be applied effectively. The tool needed is a space filling curve (SFC) such as Hilbert- or Z-curve. SFC converts a multidimensional point to a 1D value such that if two points are close in the original space, their 1D values also tend to be similar. As dimensionality has been brought to 1, gap-keeping works nicely after sorting the (converted) 1D values.

For example, based on the Z-curve2, the resulting values, called *Z-values*, of the points in  demonstrated in Figure 5 in ascending order. With gapkeeping, we will store these 8 points as the sequence 12, 3, 8, 1, 7, 9, 2, 7. Note that as the Z-values of all points can be accurately restored, the exact coordinates can be restored as well.

Let us put the ids back into consideration. Now that we have successfully dealt with the two coordinates with a 2D SFC, it would be natural to think about using a 3D SFC to cope with ids too. As far as space reduction is concerned, this 3D approach may not a bad solution. The problem is that it will destroy the locality of the points in their original space. Specifically, the converted values would no longer preserve the spatial proximity of the points, because ids in general have nothing to do with coordinates.

## 4. BR*-Tree

Spatial data is almost always indexed to facilitate fast retrieval. We can adopt the idea of Papadias et al. [26] to answer the *m*CK query. Given *N* R*-trees, one for each keyword, candidate spatial windows for the *m*CK query result can be identified by executing multiway spatial joins (MWSJ) among the R*-trees. The join condition here becomes “closest in space” instead of “overlapping in space” [26]. When *m* is very small, this approach accesses only a small portion of the data and returns the result relatively quickly. However, as *m* increases, this approach suffers from two serious drawbacks. First, it incurs high disk I/O cost for identifying the candidate windows (due to synchronous multiway traversal of R*-trees) since it does not inherently support effective summarization of keyword locations. Second, it may not be able to identify a  “tight” set of candidate windows since it determines candidate windows in an approximate manner based on the leaf-node MBRs of R*-trees without considering the actual objects. To process *m*CK queries in a more scalable manner, we propose to use one R*-tree to index all the spatial objects as well as their keywords. Integrating all the information in a single R*- tree provides more opportunities for efficient search and pruning.

To process *m*CK queries in a more scalable manner, we propose to use one R*-tree to index all the spatial objects and their keywords. In this section, we discuss the proposed index structure called the *bR*-tree*. The bR*-tree is an extension of the R*-tree. Besides the node MBR, each node is augmented with additional information. A straightforward extension is to summarize the keywords in the node. With this information, it becomes easy to decide whether *m* query keywords can be found in this node. If there are *N* keywords in the database, the keywords for each node can be represented using a bitmap of size *N*, with a “1” indicating its existence in the node and a “0” otherwise. For example, a bitmap $B = 01001$ reveals that there are five keywords in the database and the current node can only be associated with the keywords in the second and fifth positions of the bitmap. This representation incurs little storage overhead. Moreover, it can accelerate the checking process of keyword constraints due to the relatively high speed of binary operations. Given a query $Q = 00110$, if we have $B$ AND $Q = 0$, it implies that the given node does not have any query keywords and thus, this node can be eliminated from the search space.

Besides the keyword bitmap, we also store the *keyword MBR* in the node to set up more powerful pruning rules. The keyword MBR of keyword *wi* is the MBR for all the objects in the nodes that are associated with *wi*. It summarizes the spatial locations of *wi* in the node. Using this information, we know the approximate area in the node which each keyword is distributed. If *M* is the node MBR and *Mi* is the keyword MBR for *wi*, we have $Mi \subseteq M$. When *N* is a large number, the cost for storing the keyword MBR is very high. For example, suppose there are a total of 100 keywords in the database and the objects are three dimensional data. Spatial coordinates are usually stored I double precision, which occupies eight bytes per coordinate.

It would therefore take $100 \times 3 \times 8 \times 2 = 4800$ bytes to store the keyword MBRs in one node. To reduce the storage cost, we split the node MBR into segments along each dimension. Each keyword MBR is represented approximately by the start and end offsets of the segments along each dimension. The range of an offset that occupies *n* bits is $[0, 2^n - 1]$. In our implementation, we set $n = 8$ (resulting in 256 segments) and found that it provided satisfactory approximation. After being augmented with the bitmap and keyword MBR, non-leaf nodes of the bR*-tree contain entries of the form (*ptrs, mbr, bmp, kwd mbr*), where

- *ptrs* are pointers to child nodes;
- *mbr* is the node MBR that covers all the MBRs in the child nodes;
- *bmp* is a keyword bitmap, each bit of which corresponds to a specific keyword, and is marked as “1” if the MBR of the node contains the keyword and “0” otherwise;
- *kwd mbr* is the vector of keyword MBR for all the keywords contained in the node.

