AN EXPERIMENTAL ANALYSIS OF SPATIAL INDEXING ALGORITHMS FOR REAL TIME SAFETY CRITICAL MAP APPLICATION

F. Çetin[a,b,*], M. O. Kulekci[a],

[a] Graduate School, Istanbul Technical University, Turkey - (cetinf17, kulekci)@itu.edu.tr
[b] ASELSAN A.S., Turkey fcetin@aselsan.com.tr

Commission IV, WG IV/3

KEY WORDS: Spatial data processing, Space partitioning, KD-Tree, Quad-KD-Tree, PR-Tree

ABSTRACT:

This paper presents a study that compares the three space partitioning and spatial indexing techniques, KD Tree, Quad KD Tree, and PR Tree. KD Tree is a data structure proposed by Bentley (Bentley and Friedman, 1979) that aims to cluster objects according to their spatial location. Quad KD Tree is a data structure proposed by Berezcky (Bereczky et al., 2014) that aims to partition objects using heuristic methods. Unlike Bereczky's partitioning technique, a new partitioning technique is presented based on dividing objects according to space-driven, in the context of this study. PR Tree is a data structure proposed by Arge (Arge et al., 2008) that is an asymptotically optimal R-Tree variant, enables data-driven segmentation. This study mainly aimed to search and render big spatial data in real-time safety-critical avionics navigation map application. Such a real-time system needs to efficiently reach the required records inside a specific boundary. Performing range query during the runtime (such as finding the closest neighbors) is extremely important in performance. The most crucial purpose of these data structures is to reduce the number of comparisons to solve the range searching problem. With this study, the algorithms' data structures are created and indexed, and worst-case analyses are made to cover the whole area to measure the range search performance. Also, these techniques' performance is benchmarked according to elapsed time and memory usage. As a result of these experimental studies, Quad KD Tree outperformed in range search analysis over the other techniques, especially when the data set is massive and consists of different geometry types.

1. INTRODUCTION

With the increase in spatial data usage, spatial query arises as a fundamental problem in numerous applications with various geometric problems. Spatial query is used to find the nearest neighbors lying within specified ranges of coordinates (Laurini and Thompson, 1992). Handling spatial queries in an effective manner, one needs access methods based on a data structure called an index are needed. Spatial indexing serves to optimize queries in spatial databases to organize records in memory space. The general idea of spatial indexing is to place spatial data in space or clusters stored in secondary storage. Some methods create clusters by partitioning the area, while others create clusters by grouping objects. Rigaux et al. classify these two methods as space-driven and data-driven spatial indexing methods (Scholl et al., 2002). In the space-driven structure, objects are partitioned into rectangular cells, and cells are mapped concerning spatial associations (overlap or intersection). KD Tree and Quad KD Tree are examples of this type of data structure in this study. Whereas, in the data-driven structures, the set of objects is partitioned and grouped according to the distribution of the objects. PR Tree is an example of this structure in this paper. These methods primarily try to access spatial data as fast as possible. For this reason, these methods are also called spatial access methods(Manolopoulos et al., 2000).

The spatial access method needs to take into account not only spatial indexing but also clustering techniques. As spatial datasets, mostly, are so large that they cannot reside in the main memory and must be stored in secondary storage such as a disk (Longley, 2005). Clustering is often required to group objects that are close together. Without such clustering, many different disk pages would have to be fetched, resulting in a delayed response. Through clustering, objects that are spatially close to each other are also stored close together in memory. Rigaux et al. propose the using the Minimum Bounding Rectangle (MBR) as a geometric key for constructing spatial indices (Scholl et al., 2002). An MBR is a rectangle that minimally encloses the geometric objects in two-dimensional space. It consist of four coordinate values x_{max}, y_{max}, x_{min}, y_{min}. The coordinate values represent the left, right, upper and lower corners of the bounding box. With this way, it is provided to reduce expensive geometric predicates during index traversal and make constant size clustering entries to simplify the structure's design.

Spatial indexing data structures play a crucial role in displaying and finding the records for a given specific boundary, especially in time-critical applications and manipulations of massive spatial data. The most crucial purpose of these data structures is to reduce the number of comparisons to solve the range searching problems (Lewenstein, 2013). A range search is an essential query operation that retrieves all records within specified upper and lower boundary values in the spatial database (Mark de Berg, 2008).

The primary purpose of this study is to present a benchmark study of spatial indexing techniques covering an experimental analysis for use in real-time safety-critical avionics navigation map application. For this map application, the main goal is to search the various critical spatial layers such as obstacle (electric tower, power line, etc.), border line, restrictive space in the storage and then render. In addition, the time constraint is crucial since real-time operating systems are based on time priority. In this context, well-known and frequently used spatial indexing methods are examined, and these methods are presented

* Corresponding author

in this study. Among these methods, a comprehensive experimental study has been made for the methods compatible with this system in terms of memory and performance constraints.

The experimental study consists of two stages: Construction of the data structures, and Experimental analysis of the data structures. The stage of creating data structures is considered as a pre-process. In this phase, the evaluation of data structures created by algorithms in terms of elapsed time and space allocation is presented. Experimental analysis of the data structures is the fundamental element that will determine this application's performance. This analysis is based on range queries over and over to cover the entire data set. As a result of a query, a set of records within the range and required time to access these records are obtained. This time is equivalent to how many nodes have been traversed to access these records within the data structure. As a result of all range queries, the worst-case shows the performance of these indexing algorithms.

