Research Journal of Applied Sciences, Engineering and Technology 10(2): 138-150, 2015
DOI:10.19026/rjaset.10.2565
ISSN: 2040-7459; e-ISSN: 2040-7467

Submitted: September 13, 2014 Accepted: November 26, 2014 Published: May 20, 2015

Research Article

An Approach to Indexing and Retrieval of Spatial Data with Reduced *R+ Tree* and K-NN Query Algorithm

[1]S. Palaniappan, [2]T.V. Rajinikanth and [3]A. Govardhan
[1]Department of CSE, Saveetha University, Chennai,
[2]Department of CSE, SNIST, Hyderabad,
[3]Department of SIT, JNTUH, Kukatpally, Hyderabad-500085, Telangana, India

Abstract: Recently, "*spatial data* bases have been extensively adopted in the recent decade and various methods have been presented to store, browse, search and retrieve spatial objects". In this study, a method is plotted for retrieving nearest neighbors from *spatial data* indexed by *R+ tree*. The approach uses a reduced *R+ tree* for the purpose of representing the *spatial data*. Initially the *spatial data* is selected and *R+ tree* is constructed accordingly. Then a function called joining nodes is applied to reduce the number of nodes by combining the half-filled nodes to form completely filled. The idea behind reducing the nodes is to perform search and retrieval quickly and efficiently. The reduced *R+ tree* is then processed with KNN query algorithm to fetch the nearest neighbors to a point query. The basic procedures of KNN algorithm are used in the proposed approach for retrieving the nearest neighbors. The proposed approach is evaluated for its performance with *spatial data* and results are plotted in the experimental analysis section. The experimental results showed that the proposed approach is remarkably up a head than the conventional methods. The maximum time required to index the 1000 data points by the *R+ tree* is 10324 ms. The number of nodes possessed by reduced *R+ tree* is also less for 1000 data points as compared to the conventional *R+ tree* algorithm.

Keywords: Indexing, KNN algorithm, query, *R+ tree*, *spatial data*

INTRODUCTION

The *Spatial Data* Base System (SDBS) (Corral and Almedros-Jimenez, 2007) has recently surfaced as a database system which furnishes *spatial data* types in its data version and query language and possesses *spatial data* types in its functioning, meting out spatial indexing and proficient spatial query processing among others (Shekhar and Chawla, 2003). The onset of last decade has eagerly witnessed the launching and execution of mega *spatial data*bases and various techniques (Lin, 2010; Chong *et al*., 2003; Kim *et al*., 2002; Jagadish *et al*., 2005; Taniar and Rahayu, 2003) with a view to amass, scan, probe and reclaim spatial objects. A superb *spatial data*base is endowed with the qualities of sustaining and bringing together both explicit and implicit data of spatial objects. The explicit data of an object contains its locality, scope, direction, dimension and bounds. Implicit data, on the other hand, encompasses the spatial association between discrete objects, the allocation and density of objects in a specific area and the exposure for certain objects. The spatial objects are generally classified by Lin (2010) and Gaede and Gunther (1998) in d-dimensional Euclidean space. In heterogeneous *spatial data*base environs of modern Geographical Information Systems (GIS), data repositories collected and integrated from different spatial data origins more habitually coexist. Data integration surfaces as the vital subject to be faced-off (Cuzzocrea and Nucita, 2011; Butenuth *et al*., 2007) at present, which is extensively studies in the structure of *spatial data*bases (Cuzzocrea and Nucita, 2011; Calì *et al*., 2003; Ives *et al*., 1999). Conservative *spatial data*bases like those basically self-governing GIS, unrefined data files amassing geographical data and GIS-linked Web pages are outstanding examples of these sources. In diverse backdrops, things (or objects) in the Internet of Things (Tang *et al*., 2012; Ashton, 2009; Doytsher *et al*., 2012) can be designed as *MBR*s (Tang *et al*., 2012; Guttman, 1984) and spatial index and query have amazing emerged as the empowering techniques fruitful in the hunt for client-friendly objects.

In numerous reverences, the composition of *spatial data* bases is significantly divergent that of conventional databases. At the outset, the data model representing spatial objects has to be located well ahead of processing an index design. As a spatial object can take the shape of a specific point, a line segment, a curve, a polygonal segment, a 2D polygon or a

Corresponding Author: S. Palaniappan, Department of CSE, Saveetha University, Chennai, India

multidimensional polygon, it is highly essential to preserve its spatial data in a precise manner. Subsequently, insertions and deletions are broken up with revisions, as spatial objects are normally energetic (Lin, 2010). Spatial data is habitually represented by using exhaustive geometrical traits of *spatial data* base objects in a conservative *spatial data* base. This happens because geometrical one is the most inclusive depiction which can be presented on *spatial data* base objects. The input trait which shapes up a *spatial data*base as a powerful tool is its admirable acumen to have its magical sway on *spatial data*, instead of just being able to amass and represent them. The most essential form of such a system represents responding to spatial queries related to the spatial traits of data. Based specifically on their space occupancies, two-dimensional objects can be effectively categorized. Points with zero spatial occupancy are generally characterized by plain coordinates, which are aptly addressed and queried by several time-honored technologies. Polygons, circles, ellipses and rectangles get themselves parked in the domain of local data with nonzero spatial occupancy. And they are more often than not signified by rectangular objects, which are furthermore indexed and queried thus by various established methods. In view of the fact that the line segment is devoid of any area, it is impossible to be categorized as a non-zero-size object. However, it is observed that in quite a lot of previous investigations (Lin, 2010; Kollios *et al*., 1999; Papadopoulos *et al*., 2002; Yanagisawa *et al*., 2003), line segments have been integrated and represented by grids, cells, rectangles or *MBR*s (minimum bounding rectangles). It is pertinent to note that there has been a stream of innovative methods for handling point and region data over the past twenty year's duration.

Guttman (1984) and Lin (2010) gets the credit for shrewdly launching the innovative R-tree as the most outstanding configuration, for the purpose of indexing nonzero-size objects. An R-tree effectively executes *MBR*s to embrace nonzero-size objects, followed by their representation as indexed entities. In a gigantic pictorial database like the GIS application, R-trees find themselves widely exploited for the purpose of indexing the spatial objects (Lin, 2010; Papadias and Theodoridis, 1997). The indexing of *spatial data* is based on scope and the recuperation of data from the *spatial data* base by means of queries is an added coverage. In mega *spatial data* bases, spatial histograms are highly beneficial for reasonably precise query processing. The hassle of producing optimal spatial histograms is NP-hard; therefore, several heuristic-based methods have set their elegant foot during the course of the past one and a half decades. However, it is unfortunate that they are habitually haunted by hassles such as the intricate algorithmic design and sensitivity to constraint setting, which tend to place severe roadblocks of in the pathway of their zero-trouble integration into the authentic technologies. Taking cues from the R-tree index framework, many a K-NN query algorithm has been coined (Lai *et al*., 2002). Thus, an innovative branch and bound method is in the offing to move across R-tree, which is endowed with the efficiency of organizing and reducing the *MBR* in the interior of the nodes in R-tree to carry on K-nearest neighbors query (Liu *et al*., 2001). On the cards is a novel spatial k-NN query technique by employing MINDIST and MINMAXDIST intended to organize and prune rules so as to bring down tree. The more recent advent is the new-fangled Active Branch List (ABL) in leaf-level, viz. rect1, rect2; rectk, ably equipped with the nodes which have to be visited at the moment as well as the child nodes needed to extend the probe. In the ascending order of MINDIST and MINMAXDIST, each level ABL is organized in R-tree well-set to accept K-NN query (Liu *et al*., 2004). Another innovative launch is the multi-object K-NN query, which followed pruning rules to realize multi-object K-NN query in R-tree. On the other hand, R-tree paves the way for overlapping and exposure among the sister nodes, even in the case of the most precise match query, though it fails miserably to make a promise to visit only one branch while in possession of enquiries. In fact this is the most vital challenge having a telling impact on the probing prowess of R-tree.

