Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 5 Issue 01, January-2016

Indexing Structures for Range Searching with Point Objects: A Survey

P. Z. Piah
Department of Computer Science
Kenule Benson Saro-Wiwa Polytechnic
Bori-Rivers State, Nigeria

P. O. Asagba
Department of Computer Science
University of Port Harcourt
Port Harcourt, Nigeria

V. Ejiofor
Department of Computer Science
Nnamdi Azikiwe University
Awka, Nigeria

K. T. Igulu
Department of Computer Science
Kenule Benson Saro-Wiwa Polytechnic
Bori-Rivers State, Nigeria

***Abstract*—This paper discusses the various indexing structures for range queries. Attention is directed to point objects because of its correlation to tuples of the tables of relational databases. It discusses the structures in details and possible ways of implementing them in relational databases. A special attention is given to structures that are relatively efficient for range searching in multidimensional space.**

Keywords*—*Range; Searching; Query; Indexing; Structures; Multidimensional; Geometric Objects

I. INTRODUCTION

Peripherally, it seems that databases have little or nothing in common with geometry (precisely, computational geometry). But a deeper look into these two seemingly unrelated concepts will produce a mapping between them as illustrated in Table 1. To this end, we regard records or tuples in a database as points in a space (precisely multi-dimensional). Queries on the records of the database can be translated to queries of the points on the multi-dimensional space. Generally, supposing our interest is to perform queries on *k* attributes (columns) of the records in the database, records in the database are regarded as points in k-dimensional space. A range query that reports all records whose attribute values lie between intervals of the database attributes can be translated to a query of all points within a *k*-dimensional axis-parallel volume. In computational geometry parlance, such a query is called a rectangular range query, or an orthogonal range query or simply a range query[1].

Table I: Duality of Space and Database

DBMS	Computational Geometric
Database	Space
Tuples/Records/Rows	Points
Attributes	Dimensions
Values of attributes	Coordinates of dimensions

The main purpose of indexing a table of a database is to expedite query execution. This is achieved by utilizing the constraints imposed by a query in order to condense the number of disk accesses. The biggest challenge research in database community is to reduce the number of disk accesses since data cannot persist in the main memory. The quest is to develop efficient indexing structures. The b-Tree and its variants are the de facto structures used in most modern relation databases. Another challenge of the community is indexing structures for range queries spanning many attributes. This paper reviews structures that can aid efficient manipulation of range queries especially in relational databases utilizing computational geometric techniques. This paper is organized as follows: section I introduces the paper, section II discusses query and its various categories, section III discusses the selected and relevant indexing structures and section IV concludes our discussion.

II. QUERING (SEARCHING)

A. *Queries and Query Types*

Relational queries are primarily expressed by operators of the relational algebra and they operate either with a single table or span multiple tables [2]. Single table queries restrict, re-arrange or aggregate the tuples of one relation [3]. A query is a predicate φ(x) over the tuples of a relation R. The result set (RS) of a query is the subset of tuples of R sufficiently satisfy the query predicate in (1). The result set size is the cardinality of the result set as given in (2).

$$RS(R,\varphi) = \{x \varepsilon R \mid \varphi(x)\} \qquad (1)$$
$$|RS(R,\varphi)| \qquad (2)$$

A restriction query is a predicate φ(x) on the tuples 'x' of a relation R. A restriction query can be to a point-exact match query or on some dimension-partial match or partial range query. A range query is a special case of query with restrictions on all the dimensions. Reference [4] categorizes queries into single table queries and multiple table queries. Multiple table queries specifically join single table queries of different tables. Single table queries are further categorized into restriction queries and queries of re-arrangement which can be sorting, projection , grouping and aggregation. Partial range query is a type that gives restriction on some dimensions of the query and some unrestricted. This can be further categorized as exact-match query or range query.