1.1 Related Works

The spatial indexing techniques are essential in various fields: database management, geographic information system, computer graphic, game programming, terrain visualization, image processing, and many others. In the context of spatial indexing algorithms literature, the studies have been generally focused on theoretical aspects rather than experimental studies. Accordingly, the experimental analyses have been conducted to provide the performances of their proposed approach in comparison to several spatial access methods. Elashry et al. presented a new partitioning technique based on the PR Tree algorithm and comparisons of KD Tree, Quad Tree, and PR Tree with each other in SpatialHadoop (Elashry et al., 2018). Sayar et al. present a study that compares the algorithms of KD Tree and Quad Tree concerning the feasibility and efficiency of using these partitioning techniques (Sayar et al., 2015). Azri et al. present a survey on spatial indexing techniques for the scope of large urban data management applications (Azri et al., 2013). Kothuri et al. compare Quad Tree and R tree algorithms regarding the performance of large GIS datasets in Oracle Spatial (Kothuri et al., 2002). Zhang et al. propose a new index structure based on the R tree, named UI-Tree. It is space-effective and complies with different shapes of range query (Zhang et al., 2010). Wei presents a study that describes the basic principles of the KD Tree and Quad Tree with the variation of these algorithms. Furthermore, he enounced that when indexing a large data set, if the Quad Tree depth is too small, the search performance is low; if it is too large, storage deteriorates and index efficiency decreases due to the increase of duplicate data (Wei, 2010). The related studies mentioned in this section present various spatial indexing techniques and their analysis for some interest fields. However, to our knowledge no previous study has presented the experimental analysis in real-time safety-critical map applications. In this paper, the spatial indexing techniques are described, and an extensive analysis is performed to measure the performance of these techniques in the field of real-time application.

2. METHODOLOGY

2.1 KD Tree

As Bentley and Friedman introduced, KD Tree (k-dimensional tree) is a data structure based on the binary search tree concept (Bentley and Friedman, 1979). It is useful for solving the space partitioning and range searching problems to organize objects in a space with k dimensions. Unlike a standard binary search tree, which uses only one dimension for all levels, KD Tree uses k dimension and cycles through these dimensions for all levels of the tree (Moore, 1991). It recursively splits space into rectangular cells along the dimension in which the objects have the greatest spread. Each level of the K-d tree specifies the branches based on a specific search key associated with that level, called the discriminator. Each non-leaf node, on the other hand, divides the space into two parts with respect to the discriminator. The left subtree of that node represents objects in the left part, and objects falling to the right part represented by the right subtree.

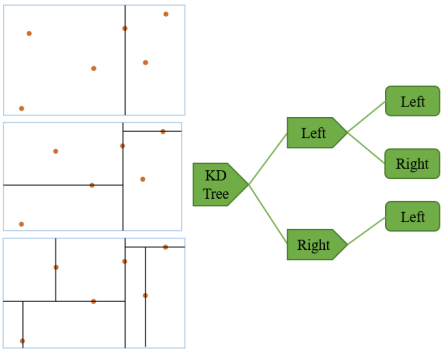

Figure 1. A Representation of a KD Tree structure

Although there are many KD tree variations, we consider balanced and 2-dimensional (Procopiuc et al., 2003) KD tree in this paper. Since the datasets that we use in our experimental study contain latitude and longitude attributes, also called xy-coordinate, with the longitude field (x-coordinate) nominated as dimension 0; latitude field (y-coordinate) nominated as dimension 1 (See Figure 1). At each level, the discriminator cycles between x-coordinate and y-coordinate, respectively. To construct a balanced tree, we also consider each node's discriminator value as the median of the objects for each recursive subtree.

Algorithm 1 Construction of a KD Tree

Input Point list P and the current $level$.
Output The root of tree
function CONSTRUCTKDTREE($P, level$)
  Determine P minimum bounding rectangle (MBR)
  if P contains only one point **then**
    return a leaf storing this point
  else
    if $level$ is even **then**
      Split P into two part with a vertical line l through the median x-axis of the points in P.
      Let P_1 be the set of points to the left of l or on l, and let $P2$ be the set of points to the right of l.
    else
      Split P into two part with a horizontal line l through the median y-axis of the points in P.
      Let P_1 be the set of points below l or on l, and let $P2$ be the set of points above l.
    end if

    n_{left} = CONSTRUCTKDTREE(P_1, level + 1)
    n_{left} = CONSTRUCTKDTREE(P_1, level + 1)
    Create node n storing, make n_{left} the left child of n and make n_{left} the right child of n
return n

The construction of a KD tree recursive algorithm is described in CONSTRUCTKDTREE algorithm. Bentley and Friedman show

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-V-4-2021-41-2021 | © Author(s) 2021. CC BY 4.0 License.

that, for n points, building time is $O(n \log n)$ time and total amount of storage is $O(n)$. A range query takes $O(\sqrt{n}+t)$ time where t is the number of points found (Bentley and Friedman, 1979). The range query algorithm is described in RANGEQUERYKDTREE recursive algorithm.