In this study, we proposed a reduced *R*+ tree with KNN algorithm for indexing and retrieval of spatial data. The *R*+ tree is constructed initially based on the selected spatial data. Then a function called joining nodes is applied to reduce the number of nodes by combining the half-filled nodes to form completely filled. The idea behind reducing the nodes is to perform search and retrieval quickly and efficiently. The reduced *R*+ *tree* is then processed with KNN query algorithm to fetch the nearest neighbors to a point query. To take the closest neighbors to a point query, the diminished *R*+ *tree* is thereafter subjected to dispensation with the KNN query algorithm. The core processes of KNN algorithm are performed in the shining strategy for reclaiming the closest neighbors. With the help of *spatial data*, the milestone method is meticulously measured for its performance and the outperforming outputs are outlined in the investigational study section.

The vital donations of the technique are shown as follows:

- Investigated several *spatial data* indexing methods in significant query processing technologies
- Envisaged and designed a reduced *R*+ *tree* technique to characterized *spatial data*
- The KNN query algorithm is extensively employed to reclaim data from the reduced *R*+ *tree*
- Performed various feats and relative appraisal for the evaluation of the epoch-making techniques.

LITERATURE REVIEW

Here, we propose to delve deep into a debate on the modern investigations focused on the *spatial data* extraction together with a live discussions on several techniques employed for the purpose. The section furnishes a recount of the modern actions in the arena of *spatial data* extraction.

For processing of spatial queries in line-based database, an innovative technique for proficient and compact indexing configuration has effectively launched by Lin (2010). In this regard, points, lines and regions constitute the three vital entities for establishing vector-based objects in *spatial data* bases. Various indexing plans have been exceedingly discussed for dealing with point or region data. These conventional methods are powerful enough to organize point or region objects in a space into a hashing or hierarchical directory and hence furnish expert access approaches for accurate reclamations. On the other hand, while employing identical techniques, two different hassles surface as regards line segments as follows. In fact, the spatial data of line segments are not exactly explained with respect to that of points and/or regions. Moreover conventional approaches intended for addressing line segments tends to generate an incredibly large quantity of dead space and overlapping areas in internal and external nodes in the hierarchical directory. In the case of a line-based database, the former hassles puts insurmountable roadblocks in the pathway of super-quality spatial preservation of line segments as against the latter which tends to distort the system presentation over a period of time.

Cuzzocrea and Nucita (2011) have amazingly conceived and scientifically appraised an innovative technique which is functional for adeptly answering range queries over ongoing *spatial data* bases by duly blending geometrical data with topological logic. In addition, they brought to light me-SQE (Spatial Query Engine for Incomplete Information), inventive query machinery performing this technique. The eye-catching approach turns out to be both effectual and proficient in relation to synthetic as well as real-life *spatial data* sets. Further, in the long run, it empowers us to fine-tune the quality and the communicative authority of reclaimed answers by deriving remarkable advantages from the amenity of characterizing *spatial data* base objects throughout both the geometrical and the topological level.

Strongly rooted on PB-tree with the parallel lines division, a K-NN query algorithm has been theoretically generated by Tang *et al*. (2012). "PB-tree index is entirely different from the conventional R-tree index, where PB-tree performs parallel lines to segregate the spatial region and applies parallel lines as the parent node. It is associated with the binary tree index framework and is dire need of querying three trivial segments contiguous to the queried object in each K-NN query. With the result, the search range is watered down and the query aptitude is fine-tuned. Related tests conducted vouchsafe the fact that PB-tree exhibits superior robustness vis-à-vis the conventional R-tree from the perspective of query presentation. PB tree is capable of avoiding the inadequacy of a mammoth size of overlap and exposure among nodes in R-tree and multiple index paths while on the lookout for data objects. Consequently, the PB-tree is competent to spot K-NN objects by satisfying the constraints swiftly and adeptly in mega databases" (Tang *et al*., 2012).

Corral and Almedros-Jimenez (2007) "have astoundingly conceived a revise on R-trees, which resulted in multifarious inferences on the competency of advocated RBFS algorithm and its contrast in relation to parallel probe approaches such as Best-First Search (BFS) and Depth-First Branch-and-Bound (DFBnB)), with respect to disk accesses, feedback duration time and vital memory stipulations, considering several pertinent parameters as maximum branching factor (Cmax), cardinality of the final query result (K), distance threshold (q) and size of a global LRU buffer (B). As a rule, RBFS is aggressive for KNNQ and KCPQ where the maximum branching factor (Cmax) is huge enough and in some cases even better than DFBnB and very near BFS. Moreover, it emerges as a better option when the main memory faces constraints in our computer because of elated process surplus in our system, as it is linear space consuming with respect to the height of the R-trees. Nevertheless, RBFS is the most awful alternative for qDRQ and qDJQ. DFBnB is additionally a linear space algorithm and exhibits behavior identical to BFS for qDRQ and qDJQ and it sails to the summit when an LRU buffer is attached" (Corral and Almedros-Jimenez, 2007). In the long run, we are delighted to find that it is proved by test outcomes that BFS emerges par-excellence among the entire DBQs, though it is competent to overwhelm several main memory resources to efficiently perform spatial queries.

Achakeev and Seeger (2012) devised an amazing class of spatial histograms gathered from the well-regarded family of R-tree indexes. They brought to spotlight a cost-conscious approach that blends the bulk-loading of R-trees and composition of spatial histograms. Endowed with efficient exactness for selectivity evaluation of spatial queries, this conceives an energetic histogram method. Especially, the appraisal deficiency persistently takes a backseat with rocking number of histogram buckets, thus our historic histogram approach tastes gains from a huge number of histogram buckets. For the purpose of test appraisal, they analyzed and contrasted the charismatic feat of our method with state-of-the-art spatial histograms. In contrast to prior-performed tests, they subjected the feat to diverse clusters of workloads.