Fig. 1 depicts an example of an internal node containing three keywords *w*1, *w*2, *w*3 represented as 111. It also maintains the keyword MBRs of *w*1, *w*2 and *w*3. The keyword MBR of *wi* is a spatial bound of all the objects with keyword *wi*. Leaf nodes contain entries of the form (*oid, loc, bmp*), where

- *oid* is a pointer to an object in the database;
- *loc* represents the coordinates of the object;
- *bmp* is the keyword bitmap.


Licensed Under Creative Commons Attribution CC BY

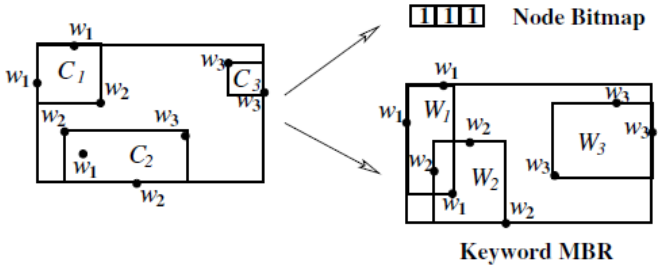


**Figure 1:** Node information of the bR*-tree

### 4.1 Searching in One Node

When searching in one node, our task is to enumerate all the subsets of its child nodes in which it is possible to find *m* closer tuples matching the query keywords. The subsets which contain all the *m* keywords and whose child nodes are close to each other are considered as candidates. There is also a constraint that the number of nodes in a subset should not exceed *m*. In order to take advantage of the a priori algorithm, we define two monotonic constraints called **distance mutex** and **keyword mutex**. If a node set *N* = *{N*1*,N*2*, . . .,Nn}* is distance mutex or keyword mutex, then any superset of *N* is also distance mutex or keyword mutex and can be pruned.

The method for searching in one node is shown in Algorithm 3. First (in line 1), we put all the child nodes in the bottom level of the lattice. The lattice is built level by level with increasing number of child nodes in the *NodeSet*. In level *i*, each *NodeSet* contains exactly *i* child nodes. For a query with *m* keywords, we only need to check *NodeSet* with at most *m* nodes, leading to a lattice with at most *m* levels. Lines 5–6 show two sets *C*1 and *C*2 in level *i* − 1 being joined, they must have *i* − 2 nodes in common. Lines 7–14 check if any of its subsets in level *i*−1 is pruned due to distance mutex or keyword mutex. If all the subsets are legal, we check whether this new candidate itself is distance mutex or keyword mutex for pruning. If it is not pruned, we add it to level *i*. In lines 19–22, after all the candidates have been generated, we check each one to see if it contains all the query keywords. Those missing any keywords are eliminated. We do not check this constraint while building the lattice because if a node does not contain all the query keywords, it can still combine with other nodes to cover the missing keywords. As long as it is neither distance mutex nor keyword mutex, we keep it in the lattice.

### 4.2 Searching In Multiple Nodes

Algorithm 4 shows how a set of *n* nodes *{N*1*, . . . , Nn}* is explored. First, *n* lists of ordered subsets of child nodes are obtained. Then Algorithm 5 is invoked to enumerate all the candidate sets. It is implemented in a recursive manner.

**Algorithm 3 — SearchInOneNode: Searching in One Node**

**Input:** A node $N$ in bR*-tree
**Output:** A list of new NodeSets
1. $L_1$ = all the child nodes in N
2. **for** $i$ from 2 to $m$ **do**
3. **for** each $NodeSet\ C_1 \in L_{i-1}$ **do**
4. **for** each $NodeSet\ C_2 \in L_{i-1}$ **do**
5. **if** $C_1$ and $C_2$ share the first $i-1$ nodes **then**
6. $C = NodeSet(C_1, C_2)$
7. **if** $C$ has subset not appear in $L_{i-1}$ **then**
8. continue
9. **if** $C$ is not distance mutex **then**
10. **if** $C$ is not keyword mutex **then**
11. $L_i = L_i \cup C$
12. **for** each $NodeSet\ S \in \cup_{i=1}^{m} L_i$ **do**
13. **if** $S$ contains all the query keywords **then**
14. add $S$ to $cList$
15. return $cList$