Algorithm 2 KD Tree Range Query

Input The root of tree and range R (Region of Interest).
Output The list point that lie in the range $IntersectedNodeList$ and the number of traversed node count in order to access record $TraversedNodeCount$

$TraversedNodeCount$ = 0
function RANGEQUERYKDTREE(n, R)
  if n is a leaf **then**
    Insert points to $IntersectedNodeList$ stored at n if it lies in R.
  else
    for each sub-node sn in node n **do**
      if sn boundary (MBR) intersects R **then**
        $TraversedNodeCount$ += 1
        RANGEQUERYKDTREE(sn, R)
      end if
    end for
  end if
  return $IntersectedNodeList$,$TraversedNodeCount$

2.2 Quad KD Tree

The Quad KD Tree is a combination of KD tree and Quad Tree. As Bereczky et al. introduced, Quad KD Tree (QKd shortly) is a data structure for the storage of multidimensional spatial objects(Bereczky et al., 2014). Unlike the heuristic method proposed by Bereczky et al., we worked on a new space-driven partitioning technique that contributes to this study. The method aimed to partition as in KD Tree while the number of partition in one cycle - or in other words, the number of the child- is determined as in Quad Tree. The purpose of this data-dependent method aims to partition by finding the median coordinate value as in the balanced KD tree. The balanced KD-tree divides the set of points into two sets of (approximately) equal size (Bentley and Friedman, 1979), whereas Quad KD Tree divides into four subsets with respect to the median value of points. It is based on the equal division of 2D space by four regions according to a particular value. Each node represents a bounding box covering a set of points depending on the coordinate along the x-axis and y-axis (Zhang and Du, 2017). Furthermore, each node has four nodes, north-west, north-east, south-west, and south-east (Azri et al., 2013) (See Figure 2).

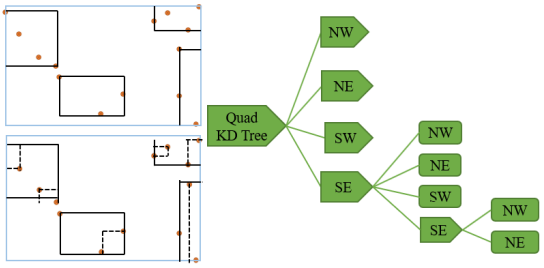

Figure 2. A Representation of a Quad KD Tree structure

Each node splits the set of points into four subsets according to vertical and horizontal axes based on whether their coordinate along these axes is greater than or less than a median coordinate value. This splitting continues until each set contains a particular value at most. This value represents the 'maximum objects count' in one MBR and indicates the number of points at most under a leaf in a tree. This value is preferred to fit within one disk page. The nodes are regimented, and each group is stored on one page to minimize the number of disk access. In order to make the number of points within the leaf more balanced and adaptive(Egas et al., 2003) in constructing a tree, additional splitting by two is applied in some cases (see CONSTRUCTQUADKDTREE algorithm).

Algorithm 3 Construction of a Quad KD Tree

Input Point list P and the current $level$.
Output The root of tree
function CONSTRUCTQUADKDTREE(P, $level$)

  Determine P minimum bounding rectangle (MBR)
  if P contains less than $MaxFeatureCount$ point **then**
    return a leaf storing this point
  else
    if P contains more than $2*MaxFeatureCount$ point **then**
      Split P into two part (P_1,P_2) with a vert. line
      Split P_1 into two part (P_3,P_4) with a horiz. line
      Split P_2 into two part (P_5,P_6) with a horiz. line

      n_{nw}=CONSTRUCTQUADKDTREE(P_3, level+1)
      n_{ne}=CONSTRUCTQUADKDTREE(P_4, level+1)
      n_{sw}=CONSTRUCTQUADKDTREE(P_5, level+1)
      n_{se}=CONSTRUCTQUADKDTREE(P_6, level+1)
    else
      Split P into two part (P_1,P_2) with a vert. line

      n_{nw}=CONSTRUCTQUADKDTREE(P_1, level+1)
      n_{ne}=CONSTRUCTQUADKDTREE(P_2, level+1)
    end if

    Create a node n storing , make n_{nw}, n_{ne}, n_{se}, and n_{sw} north-west, north-east, south-west, and south-east child of n, respectively
  return n

Algorithm 4 Quad KD Tree Range Query

Input The root of tree and range R (Region of Interest).
Output The list point that lie in the range $IntersectedNodeList$ and the number of traversed node count in order to access record $TraversedNodeCount$

$TraversedNodeCount$ = 0
function RANGEQUERYQUADKDTREE(n, R)
  if n is a leaf **then**
    Insert points to $IntersectedNodeList$ stored at n if it lies in R.
  else
    for each sub-node sn in node n **do**
      if sn boundary (MBR) intersects R **then**
        $TraversedNodeCount$ += 1
        RANGEQUERYQUADKDTREE(sn, R)
      end if
    end for
  end if
  return $IntersectedNodeList$,$TraversedNodeCount$

The construction of a Quad KD tree recursive algorithm is described in CONSTRUCTQUADKDTREE algorithm. The range query algorithm is described in RANGEQUERYKDTREE recursive algorithm.

- KD tree: for n points, building time is $O(n \log n)$ time, and the total amount of storage is $O(n)$. A range query takes $O(\sqrt{n}+t)$ time, where t is the number of points found (Lewenstein, 2013).
- Quad Tree: for n points, building time is $O((d+1)+t)$

time, where d is the depth of the tree. A range query takes $O(n \log n)$ (Mark de Berg, 2008).

2.3 PR Tree

As Arge et al. proposed in (Arge et al., 2008), PR Tree (Priority R-tree) is a data structure that is a provably asymptotically optimal R-tree variant. The term priority in the PR tree's name originates from the bulk loading algorithm using priority rectangles. It is a data-driven structure that uses the idea of a spatial containment relationship instead of the order of the index (Scholl et al., 2002).