Motivation behind the approach: A *spatial data* can be described as a set of data "that identifies the geographic location of features and boundaries on Earth, such as natural or constructed features, oceans and more. *Spatial data* is usually stored as coordinates and topology and is data that can be mapped. *Spatial data* is often accessed, manipulated or analyzed through Geographic Information Systems" (GeoMAPP, 2010). The main concentration of the researchers is to find an efficient way for indexing the *spatial data* for efficient retrieving according to the need of situations. The algorithms like, R-tree, *R+ tree*, B tree, etc are the commonly used indexing algorithms for *spatial data*. The need for indexing is that, the indexed *spatial data* can be easily retrieved from the databases using query processing algorithms. Lin (2010) has recently proposed an approach for processing *spatial data* through efficient in line based queries. The approach was concentrating on B+ tree in a compressed manner, which efficient in time and space complexity. Inspired from the research, we have intended to propose a method for *spatial data* representation using reduced *R+ tree*. In the proposed approach, the reduced *R+ tree* is used extract nearest neighbors using KNN query algorithm. The reduced *R+ tree* is used to make the searching process and retrieval process efficient and quick.

Spatial data representation: A *spatial data* base, in essence, is a unique database which is optimized to amass and enquire data which characterizes objects demarcated in a geometric space. A major chunk of *spatial data* bases entails characterization of plain geometric objects like points, lines and polygons. Still, certain s*patial data* bases deal with further intricate configurations like 3D objects, topological coverage, linear networks and TINs. Whereas characteristic databases are planned to organize diverse numeric and character types of data, supplementary functionality has to be ensured for databases to tackle *spatial data* types professionally, which are normally known as geometry or feature. The Open Geospatial Consortium defines the Simple Features specification and sets benchmarks for complementing spatial functionality to database systems. Taking into account the zooming significance of *spatial data*, there surfaces the critical requirement for techniques to proficiently organize and interrelate with the *spatial data*. It is high time a depiction model is put in place for the *spatial data* to successfully administer the same. The sterling method is all set to design an approach to characterize the *spatial data*. The most popular techniques doing the rounds for the characterization of the *spatial data* are known by the names, R *tree*, *R+ tree*, B+ tree. Though generally employed for representing the *spatial data*, they are prone to certain glaring deficiencies described as follows. *R tree* algorithm faces the music in regard to probing challenges during the incidence of the over lapping of rectangle. As a corrective measure, the *R+ tree* is shown the limelight to overshoot the thorny menace, though it leads to bottlenecks in the node administration. Further, B+ tree is equipped with the acumen of effectually organizing and successfully addressing the dilemma of the rectangle over lapping. Thus, it goes without saying that there is an ever-zooming necessity for ushering in a proficient technique for addressing the issue of the *spatial data* illustration.

Proposed approach for nearest neighbor retrieval from reduced *R+ tree*: Recently, "*spatial data* bases have been extensively adopted in the recent decade and various methods have been presented to store, browse, search and retrieve spatial objects. A *spatial data*" (Lin, 2010) can be described as a set of "data that identifies the geographic location of features and boundaries on Earth, such as natural or constructed features, oceans and more. *Spatial data* is usually stored as coordinates and topology and is data that can be mapped. *Spatial data* is often accessed, manipulated or analyzed through Geographic Information Systems "(GeoMAPP, 2010). The main concentration of the researchers is to find an efficient way for indexing the *spatial data* for efficient retrieving according to the need of situations. The algorithms like, R-tree, R+tree, B tree, etc are the commonly used indexing algorithms for *spatial data*. The need for indexing is that, the indexed *spatial data* can be easily retrieved from the databases using query processing algorithms. Lin (2010) has recently proposed an approach for processing *spatial data* through efficient in line based queries. The approach was concentrating on B+ tree in a reduced manner, which efficient in time and space complexity. Inspired from the research, I have intended to propose a method for processing *spatial data*. The proposed approach uses two steps for extraction relational information from the *spatial data*:

- *Spatial data* representation with reduced *R+ tree*
- Information extraction with KNN Algorithm

In the first step, the *spatial data* is accepted as the input and represented in the form of *R+ tree*. Here, we use a reduced *R+ tree* to reduce the number of nodes, which result in efficient node search and information retrieval. In next step, that is the information retrieval phase, the KNN algorithm extracts relevant information from the *spatial data* with the help of reduced *R+ tree*. The detailed explanation of the processes is plotted in the coming sections.

The reduced *R+* tree for spatial data representation: The issue of depiction of *spatial data* is one of the vital challenges of the innovative technique. The cardinal objective of the ambitious approach is invariably targeted in representing the *spatial data* in *R+ tree*. The

relevant task has to be completed without losing significant data and by bringing down the number of nodes. The decrease in the number of nodes has to be integrated for the swift reclamation of data. The *R+ tree* employed in the ground-breaking technique is an innovative, reduced *R+ tree*, where all our attention is focused on cutting down the number of nodes by empowering a node to allow its maximum occupancy. The building of the reduced *R+ tree* is exceedingly akin to the creation of *R+ tree*s. In effect, though the *R+ tree* is an offshoot of *R tree,* it follows a dissimilar process in probing and insertion of nodes, which makes it vitally divergent from the *R tree*. Moreover, the big-bang *R+ tree* envisages a unique process termed partitioning, wherein the hassles posed by overlapping in the case of R *tree* is effectively kept at bay.

Figure 1 represents a sample of *R+ tree*, which possess bounding rectangles and partitions. Here A, B and C are considered as the bounding rectangles or minimum bounding rectangles. The partition of the *R+ tree* is represented using rectangle P. The minimum bounding rectangle or *MBR* of *R+ tree* is defined as the Rectangle that possess the minimum number of data elements in it. The reduced *R+ tree* tries to minimize the *MBR* and defines every bounding rectangle to be built with maximum number of data points. The reduced *R+ tree* give more concentration on the process like searching and insertion. Consider Fig. 2.