B. Range Queries

Range queries give restriction on all dimensions (attributes) of the query. From the geometric perspective, let P be a set of n points in $Đ^k$ (i.e. k dimensional- $Đ^k = \{d_1, d_2, d_3, \ldots, d_k\}$) and let D be a family of subsets of $Đ^k$ (i.e.$D \subseteq Đ^k$). Let r_i represent a range of the i^{th} dimension ($r_i \subseteq d_i$). We call r_i an interval represented by its lower and upper bound ($r_i = (l,h)$). Elements of D are called ranges (i.e. $D = \{r_1, r_2, \ldots, r_k\}$). Let Δ be points subsumed by the range set D. Given the above, the requirement is to build appropriate data structure that supports range reporting, range count (number of points in the result set or the cardinality of the result set), emptiness query (determining if the result set size is not zero) given in (3), (4) and (5) respectively.

$$\Delta \cap P \qquad (3)$$
$$|\Delta \cap P| \qquad (4)$$
$$\Delta \cap P = \Phi \text{ or } |\Delta \cap P| = 0 \qquad (5)$$

We shall regard an exact point query as a special case of range query whose intervals are equal values on the various dimensions. i.e. l=h for all the k dimensions. No two points on the plane have the same address (no two points have the same x-and y-coordinates for 2-d space).

III. THE SURVEY

A. Balanced Binary Search Tree (BST)

A BST is a binary tree whose leaves are at the same level. This usually efficient for range queries in one dimension (one attribute). Let $P = \{p_1, p_2, \ldots, p_n\}$ be the given set of points on the real line. A solution that uses an array is of course also feasible for 1-D space. However, solution with array apparently does not give room for efficient update operations on the set *P* [1]. This is generally the limitation of array. The points in the set P are stored in the leaves of the BST. The internal nodes maintain splitting values to guide the traversal. Splitting value stored at a node *v* is designated x_v. By assumption, the subtree at the left of a node *v* contains all the points smaller than or equal to x_v, and all points strictly greater than x_v are stored in the right subtree. To report points in a range [x:x′], we traverse the tree with the lower bound and upper bound i.e. x and x′ respectively in BST. Supposing μ and μ′ be the two leaves where the searches terminate, respectively. The points in the interval [x:x′] are the points stored in the leaves in-between μ and μ′, μ and μ′ points possibly inclusive. Reference [1] summarizes as a theorem : "Let P be a set of n points in 1-dimensional space. The set P can be stored in a balanced binary search tree, which uses S(t)=O(n) storage and has P(t)=O(nlogn) construction time, such that the points in a query range can be reported in time Q(t)=O(k+logn), where k is the number of reported points."

B. Quad-Tree

The quad-tree was one of the early data structures for rectangle-parallel/orthogonal range searching first mentioned in [5]. The quad-tree was proposed as a data structure for composite key. The quad-tree, like kd-tree (subsection C) splits the space into iso-oriented hyper-planes. Although the term quad-tree in literatures mainly refers to the 2-d variant but the concept can be applied to any arbitrary d. The original work of [5] was principally a multi-dimensional binary search tree for point data. It must not necessarily be a balanced tree. Since then, there has been hundreds of publications dealing with quad-trees. References [6,7,8,9,10] give a far-reaching synopsis of the various types of quad-trees and their applications. Reference [11] introduced the region quad-trees which was based on regular (perfect-equal-sized) decomposition of the space into 2^d subspaces. The uniform partitioning greatly impact the performance of searching. Reference [12] proposed the PM quad-tree which can store polygonal data directly. PM quad-trees divide the quad-tree regions (and the data objects in them) until they contain only a small number of polygon edges or vertices.

C. Kd-Tree

Regrettably, the worst-case behavior of quad-trees is quite bad. Barely a year after, the kd-tree which is an improved quad-tree was first mentioned by [13]. According to [13], the k-d-tree is a binary search tree that represents a recursive subdivision of the universe into subspaces by means of (*d*-1)-dimensional hyper-planes. The hyper-planes are iso-oriented, and their direction interchanges among the *d* possibilities. Each splitting must contain at least one point. The Insertion and searching operations are quite simple and straightforward but the deletion operation is quite complicated which could result to reorganization of the sub-trees beneath the deleted point. The Kd-tree as proposed by [13] is mainly for point data. The main limitation of the original Kd-tree is its sensitivity to the order inwhich the points are inserted and points are strewn all over the universe. In 1979 [14] introduced the Adaptive Kd-tree. The Adaptive Kd-tree ameliorates the problems of the original Kd-tree by choosing a split of almost equal number of points on both sides of the plane. Splitting is continued recursively until each subspace holds only a certain number of points. The adaptive k-d-tree is static in principle; it is apparently difficult to keep the tree balanced where frequent insertions and deletions is the order. Adaptive Kd-tree works best if all the data are known beforehand (static) and if updates are infrequent. By this, the structure has a bad performance in dynamic cases.