A PR Tree consists of a node with six children, the four leaves (most-left, most-bottom, most-right, and most-top), and two recursive subtrees (left, right). Regarding the construction of the priority leaves: the first such leaf is denoted as the most-left leaf and contains the leftmost geometric features in two-dimensional space. Analogously, the other leaves contain the bottommost, rightmost, and topmost objects, respectively. Hence, the priority leaves store the objects according to the minimum and maximum of coordinates' values. After leaves construction, the rest of the space is divided into two subsets to create two recursive sub-trees. The splitting is applied by using a round-robin procedure similar to constructing a four-dimensional KD Tree. Each leaf node contains a particular number of MBR, and each MBR contains a particular number of geometric features (Arge et al., 2008). The structure is represented as in Figure 3.

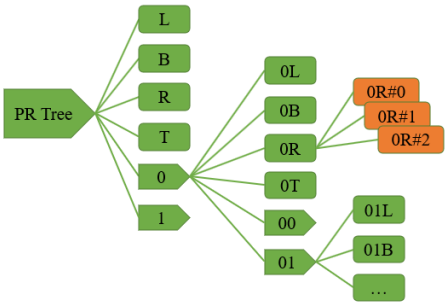

Figure 3. A Representation of a PR Tree structure

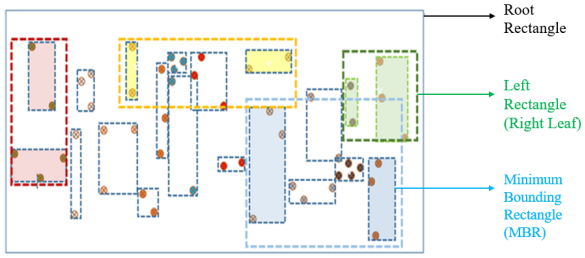

Figure 4. General View of a PR Tree Construction

The construction of a PR tree algorithm has been described in CONSTRUCTPRTREE algorithm and represented as in Figure 4. Arge et al. (Arge et al., 2008) show that, construction time is $O(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B})$ time where N is the number of rectangles and B is the number of objects that is stored in each node. A range query takes $O(\sqrt{\frac{N}{B}} + \frac{T}{B})$ time in the worst case, where T is the number of rectangles that satisfy the range query.The range query algorithm is described in RANGEQUERYPRTREE recursive algorithm.

Algorithm 5 Construction of a PR Tree

Input Point list P and the current $level$.
Output The root of tree
function CONSTRUCTPRTREE($P, level$)
  Determine P minimum bounding rectangle (MBR)
  Cluster points in both vertical and horizontal direction
  Create leaf node n_l for the most-left part
  Create leaf node n_b for the most-bottom part
  Create leaf node n_r for the most-right part
  Create leaf node n_t for the most-top part

  if Any point is left **then**
    Split rest of points into two part P_1 and P_2
    n_{lt}=CONSTRUCTPRTREE(P_3, level+1)
    n_{rt}=CONSTRUCTPRTREE(P_3, level+1)
  end if

  Create a node n storing four leafs (n_l, n_b, n_r, and n_t) and two inner tree left (n_{lt}) and right (n_{rt}).
return n

Algorithm 6 PR Tree Range Query

Input The root of tree and range R (Region of Interest).
Output The list point that lie in the range $IntersectedNodeList$ and the number of traversed node count in order to access record $TraversedNodeCount$
$TraversedNodeCount$ = 0
function RANGEQUERYPRTREE(n, R)
  for each leaf l (left, bottom, right, and top) in node n **do**
    if l boundary (MBR) intersects R **then**
      for each MBR m in MBR list of leaf l **do**
        $TraversedNodeCount$ += 1
        Insert points to $IntersectedNodeList$ stored at m if it lies in
      end for
    end if
  end for
  if n_{lt} left sub tree boundary (MBR) intersects R **then**
    $TraversedNodeCount$ += 1
    RANGEQUERYPRTREE(n_{lt}, R)
  end if
  if n_{rt} right sub-tree boundary (MBR) intersects R **then**
    $TraversedNodeCount$ += 1
    RANGEQUERYPRTREE(n_{rt}, R)
  end if
  return $IntersectedNodeList$,$TraversedNodeCount$

3. EXPERIMENTAL STUDY

3.1 Experimental Setup

All experiments were performed on a Hp Prodesk desktop with one Intel i5 processor with 16GB main memory running on Windows 10. A local 512GB M.2 SSD hard disk was used to store all required files. Since these studies are based on the comparison numbers, these comparisons have been made with a desktop. The numbers resulting from these comparisons show the algorithms' states relative to each other.

The real datasets extracted from Open Street Map were used for all experiments (See Table 1), precisely a point of interest (POI) data file that had 238K point records in Turkey, a POI data file that had 8.3M point records in Europe, a Street data file that had 1.6M line records in Turkey and a Street data file that 20M line records in Europe (Geofabrik Download Server, n.d.).

While creating the spatial access methods' data structures, the maximum number of objects in an MBR is used as a parameter. This number is called "bucket size" (Azri et al., 2013) and is mentioned as "Max Feature Count in One MBR" in this paper.

Location	Data Type	Data Size	Record Number
Turkey	POI	11.1 MB	238 K
Europe	POI	388 MB	8.3 M
Turkey	Street	671 MB	1.6 M
Europe	Street	5.11 GB	20 M

Table 1. Real Spatial Dataset

While determining this number, the skewness and the dataset's size were taken into account. When the number is determined as small, the number of nodes, leaves, and MBRs will be increased, and it will cause an increase in memory usage. For these reasons, if the dataset's size increases, this parameter is expected to increase, which gives positive results in terms of memory usage. In light of this information, the Max Feature Count's value in One MBR parameter was determined as 20, 50, 100, and 100 for Turkey POI, Europe POI, Turkey Street, and Europe Street real datasets, respectively.