Figure 2 characterizes the *R+ tree* generated for Fig. 1. It illustrates that, the ultimate tree houses a number of half-filled nodes. Therefore, when a probe is intended to the *R+ tree*, it tends to consume further e time to bring in data because the pointer has to be routed along all the nodes. With a view to effective address the menace, we set out to execute a unique compression method, which cuts back the number of

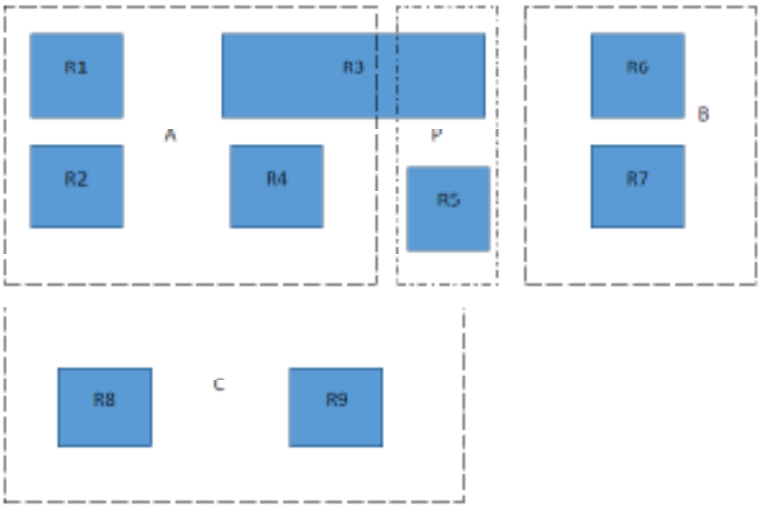

Fig. 1: Rectangle representation of the *R+ tree*

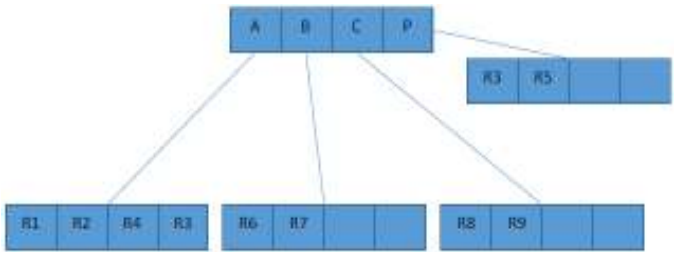

Fig. 2: *R+ tree* for the Fig. 1

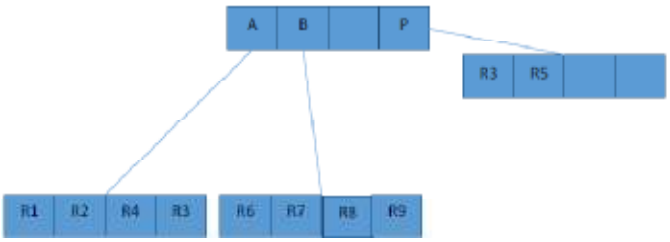

Fig. 3: Reduced *R+ tree*

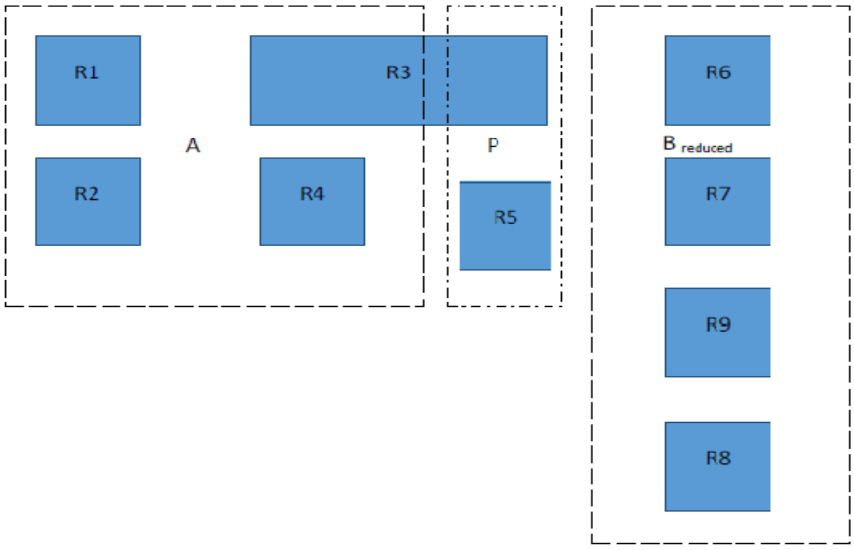

Fig. 4: Rectangle represented from the reduced *R+ tree*

nodes by directing the address of consecutive data objects to the previous empty spaces of the prior nodes. With the result, the above-discussed *R+ tree* takes a new shape as Fig. 3.

Figure 3 shows the reduced *R+ tree* generated for the represented Fig. 2. Through the compression, the nodes in rectangles B and C are reduced to form a single bounding rectangle $B_{reduced}$.

Figure 4 represents the minimum bounding rectangles of the reduced *R+ tree*. This picture reveals that, there is a limited number of *MBR* to search for and it is easy to point any data through search as compared to the Fig. 1. Now, move on to the process of constructing the reduced *R+ tree*. Similar to the othe*R trees* used for *spatial data* representation, the *R+ tree* also includes five major operations:

- Search
- Insertion
- Deletion
- Splitting the nodes
- Joining the nodes

The following section discusses the four process in detail.

Searching data in reduced *R+ tree:* The probing algorithm is analogus to the technique employed in *R+ trees*. The motive underlying the probe is to initially decay the search space into disjoint sub-regions and for each and every one of these, move down the tree until the authentic data objects are located in the leaves. As the nodes are decreased in reduced *R+ tree*, the probe tends to be cost-effective with respect to the *R+ trees*. The search in reduced *R+ tree* is shown below in Fig. 5.

Insertion on reduced *R+ tree*: Inserting a new rectangle in an R+-tree is carried out by probing the tree and supplementing the rectangle in leaf nodes. The divergence with the analogous algorithm for R-trees is that the input rectangle may be joined to more than one leaf node, in view of the fact that it may be divided to sub-rectangles along current partitions of the space. In the long run, brimming nodes are divided and splits are spread to parent as well as children nodes. The latter has to be kept updated, as a split to a parent node may bring in a space partition that influences the children nodes also. The code for incertion is exhibited in Fig. 6 given below:

Deletion: Deletion of a rectangle from an *R+-tree* is performed in the same way as in R-trees by initially spotting the rectangle (s) to be subjected to the process of deletion and thereafter steering clear of it (them) from the leaf nodes. The underlying motive for dispensation of multiple rectangles from leaf nodes is that the insertion schedule envisaged earlier is likely to bring in multiple copies for a newly inserted rectangle. In the case of diminished *R+ tree* deletion is drawn in the event of incidence of immaterial entry in the generated *R+ tree*.

Splitting the nodes: In case a node overflows, there is a necessity for certain splitting algorithm so as to generate two new nodes. And it is highly essential that the two sub-nodes cover up reciprocally disjoint domains, we have to initially probe for a "good" partition (vertical or horizontal) which eventually decays the space into two sub-regions. This process is generally known as Partition, which invariably issues a clarion call for downward propagation of the split. Let us for instance, consider in Fig. 2. We assume that A is a parent node of *B* which, conversely, is a parent node of *C*. Therefore, if node *A* has to be divided, it is necessary that the lower level nodes *B* and *C* are also

Algorithm search:
Psuedocode for search in reduced *R+ tree*
Input: R+ tree
Output: All data objects including overlapped
Step 1: Select the top Node R, from R + tree
Step 2: If node N is not leaf, then
For all (*P*, *R*) of *top node R* check if next node overlaps *W,* search window.
Step 3: If so, Initiate Search (*C, W, N*), where *C* is the node that user pointed.
Step 4: If *N* is a leaf, check all objects *R* in *O* and return those that overlap with *W*.