**Algorithm 3:** Algorithm for searchInOneNode

**Algorithm 4 — SearchInMultiNodes: Search In Multiple Nodes**

**Input:** A set of $\{N_1, \ldots, N_n\}$ in bR*-tree
**Output:** A list of new NodeSets
1. **for** each node $N_i$ **do**
2. $L_i$ = SearchInOneNode($N_i$)
3. perform an initial filtering on $L_i$
4. return Enumerate($L_1, \ldots, L_n$, $n$, NULL)

**Algorithm 4:** Algorithm for SearchInMultiNodes

Each time an enumerated candidate is generated, we check if it contains all the query keywords to decide whether to prune it or to put it in the candidate list(see Lines 1–4). Lines 5–8 indicate the beginning of the recursion process. It starts from each child node subset in list *Ln* and makes it as our current partial node set *curSet*. *curSet* recursively combines with other child node subsets until it finally contains child nodes from *{N*1*, . . .,Nn}*. In each recursion, we iterate the child node sets in list *Li* to combine with *curSet* and generate a new set denoted as *newSet*. Lines 12–13 show that if *newSet* already has more than *m* child nodes, we stop the iteration because the list is ordered. The child node subsets which are not checked could only have more child nodes and will result in even more nodes in *newSet*. Otherwise, we check if any subsets of *newSet* have been pruned due to distance mutex or keyword mutex. If not, we go on checking whether this new *NodeSet* itself is distance mutex or keyword mutex. All these checking processes are shown in Lines 14–17. If *newSet* is not pruned, we set it as *curSet* and continue the recursion. Finally, the algorithm returns all the candidates that were not pruned away. In the following subsections, we propose two novel methods to efficiently check whether a set is distance mutex or keyword mutex.


Licensed Under Creative Commons Attribution CC BY

**Algorithm 5 — Enumerate: Enumerate All Possible Candidates**

**Input:** $n$ lists of sets of child nodes $L_1, \ldots, L_n$, count and $curSet$
**Output:** A list of new NodeSets

```
1. if count = 0 then
2.   if curSet contains all the query keywords then
3.     push curSet into the candidate list cList
4.     return
5. if count = n then
6.   for each NodeSet S ∈ L_n do
7.     curSet = S
8.     Enumerate(L_1, ..., L_n, count−1, curSet)
9. else
10.  for each NodeSet S ∈ L_n do
11.    newSet = NodeSet(curSet, S)
12.    if newSet contains more than m nodes then
13.      break
14.    if newSet has any illegal subset candidate then
15.      continue
16.    if newSet is not distance mutex then
17.      if newSet is not keyword mutex then
18.        Enumerate(L_1, ..., L_n, count−1, newSet)
19. return cList
```

**Algorithm 5:** Algorithm for Enumerate Candidates

## 5. Conclusions

We introduced the problem of spatial keyword search and explained the performance limitations of current approaches. We proposed a solution which is dramatically faster than current approaches and is based on a combination of R-Trees and signature files techniques. An efficient incremental algorithm was presented that uses the IR2-Tree to answer spatial keyword queries. In second method, we have remedied the situation by developing an access method called *the spatial inverted index* (SI-index). Not only that the SI-index is fairly space economical, but also it has the ability to perform keyword-augmented nearest neighbor search in time that is at the order of dozens of milliseconds. we take a step towards searching by document by addressing the *m*CK query. We use the bR*-tree to effectively summarize keyword locations, thereby facilitating pruning. We propose effective a priori- based search strategies for *m*CK query processing. While handling large number of keywords is an important step towards searching by document, there is still much room for future research.