3.2 Experimental Result

Experimental studies have been performed regarding space complexity and time complexity. While the space complexity has been evaluated by memory consumption for the indexed data, the time complexity has been measured by elapsed time for indexing. On the other hand, the minimum bounding rectangles' (MBR) figures created by the indexing algorithms are presented in the following sections. These rectangles represent the lowest level nodes where the desired records are involved.

3.2.1 Construction of the Data Structures: The value of "Max Feature Count in One MBR" was determined according to the dataset's data density and size. The data structures were created based on the algorithms. The memory consumption and the elapsed time for indexing time, for all real data set have been presented with the following charts (See Figures 5-6-7-8).

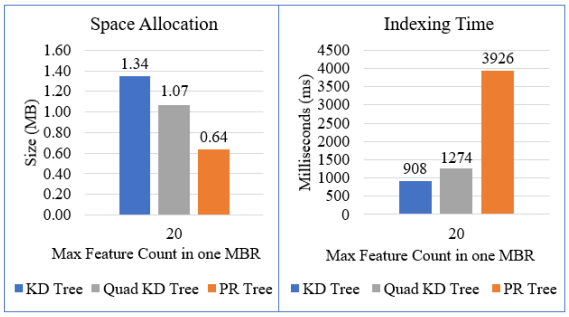

Figure 5. Space Allocation and Indexing Time for Turkey POI dataset

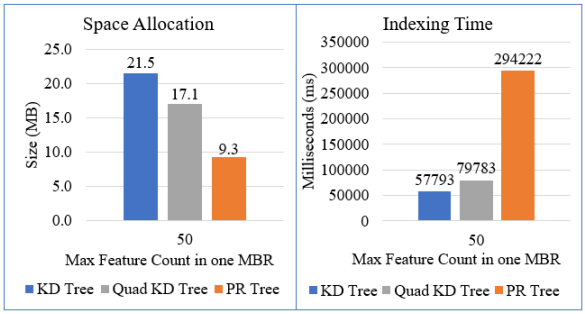

Figure 6. Space Allocation and Indexing Time for Europe POI dataset

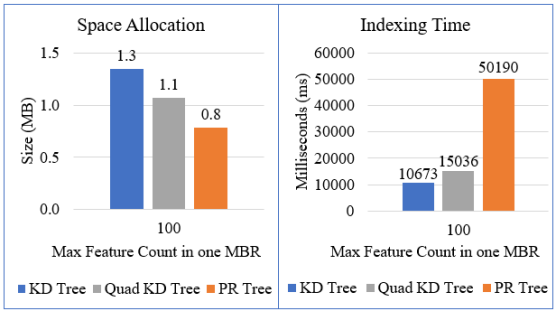

Figure 7. Space Allocation and Indexing Time for Turkey Street dataset

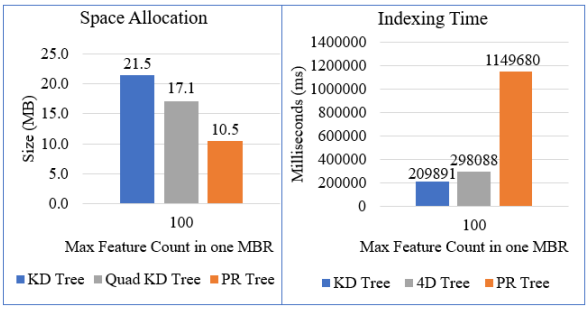

Figure 8. Space Allocation and Indexing Time for Europe Street dataset

As shown in the space allocations charts created according to different data sets, PR Tree outperforms in terms of space allocation, while Quad KD Tree allocates relatively more space than it. On the other hand, KD Tree is recorded as the most resource-consuming algorithm than others in terms of memory. Furthermore, in indexing time charts, KD Tree outperforms over the other indexing techniques since it is approximately one-fifth of the PR Tree indexing time and two-third of KD Tree indexing time.

For instance, the indexed data structure created after mentioned spatial indexing algorithm techniques is provided in the Figures 9-10-11 and in the Table 2 for Turkey POI real dataset. The images show the minimum bounding rectangles (MBR) that is created by the mentioned techniques. These rectangles represent the lowest level nodes where the desired records are involved. As can be seen in these figures, it can be observed that the Quad KD Tree clustered more explicit than the others. So that, it can be seen that the rectangles created by the KD Tree and PR Tree algorithm are larger and wider, whereas it can be seen that the rectangles created by the Quad KD Tree algorithm are tighter and smaller than the others.

Algorithm	MBR Count	Total Node Count
KD	16384	32767
Quad KD	16384	21845
PR	12268	18407

Table 2. MBR and Total Node Counts of the indexed data structure for Turkey POI Real Dataset

Constructing data structures is a preliminary preparation for the real-time map application that is planned to be used. Within the scope of these systems, the memory allocation process required for the data structure resulting from the preliminary stage is essential for the system's functioning. For this reason, PR Tree and Quad KD Tree come to the fore first, which has a significant effect on memory allocation. However, the main factor