Fig. 5: Psuedocode for search in reduced *R+ tree*

Algorithm insert:
Insertion psuedocode
Input: *R+ tree with input rectangle (IR)*
Output: *The new R+ tree with inserted point*
Step 1: Select the node where IR goes and add
Step 2: If node N is not a leaf, then for each entry (p, R) of N,
Search for any overlapping
If overlap then, Insert (C, IR), C is the child node
Step 3: If N a leaf, add IR in R. If the entries into the rectangle is more the limit M, the initiate Split Node () Function

Fig. 6: Insertion psuedocode

Algorithm Split Node (N):
Input: A node *N*
Output: The new R+tree
Step 1: Find a partition and create two new nodes
Step 2: Find Partition *on Rectangle X* using the search method. *P,* be the pointer associated with node *R*
Step 3: Create *n*1 = (*p*1, *X*1) and *n*2 = (*p*2, X2), the two nodes resulting from the split of R, Where X_1 and X_2 are new sub rectangles
Step 4: Put in *ni* all nodes (*pk*, *Rk*) of *O* such that *Rk*lies completely in *Ri*, for *i* = 1, 2. For those nodes that *Rk, Ri*
a) if*O* is a leaf node, then put *Rk*in both new nodes
b) Otherwise, use Split Node to recursively split the children nodes along the partition. Let (*Pk*1, *Rk*1) and (*Pk*2, *Rk* 2) be the two nodes after splitting (*Pk*, *Rk*), where *Rki* lies completely in *Ri*, *i* = 1, 2. Add those two nodes to the corresponding node *ni*.

Fig. 7: Pseudo code for node splitting

Algorithm joining nodes (R-reduced):
Input: *R+ tree* with half-filled nodes
Output: Reduced *R+ tree*
Step 1: Let maxOccu *MBR* = n
Step 2: Check each *MBR* for maxOccu*MBR*
Step 3: For any *MBR* if (occu*MBR*<n)
Select those *MBR*s;
Step4: If (MBR_I&& MBR_J) has Less occupancy than maxOccu*MBR*
Find distance (MBR_I, MBR_J)
If distance == Minimum
Check,
occu*MBR*(MBR_I)&&occu*MBR*(MBR_J)
Then,
Add point p from MBR_I to MBR_J
Step 5: Add points from MBR_I to MBR_J until,
MBR_I == Empty or MBR_J == fully filled
Step 6: Repeat 3 to 5 untill maximum nodes are fully filled
Step 7: End

Fig. 8: Psuedocode for joining node

subject to the process of division. The pseudopodia meant for node division is beautifully carved out in Fig. 7 shown above:

Joining the nodes: The process of joining nodes trigger after the *R+ tree* is built completely. The procedure will search for half-filled and full filled nodes from the *R+ tree*. The half-filled nodes are selected and their empty spaces are accounted. The nearest *MBR* are selected and if both are half filled, we subject a joining function that will add the nodes to one *MBR* to another until an *MBR* gets completely filled or empty. In details, if *MBR*1 possess two data points and *MBR*2 possess 2 data points. The condition is that the maximum data points possessed by an *MBR* is 4. If *MBR*1 and *MBR*2 are near, then data points from *MBR*2 is added to *MBR*1.Finally *MBR*1 gets completely filled and *MBR*2 gets empty. So a *MBR* is reduced from the total structure, this helps in easy data search and retrieval. If *MBR*2 has three data points, the rest is added to another near *MBR*, proved the nearest *MBR* is half filled.

The above Fig. 8 represents the psuedocode for joining the nodes according to the *MBR* present. Initially, we set the maximum occupancy of each *MBR* as "n". Then each *MBR* present is checked for maximum occupancy and the *MBR*s with occupancy less than n is selected for the joining node process. Select MBR_I and MBR_J, which has less occupancy than maximum. We subject a distance calculation between the MBR_I and MBR_J and if the distance is minimum, the points from MBR_I is passed to MBR_J. Assuming that MBR_J has more vacant space than MBR_I. The process continues till maximum nodes are completely filled.

Region queries handling: In the previous section, we are discussing about the reduced *R+ tree* and the application of algorithm to index *spatial data* and retrieve data accordingly. So far the discussions are through a point query based manner are just retrieving the data based on a single value query. The main challenge arises when a range of queries are subjected to the system for retrieving the data. In this section we discuss about how to handle range queries using the proposed system.

***MBR* selection:** The initial step in the process to map the data based on the *MBR*s. As per the proposed approach, we defined the Min_dist and Max-Min_dist of the *MBR* with respect to a point in the problem space. If the query is a single point, the Min_dist and Max_min_dist can be easily calculated. When a range of query is subjected, we have to represent the range as a single value point in order to ease the calculation as per the proposed approach. The idea is to find the centroid of the range queries. So if there is a 100 queries are there in the input range query, we cannot individually select each point and find its index from

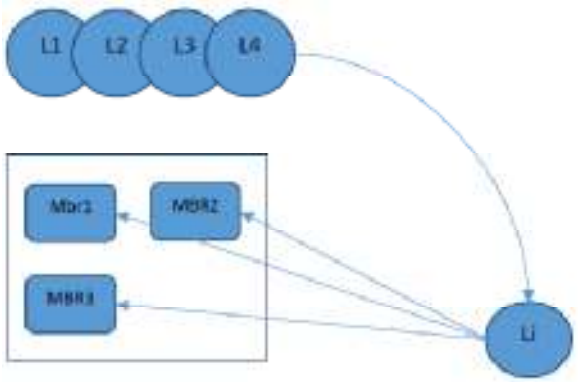

Fig. 9: Point to *MBR* matching

our system. Thus a clustering algorithm is applied to range values; simple k-means algorithm is the algorithm we used, to represent the whole query as a centroid. We initially, generate 3 clusters for every set of queries and will subject three centroid values. Now, we select the each centroid and calculate the Min_Dist and Max_min_distance with respect to each *MBR*. The *MBR* that possess least value for both will be selected for the further process. The KNN query algorithm accepts the centroid values as the input and proceeds to the processing.

Figure 9 represents the process of matching each point in the region based query in the desired Tree. Each point L_i in the region query is separately selected and their distance to the *MBR* is calculated using the MinDistAndMaxDist calculation. The *MBR* with minimum distance for both MinDist and MaxMinDist is selected and the region is assigned to the particular *MBR*. The detailed process can be explained with help of point query basis. The second step of the proposed approach deals with query processing through KNN algorithm. We test the program by giving points query and process it with KNN algorithm. The idea behind the approach is to extract nearest neighbors from the data set corresponding to the given query. Since the *spatial data* concerns with information regarding geographical co-ordinates. The nearest neighbors constitute similar characteristics. The proposed approach uses the KNN algorithm for the process of extracting nearest neighbors from the *spatial data*base. The KNN algorithm uses the reduced *R+ tree* for extracting the nearest neighbors. The problem can be defined as, a point query is subjected to the dataset and we need to identify the K nearest neighbors to the given query. Here, the K values is given from the user side and it is the number neighbors need to be extracted. The main object that KNN concentrate is the *MBR* of the reduced *R+ tree*. Consider the following Fig. 10.