## References

[1] A. J. Broder. Strategies for efficient incremental neighbor search. In Pattern Recognition, 23(1–2):171–178,January 1990.
[2] Nicolas Bruno, Luis Gravano, Amelie Marian. Evaluating Top-k Queries over Web-Accessible Databases., ICDE 2002.
[3] U. Deppisch. S-Tree: A dynamic balanced signature index for office retrieval. In Proc. of the ACM Conf. on Research and Development in Information Retrieval, Pisa, 1986.
[4] Ron Sacks-Davis, Kotagiri Ramamohanarao: A two level superimposed coding scheme for partial match retrieval. Inf.Syst. 8(4): 273-289 (1983).
[5] Ronald Fagin, Amnon Lotem, Moni Naor: Optimal Aggregation Algorithms for Middleware. In PODS 2001.
[6] Christos Faloutsos: Signature files: Design and PerformanceComparison of Some Signature Extraction Methods. In SIGMOD Conference 1985.
[7] Christos Faloutsos, Stavros Christodoulakis: Signature Files: An Access Method for Documents and Its Analytical Performance Evaluation. In ACM Trans. Inf. Syst. 2(4): 267-288(1984) S.
[8] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In Proc. of ACM Management of Data (SIGMOD), pages 322–331,1990.
[9] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks.In Proc. of International Conference on Data Engineering (ICDE), pages 431–440, 2002.
[10] X. Cao, L. Chen, G. Cong, C. S. Jensen, Q. Qu, A. Skovsgaard, D. Wu, and M. L. Yiu. Spatial keyword querying. In ER, pages 16–29, 2012. X. Cao, G. Cong, and C. S. Jensen. Retrieving top-k prestige-based relevant spatial web objects. PVLDB, 3(1):373–384, 2010.
[11] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In Proc. of ACM Management of Data (SIG- MOD), pages 373–384, 2011.
[12] B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal. The bloomier filter: an efficient data structure for static support lookup tables. In Proc. of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 30–39, 2004.
[13] Y.-Y. Chen, T. Suel, and A. Markowetz. Efficient query processing in geographic web search engines. In Proc. of ACM Management of Data (SIGMOD), pages 277–288, 2006.
[14] E. Chu, A. Baid, X. Chai, A. Doan, and J. Naughton. Combining keyword search and forms for ad hoc querying of databases. In Proc. of ACM Management of Data (SIGMOD), 2009.
[15] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. PVLDB, 2(1):337–348, 2009.
[16] C. Faloutsos and S. Christodoulakis. Signature files: An access method for documents and its analytical performance evaluation. ACM Transactions on Information Systems (TOIS), 2(4):267–288, 1984.
[17] I. D. Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In Proc. of International Conference on Data Engineering (ICDE), pages 656–665, 2008.
[18] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing spatialkeyword (SK) queries in geographic information retrieval (GIR) systems. In Proc. of Scientific and Statistical Database Management (SSDBM), 2007.


Licensed Under Creative Commons Attribution CC BY

[19] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. ACM Transactions on Database Systems (TODS), 24(2):265–318, 1999.
[20] W. Aref, D. Barbara, S. Johnson, and S. Mehrotra. Efficient processing of proximity queries for large databases. Proc. ICDE, pages 147–154, 1995.
[21] W. Aref and H. Samet. Efficient processing of window queries in the pyramid data structure. Proc. PODS, pages 265–272, 1990.
[22] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. Proc SIGMOD, pages 322–331, 1990.
[23] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. SudarshanKeyword searching and browsing in databases using banks. In Proceedings of ICDE, 2002.
[24] T. Brinkhoff, H. Kriegel, and B. Seeger. Efficient processing of spatial joins using R-trees. Proc. SIGMOD, pages 237–246, 1993.
[25] B.-U. Pagel, H.-W. Six, H. Toben, and P. Widmayer. Towards an analysis of range query performance in spatial data structures. In PODS, pages 214–221, New York, NY, USA, 1993. ACM.
[26] D. Papadias and D. Arkoumanis. Approximate processing of multiway spatial joins in very large databases. Proc. EDBT, pages 179–196, 2002.
[27] N. Mamoulis and D. Papadias. Multiway spatial joins. Proc. TODS, 26(4):424–475, 2001.
[28] D. Papadias, N. Mamoulis, and Y. Theodoridis. Processing and optimization of multiway spatial joins using R-trees. Proc. PODS, pages 44–55, 1999.
[29] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. Proc. SIGMOD, pages 71–79, 1995.

## Author Profile

**Shimna P T** received the B Tech degree in Information Technology from Anna University in 2014. Currently pursuing M Tech in Computer Science and Engineering from KTU university, respectively.

**Dilna V C** received the B Tech and M Tech degree in Computer Science and Engineering from Calicut university in 2010 and 2012 respectively. Now she is currently working as assistant professor in AWH Engineering College.


Licensed Under Creative Commons Attribution CC BY