Figure 9. KD Tree: Indexed Data Structure for Turkey POI Real Dataset

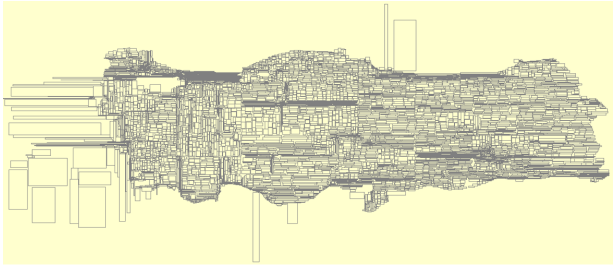

Figure 10. Quad KD Tree: Indexed Data Structure for Turkey POI Real Dataset

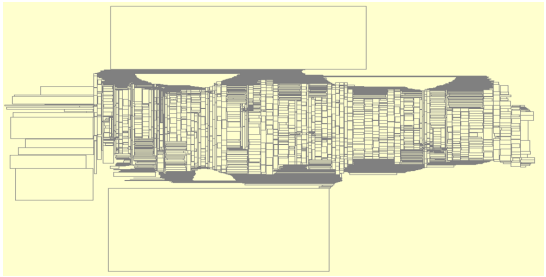

Figure 11. PR Tree: Indexed Data Structure for Turkey POI Real Dataset

determining the system's performance, in general, is the output it gives to search queries. In relation to this, a comprehensive analysis process is covered in the next section.

3.2.2 Boundary Analysis: The boundary analysis is performed to measure the spatial indexing algorithm performance, and this section presents the details of the boundary analysis to evaluate algorithms' run time performance.

Boundary analysis aims to make a comprehensive analysis to include the entire dataset in 2D space. The region of interest for using range search is created concerning the specified parameters. A range query is performed in relevant data structures for each region of interest to cover all boundaries in the dataset. In this analysis, the number of intersected MBRs and the traversed node (while traversing on the tree) are recorded. The comparisons are evaluated according to the maximum number of intersected MBRs and the maximum number of the traversed node for the worst case.

The boundary analysis parameters are specified according to the region of interest's height and width (Extent width and height) and the geographical degree varying horizontally and vertically for moving the rectangle. For this experimental study (5 KM, 0.025°), (25 KM, 0.1°) and (200 KM, 0.25°) have been selected as (region of interest extent width and height in KM, sampling intervals value in degree) value tuple. The size of the

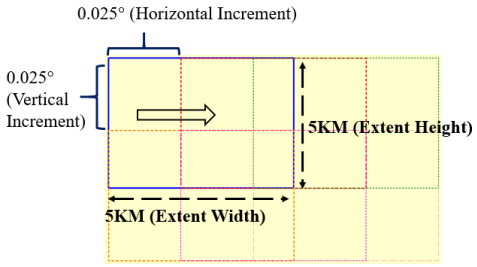

Figure 12. Region of Interests: Sliding Window for the Query

region of interest refers to the horizontal and vertical distances of the box (in KM), while the sampling interval value indicates how often the floating window (in degree) will move (See Figure 12). These values have been chosen to perform a range search to cover the whole area. The sampling interval values are determined in degrees since the dataset uses geometric degree coordinates system (WGS-84). For instance, the first value tuple (5 KM, 0.025°) is tried to find records within the area of 5 kilometers (box) horizontally and vertically. For the analysis to cover the whole area, 0.025 ° (approximately 2.5 KM) windows are shifted in the x-axis and y-axis, respectively.

Relevant MBRs within the specified area as a result of the range search query are retrieved. There are related features in these MBRs. Boundary analysis in this experimental study, the number of intersected features in these boxes is based on instead of the MBR numbers obtained the query result. The smaller number of MBRs generated by the algorithms does not mean fewer records at the intersection. For example, if we indexed the entire data set to a single node, we would always get a single MBR from a range query result. However, we would have to go through the entire dataset to find relevant records. Since this method is not a logical comparison method, it is a more logical measurement method to compare the intersected feature count obtained due to the range query.

With the given boundary analysis charts (See Figures 13-14-15-16-17-18-19-20), maximum intersected feature count and maximum traversed node count in the worst-case scenario of the data structures that are created according to the three spatial indexing techniques are presented.