Figure 10 represents the scenario of point query and the *MBR* in the problem space. Now in order to extract the nearest neighbors of the point P(x,y), the KNN algorithm is applied. The KNN defines two parameters for finding the nearest neighbors from the *MBR*.

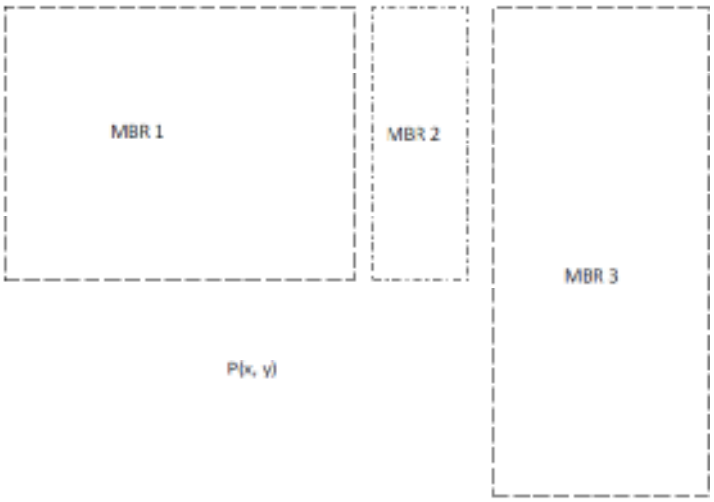

Fig. 10: *MBR* and point query

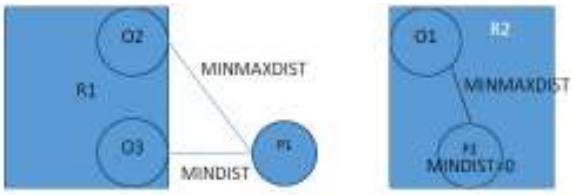

Fig. 11: MINDIST and MINMAXDIST

MINDIST: The parameters MINDIST is defined the minimum distance between the point query P(x, y) and an *MBR* R. If the point that is subjected as the query is inside a Rectangle R then, the minimum distance value between the point and rectangle will be zero. When the point is outside the rectangle R, then the MINDIST value will calculated as the distance between the point P and any data point in the perimeter of the rectangle R. i.e., MINDIST = distance (P, O).

Here, distance () is the function used for calculating the distance between point P and data point in the rectangle R. The distance function used for the purpose will be the Euclidian distance.

MINMAXDIST: The parameter MINMAXDIST is defined as the distance between the point P and object furthest and closest phase of the rectangle R. MINMAXDIST is the smallest possible upper bound of distances from the point P to the rectangle R. MINMAXDIST guarantees there is an object within the R at a distance to P less than or equal to it. The MINDIST and MINMAXDIST can be explained with the help of Fig. 11.

Figure 11 represents the MINDIST and MINMAXDIST with respect to two points P1 and P2. The point P1 is outside the R1. So the distance between P1 and data point in R1, O3 is considered as the MINDIST between P1 and R1. The distance between P1 and O2 is considered as the MINMAXDIST. The case of P2 is different, because P2 is inside the rectangle R2, so the MINDIST is zero and there is only MINMAXDIST is taken into account. Note that, the KNN will process only one point query at a time. When a point query is subjected to the *R+ tree*, all the MINDIST and MINMAXDIST will be calculated with respect to the input query point. The KNN uses pruning conditions to retrieve the nearest neighbors efficiently:

- An *MBR* R is discarded if there exists another R' such that MINDIST (P, R) >MINMAXDIST(P, R')
- An object O is discarded if there exists an R such that ACTUALDIST (P, O) >MINIMAXDIST(P, R)
- An *MBR* R is discarded if an object O is found such that MINDIST (P, R) >ACTUAL_DIST (P, O)

As per these pruning conditions, the relevant rectangles are sustained and rest of the rectangles are discarded. The nodes present in the sustained rectangles are selected and the nearest neighbors are selected from it according to the K value. The data points, which is least distinct to the point query is selected and rest are discarded:

$$NN(P) = Distance(P, O_i) \rightarrow Minimum$$

Here, NN (P) represents the nearest neighbors of P, O_i represents the data points in the sustained rectangles. The value of "i" varies from 1 to n, n is the total number of data points available. Similar to the above explanation of point query, each point in the region query follows these steps and they it will be assigned to *MBR*, which possess average minimum distance. Now with the help of *MBR*, we construct the reduced *R+ tree* so as to minimize the number of nodes. The process executed in the selection of *MBR* helps the proposed approach to minimize the number of nodes because a

region in the *spatial data* is plotted to the nearest *MBR* without overflow. The *MBR* is selected in such way that, the process check whether the *MBR* is fully filled or partially filled. The *MBR* is selected only if it is partially filled. The partially filled *MBR* is then selected and populated with the points in the region query and if the *MBR* exceeds the capacity and data points in the region query is left behind. Then another nearest *MBR* is selected and it is populated with the rest of the data.

EXPERIMENTAL RESULTS AND ANALYSIS

The working and detailing of the proposed approach is plotted in the prior section. In this section, we discuss about the experimental analysis of the proposed approach. A *spatial data*set is given to the proposed approach and the responses of the proposed approach is recorded to evaluate the performance.

Experimental set up: The proposed approach is programmed in java program with JDK 1.7.0. The system used for developing the program uses anIntel core i5 processor, 500GB hard disk and 3GB of RAM. The dataset used for the proposed approach is a *spatial data* set with more than 1000 data points. The datasets are classified into 5 groups and tested with the proposed approach. The dataset is divided into groups like 200, 400, 600, 800 and 1000. The dataset is supplied to both *R+ tree* and reduced *R+ tree* for evaluation purposes. The detailed evaluation of the proposed approach is plotted in the following section. We are using two dataset from the *spatial data* repository (Spatialkey, 2012) and processed the database as explained above.

Performance analysis: In this section, we plot the performance of the proposed *R+ tree* algorithm aided with KNN for extracting the neighbors. The performance of the proposed approach is processed in terms of the parameters like computation time, retrieval time and node generated. The two dataset downloaded are selected for the evaluation purposes. The computation time represents the time required to run the whole process. The retrieval time deals with time required to retrieve the nearest neighbors through the give query based on proposed algorithm. The performance is evaluated as a comparison to the existing *R+ tree* based method.

Performance based on computation time: Initial the datasets are labeled as dataset 1 and dataset 2. There datasets are supplied to the *R+ tree* based method and reduced *R+ tree* based method. The total time required to represent the total data points into the *R+ tree* and reduced *R+ tree* is plotted below. The both datasets are reduced to 1000 data point and are divided into five groups as mentioned above. Here, the computation time obtained for five different data records for both the proposed reduced *R+ tree* and the existing *R+ tree* is tabulated above in Table 1 for dataset 1 and in Table 2 for dataset 2.