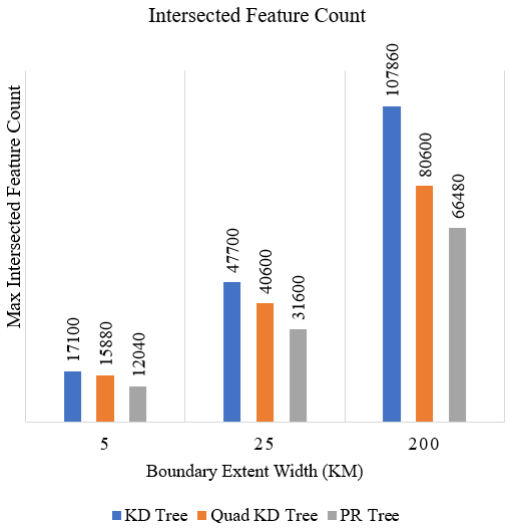

Figure 13. Intersected Feature Count for Turkey POI

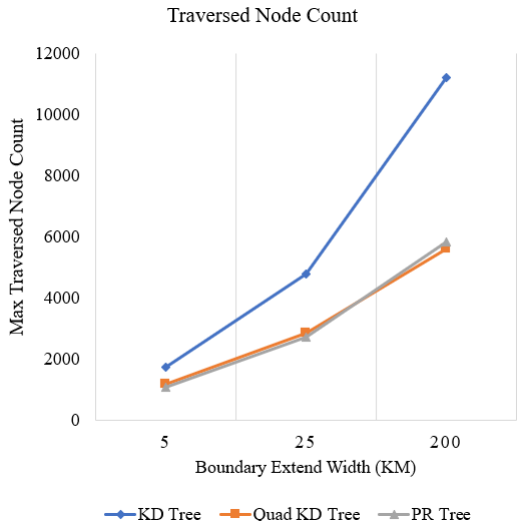

Figure 14. Traversed Node Count for Turkey POI

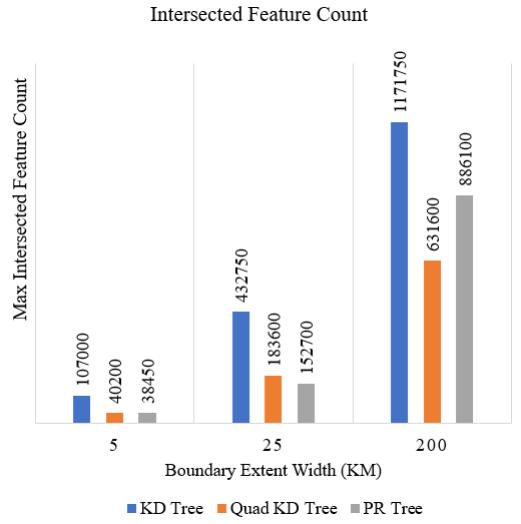

Figure 15. Intersected Feature Count for Europe POI

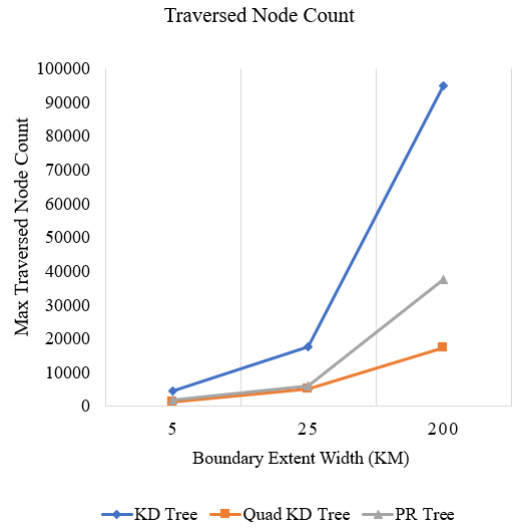

Figure 16. Traversed Node Count for Europe POI

When these results are evaluated, PR Tree and Quad KD Tree outperform as the two best results. For Turkey's real POI data set, PR Tree and Quad KD Tree are gave the nearly same results, but a little different in favor of PR Tree by a small margin. For Europe real POI data set, PR Tree and Quad KD Tree

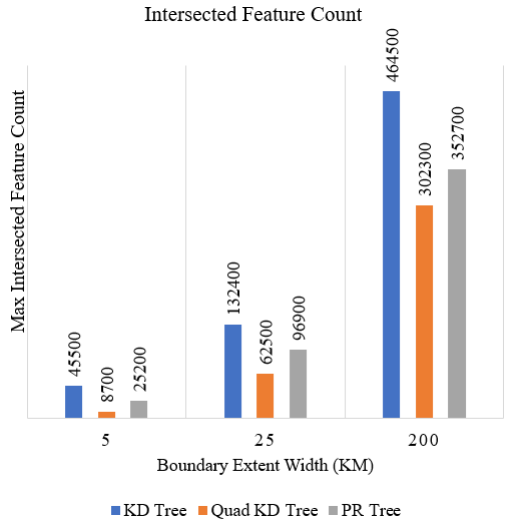

Figure 17. Intersected Feature Count for Turkey Street

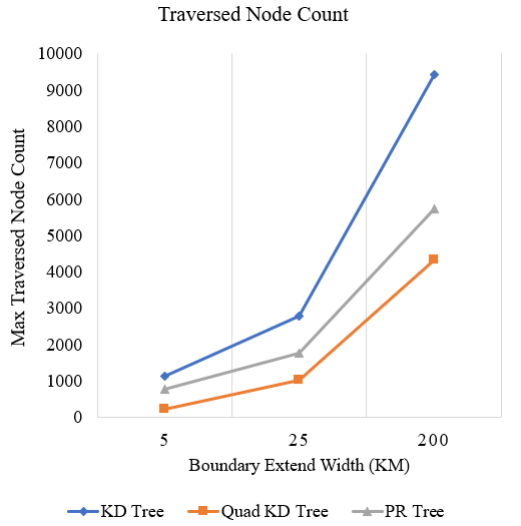

Figure 18. Traversed Node Count for Turkey Street

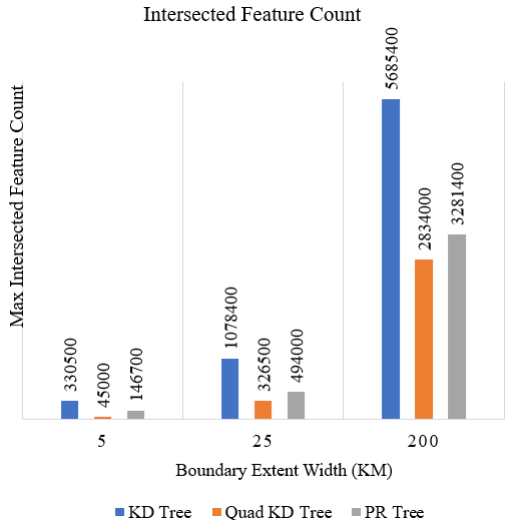

Figure 19. Intersected Feature Count for Europe Street

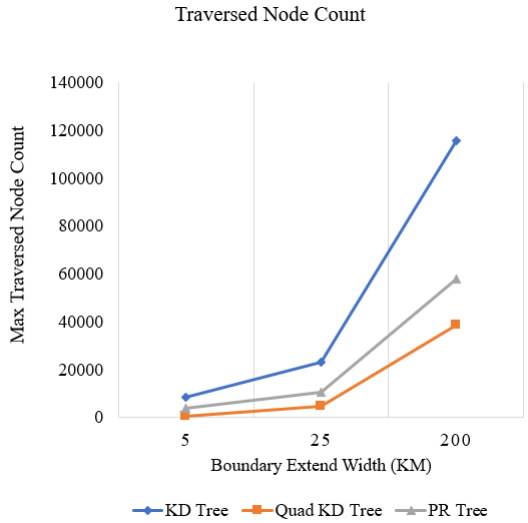

Figure 20. Traversed Node Count for Europe Street

are yielded nearly the same results for the 5 KM and 25 KM boundary analysis. However, Quad KD Tree is outperformed significantly over the others for 200 KM boundary analysis. For another data set, Turkey Street, Quad KD Tree is outperformed significantly over the other algorithms when all control parameters are taken into account. Besides, It can be observed that the algorithm with the least dead space in the nodes and MBR can be stated as the Quad KD tree. It is the reason that gives a good result during the range search in terms of the number of intersected feature counts and the number of traversed node count.

4. CONCLUSION

In this paper, the spatial indexing algorithms were explained and the differences between each other were presented. The amount of memory usage by these algorithms and the indexing time when run on the real dataset were presented and evaluated. On the other hand, an extensive experimental study was performed to compare the three spatial indexing algorithms. Various techniques were experimentally evaluated using different types of real datasets, and the experimental results of the worst-case scenarios were presented as experimental output.

In line with these experimental results, the KD tree gave the best result in terms of indexing time during construction, while Quad KD Tree came after the KD tree with a minimal difference. In another criterion, PR Tree and Quad KD Tree were identified as the two best memory allocation algorithms and performance algorithms. Moreover, PR Tree took relatively less space in memory for all real datasets than Quad KD Tree. However, as the size and the complexity of the dataset increased, the Quad KD Tree stood out remarkably in terms of range query performance (Geometry type can be evaluated as data complexity). In conclusion, when these experimental outputs are evaluated, the Quad KD Tree is better than the other spatial indexing algorithms in map applications of the safety critic avionics navigation systems since the real dataset is massive, comprehensive, and different in geometry type.

REFERENCES

Arge, L., Berg, M. D., Haverkort, H., Yi, K., 2008. The priority R-tree. *ACM Transactions on Algorithms*, 4(1), 1–30.

Azri, S., Ujang, U., Anton, F., Mioc, D., Rahman, A. A., 2013. Review of Spatial Indexing Techniques for Large Urban Data Management.

Bentley, J. L., Friedman, J. H., 1979. Data Structures for Range Searching. *ACM Computing Surveys*, 11(4), 397-409.

Bereczky, N., Duch, A., Németh, K., Roura, S., 2014. Quad-K-d trees.

Egas, R., Huijsmans, N., Lew, M., Sebe, N., 2003. Adapting k-d Trees to Visual Retrieval. 1614.

Elashry, A., Shehab, A., Riad, A., Aboul-Fotouh, A., 2018. 2DPR-Tree: Two-Dimensional Priority R-Tree Algorithm for Spatial Partitioning in SpatialHadoop. *ISPRS International Journal of Geo-Information*, 7(5), 179.

Geofabrik Download Server, n.d.

Kothuri, R. K. V., Ravada, S., Abugov, D., 2002. Quadtree and R-tree indexes in oracle spatial. *Proceedings of the 2002 ACM SIGMOD international conference on Management of data - SIGMOD '02*.

Laurini, R., Thompson, D., 1992. 15 - access and quality: Spatial indices and integrity constraints. R. Laurini, D. Thompson (eds), *Fundamentals of Spatial Information Systems*, Academic Press, Boston, 557–593.

Lewenstein, M., 2013. Orthogonal range searching for text indexing.

Longley, P. A., 2005. *Geographical information systems: principles, techniques, applications, and management*. John Wiley and Sons.

Manolopoulos, Y., Theodoridis, Y., Tsotras, V., 2000. Spatial Access Methods.

Mark de Berg, Otfried Cheong, M. v. K. M. O., 2008. *Orthogonal Range Searching*. Springer Berlin Heidelberg, Berlin, Heidelberg.

Moore, A., 1991. An introductory tutorial on kd-trees. Technical Report Technical Report No. 209, Computer Laboratory, University of Cambridge, Carnegie Mellon University, Pittsburgh, PA.

Procopiuc, O., Agarwal, P., Arge, L., Vitter, J., 2003. Bkd-Tree: A Dynamic Scalable kd-Tree. 46-65.

Sayar, A., Eken, S., Öztürk, O., 2015. Kd-tree and quad-tree decompositions for declustering of 2D range queries over uncertain space. *Frontiers of Information Technology Electronic Engineering*, 16(2), 98–108.

Scholl, M. O., Voisard, A., Rigaux, P., 2002. *Spatial Databases: With Application to GIS (Morgan Kaufmann series in data management systems)*. Morgan Kaufmann Publishers.

Wei, W., 2010. Analysis of spatial database index technology. *2010 2nd International Conference on Computer Engineering and Technology*, 4, IEEE, V4–29.

Zhang, X., Du, Z., 2017. Spatial Index. *Geographic Information Science Technology Body of Knowledge*, 2017(Q4).

Zhang, Y., Lin, X., Zhang, W., Wang, J., Lin, Q., 2010. Effectively indexing the uncertain space. *IEEE Transactions on Knowledge and Data Engineering*, 22(9), 1247–1261.