Table 1: Computation time dataset 1

Data records	*R+* tree	Reduced *R+* tree
200	1087	854
400	1158	890
600	1378	901
800	1388	910
1000	1476	914

Table 2: Computation time dataset 2

Data records	*R+* tree	Reduced *R+* tree
200	13724	11241
400	12475	11800
600	13547	12001
800	14251	12009
1000	15374	12900

Table 3: Retrieval time from dataset 1

Data records	*R+* tree	Reduced *R+* tree
200	220	180
400	325	195
600	360	202
800	496	256
1000	524	302

Table 4: Retrieval time from dataset 2

Data records	*R+* tree	Reduced *R+* tree
200	620	540
400	725	590
600	788	620
800	801	645
1000	901	711

Figure 12 and 13 represents the performance of the proposed approach in terms of dataset 1 and dataset 2. The analysis from the fig shows that at every level the proposed reduced *R+ tree* algorithm has the upper hand in computation time. This implicate that the time required for indexing the data points in reduced *R+ tree* is less than that of the traditional *R+ tree* method. The reduced *R+ tree* has consumed indexing time for the peak level of data is only 914 ms for dataset 1 and 12900 ms for dataset 2. Even though the numbers of data points are same there is huge difference between the computation times of the datasets. The reason behind that is the data, which is possessed by both datasets. The analysis from the figs shows that as the number of data increases, the computation time increases in both cases. The difference time represents the effectiveness of the proposed approach over traditional *R+ tree*.

Performance based on retrieval time: The retrieval time deals with the time required to retrieve nearest neighbors by the KNN algorithm from the indexed tree upon giving the input query. The same datasets are used in this process also, so we can have a clear record on the performance of the proposed approach. For the both dataset 1 and dataset 2 the retrieval obtained for the proposed reduces *R+ tree* and existing *R+ tree* for five different data records are tabulated in Table 3 and 4.

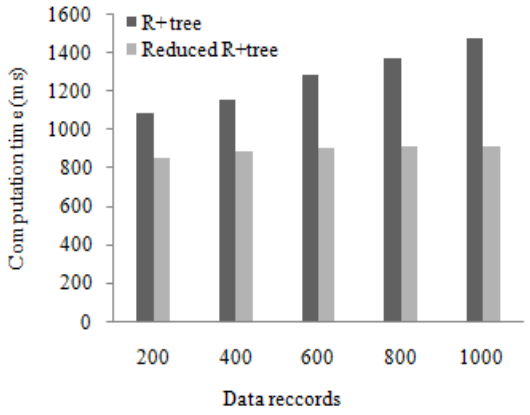

Fig. 12: Computation time of dataset 1

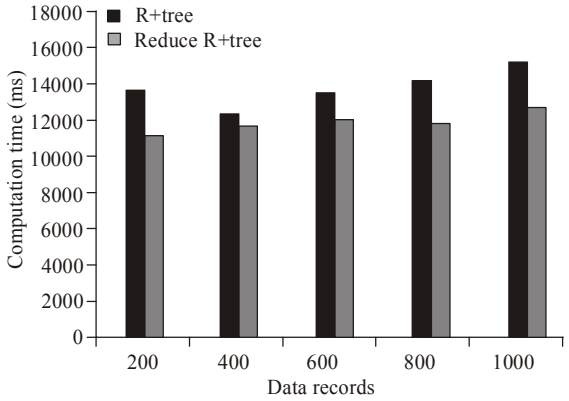

Fig. 13: Computation time for dataset 2

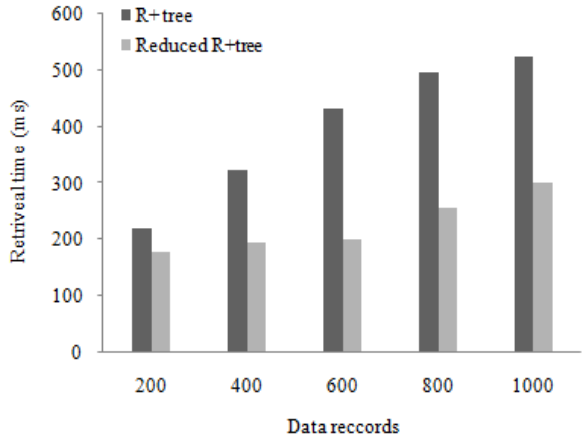

Fig. 14: Retrieval time from dataset 1

Figure 14 and 15 represents the retrieval time consumed by the proposed *R+ tree* and conventional *R+ tree*. The KNN algorithm is applied after indexing the data with conventional *R+ tree* and reduced *R+ tree*. So the responses are plotted in the above depicted figs. The analysis from the figs shows that retrieval time is considerable low for the reduced *R+ tree* based method as compared to conventional *R+ tree* based

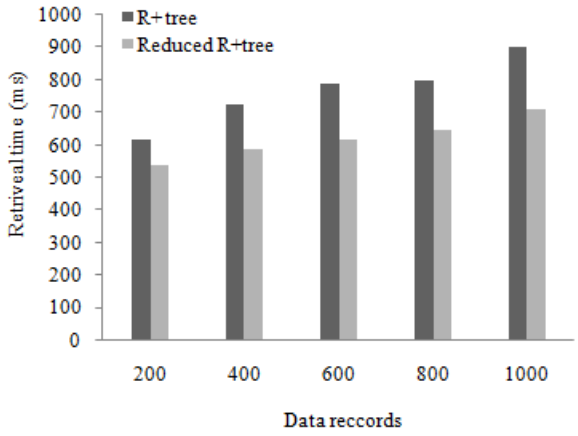

Fig. 15: Retrieval time from dataset 2

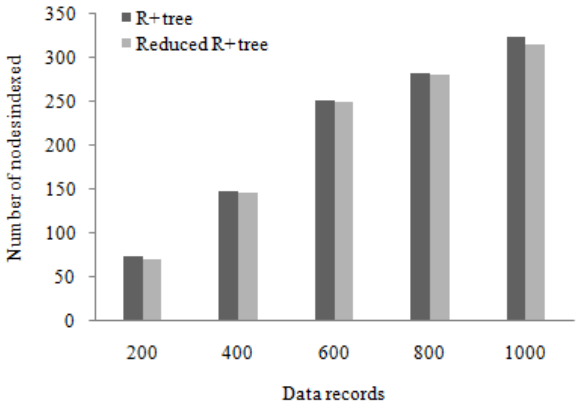

Fig. 16: Number of nodes

Table 5: Number of nodes

Data records	*R+* tree	Reduced *R+* tree
200	72	70
400	148	145
600	251	250
800	282	280
1000	324	316

method. The proposed approach has achieved an average retrieval time of 227 ms for dataset 2 and 621.2 ms for dataset 1, while the conventional *R+ tree* based method achieved about 767 ms for dataset 2 and 400 ms for dataset 1.

Number of nodes: The number of nodes obtained for both the proposed reduced *R+* tree and existing *R+* tree for different data records are tabulated in Table 5.

Figure 16 represents the number of nodes possessed by *R+ tree* and Reduced *R+ tree* after indexing the data points. As per the name states, the proposed approach has less number of nodes with respect to the conventional *R+ tree*. We can see a remarkable difference in the number of nodes between the two algorithms. The conventional *R+ tree* requires about 324 nodes for representing 1000 data points while the

reduced *R+ tree* takes about 316 nodes. The analysis can be accounted that the proposed approach also index similar number of nodes as compared to the *R+ tree*. The advantage of the proposed approach is terms of retrieval time and computational time. The process is conduced setting a limit of 4 data points as the maximum occupancy of a node. The remarkable difference number of nodes possessed by reduced *R+ tree* is because of eliminating the empty spaces in the nodes. The depicted image uses the dataset 2 for the purpose of counting number of nodes.

CONCLUSION

In this study, a method is plotted for retrieving nearest neighbors from *spatial data* indexed by *R+ tree*. The approach uses a reduced *R+ tree* for the purpose of representing the *spatial data*. Initially the *spatial data* is selected and *R+ tree* is constructed accordingly. Then a function called joining nodes is applied to reduce the number of nodes by combining the half-filled nodes to form completely filled. The idea behind reducing the nodes is to perform search and retrieval quickly and efficiently. The reduced *R+ tree* is then processed with KNN query algorithm to fetch the nearest neighbors to a point query. The basic procedures of KNN algorithm are used in the proposed approach for retrieving the nearest neighbors. The proposed approach is evaluated for its performance with *spatial data* and results are plotted in the experimental analysis section. The experiments showed that the number of nodes possessed by reduced *R+ tree* is remarkably lower than that of the conventional *R+ tree*. The proposed approach is efficient in terms computation time and retrieval time also.

REFERENCES

Achakeev, D. and B. Seeger, 2012. A class of R-tree histograms for *spatial data*bases. Proceeding of the 20th International Conference on Advances in Geographic Information Systems (SIGSPATIAL '12), pp: 450-453.

Ashton, K., 2009. That 'Internet of Things' thing. RFID J., 22(2009): 97-114.

Butenuth, M., G. Gsseln, M. Tiedge, C. Heipke, U. Lipeck and M. Sester, 2007. Integration of heterogeneous *Spatial data* in a federated database. ISPRS J. Photogramm., 62(5): 328-346.

Calì, A., D. Lembo and R. Rosati, 2003. Query rewriting and answering under constraints in data integration systems. Proceeding of the International Joint Conference on Artificial Intelligence, pp: 16-21.

Chong, E.I., J. Srinivasan, S. Das, C. Freiwald, A. Yalamanchi, M. Jagannath, A.T. Tran, R. Krishnan and R. Jiang, 2003. A mapping mechanism to support bitmap index and other auxiliary structures on tables stored as primary B^+-trees. SIGMOD Rec., 32(2): 78-88.

Corral, A. and J. Almedros-Jimenez, 2007. A performance comparison of distance-based query algorithms using R-trees in spatial databases. Inform. Sciences, 177(11): 2207-2237.

Cuzzocrea, A. and A. Nucita, 2011. Enhancing accuracy and expressive power of range query answers over incomplete spatial databases via a novel reasoning approach. Data Knowl. Eng., 70(8): 702-716.

Doytsher, Y., B. Galon and Y. Kanza, 2012. Querying socio-spatial networks on the world-wide web. Proceeding of the 21st International World Wide Web Conference (WWW). Lyon, France, pp: 329-332.

Gaede, V. and O. Gunther, 1998. Multidimensional access methods. ACM Comput. Surv., 30(2): 170-231.

GEOMAPP, 2010. Retrieved from: www.geomapp.net.

Guttman, A., 1984. R-trees: A dynamic index structure for spatial searching. Proceeding of the ACM SIGMOD International Conference on Management of Data. Boston, Massachusetts, pp: 47-57.

Ives, Z.G., D. Florescu, M. Friedman, A. Levy and D.S. Weld, 1999. An adaptive query execution system for data integration. Proceeding of the ACM International Conference on Management of Data, pp: 299-310.

Jagadish, H.V., B.C. Ooi, K.L. Tan, C. Yu and R. Zhang, 2005. iDistance: An adaptive B^+-tree based indexing method for nearest neighbor search. ACM T. Database Syst., 30(2): 364-397.

Kim, J.H., Y.H. Kim, S.W. Kim and S.H. Ok, 2002. An efficient processing of queries with joins and aggregate functions in data warehousing environment. Proceeding of the 13th International Workshop on Database and Expert Systems Applications. Aix-en-Provence, France, pp: 785-794.

Kollios, G., D. Gunopulos and V.J. Tsotras, 1999. On indexing mobile objects. Proceeding of the 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'99), pp: 261-272.

Lai, L., Z. Liu and L. Yan, 2002. K-nearest neighbor search algorithm by using R-tree. Comput. Eng. Des., 23(9).

Lin, H.Y., 2010. Efficient and compact indexing structure for processing of spatial queries in line-based databases. Data Knowl. Eng., 64(1): 365-380.

Liu, Y., Z. Zhu and S. Shi, 2001. A new space k-nearest neighbor query strategy. J. Shanghai Jiao Tong Univ., 35(9).

Liu, Y., S. Bo, Q. Zhang and Z. Hao, 2004. Multi-object nearest neighbor queries. Comput. Eng., 30(11): 66-68.

Papadias, D. and Y. Theodoridis, 1997. Spatial relations, minimum bounding rectangles and spatial data structures. Int. J. Geogr. Inf. Sci., 11(2): 111-138.

Papadopoulos, D., G. Kollios, D. Gunopulos and V.J. Tsotras, 2002. Indexing mobile objects on the plane. Proceedings of the 13th International Workshop on Database and Expert Systems Applications (DESA'02), pp: 693-697.

Shekhar, S. and S. Chawla, 2003. Spatial Databases: A Tour. Prentice-Hall, New Jersey.

Spatialkey, 2012. Retrieved from: http://support.spatialkey.com/spatialkey-sample-csv-data/.

Tang, J., Z.B. Zhou and Q. Wang, 2012. K-NN query algorithm based on PB-tree with the parallel lines division. Commun. Mobile Comput., 1(1): 1-10.

Taniar, D. and J.W. Rahayu, 2003. Global B^+-tree indexing in parallel database systems. Lect. Notes Comput. Sc., 2690: 701-708.

Yanagisawa, Y., J. Akahani and T. Satoh, 2003. Shape-based similarity query for trajectory of mobile objects. Proceeding of the 4th International Conference on Mobile Data Management. Melbourne, Australia, pp: 63-77.