The Bintree due to [15] is another variant of the Kd-tree. This structure subdivides the universe recursively into *d*-dimensional boxes of equal size until each contains only a certain number of points. Although it is apparent this kind of partitioning is less adaptive, it has several advantages, such as the implicit knowledge of the partitioning hyper planes [4].

Other variants of Kd-tree worth mentioning are K-d-B-Trees due to [16], hB-Trees due to [17,18], Extended Kd-tree due to [19], BD-Tree due to [20], SKD-Tree due to [21], GBD-Tree due to [22], LSD-Tree due to [23], KD2B-Tree due to [24], G-Tree due to [25].

The k-d-B-Trees exhibit a forced split effect, which does not allow one to give any space utilization guarantees. In worst case a large amount of pages may be completely empty. The hB-Trees have a complex organization and extremely difficult algorithms, since they are a hybrid data structure. In addition hB-Trees may store several references of a node to the same child node, which may result in a super linear growth of the index nodes with respect to the number of regions in space.The performance of Kd-trees is summarized in [1] which uses O(n) for storage, O(nlogn) for preprocessing and $O(\sqrt{n} + k)$ for rectangular range query reporting. Where k is the number of points reported. For d-dimesnsional space, the query time is bounded by $O(n^{1-1/d}+k)$. Common limitations of the kd-tree and its variants is that for some distributions and cases no hyper-plane can be found that splits/partitions the data points evenly and uniformly [18], they rely on the order of insertion (it's not appropriate for ordered data), dead (empty immaterial) spaces are covered and as such it is not sufficiently adequate for secondary memory indexing [26].

D. Range Trees

The Range tree was independently proposed by several researchers [27,28,29,30]. The range trees ameliorate the query time of a range query at the expense of storage time (speed and space tradeoff) compared to the Kd-trees. The range tree is a multi-level/layered structure. It handles the intervals of the dimensions independently by constructing canonical structures. The performance as summarized in [1] uses O(nlogn) for storage, preprocessing time of O(nlogn) and $O(\log^2 n + k)$ for range query. Where k is the number of points reported. Experience shows that the query time can be enhanced utilizing fractional cascading. Reference [28,29] described an improved query time to $O(\log n+k)$ by fractional cascading. Fractional cascading applies in fact not only to range trees, but in many situations where many searches are done with the same search key [1]. Reference [31,32] discuss this technique in its full granularity. Reference [33] disscussed the usage of fractional cascading in a dynamic setting.

Reference [34] described the modified and improved version of the layered range tree which is the most efficient data structure for 2-dimensional range queries; he enhanced the storage to $O(n\log n/\log \log n)$ while keeping the query time $O(\log n+k)$. Reference [35,36] also verified the optimality of the modification. If the query range is unbounded to one side (i.e. $[x : x']\times[y : +\infty]$), then $O(\log n)$ query time can be achieved with only linear space, using a priority search tree[1]. In higher dimensions the best result for orthogonal range searching (albeit in theory) is also due to [35] which proposes a structure for d-dimensional queries with $O(n(\log n/\log \log n)^{d-1})$ storage and polylogarithmic query time. This result is apparently optimal (albeit in theory). Storage and query time trade-offs are also possible [37, 38].

Reference [39] describes more efficient data structures for range searching when the points lie on a $U\times U$ grid, yielding query time bounds of $O(\log\log U +k)$ or $O(\sqrt{U} +k)$, contingent on the preprocessing time allowed. The results use data structures described earlier [40,41]. With respect to general case, better time bounds can be attained for many computational geometry problems if the coordinates of the objects are restricted to lie on grid points. Examples are the nearest neighbor searching problem [41], point location, and line segment intersection [42]. For queries unbounded on one side, priority tree [43], interval trees [44,45] and segment trees are recommended structures for querying. Since our focus is on point data, we assume queries are bounded on all sides and thus we skip the discussion of these unbounded-based structures in much detail.

E. R-Tree

The R-Tree is due to [46]. It is a height-balanced tree like the B-Trees. An R-tree corresponds to a hierarchy of nested *d*-dimensional intervals (boxes). Each node n of the R-tree corresponds to a disk page and a *d*-dimensional interval $I^d(v)$. If v is an interior node then the intervals corresponding to the descendants v_i of n are contained in $I^d(v)$. Intervals at the same tree level may overlap. If v is a leaf node, $I^d(v)$ is the *d*-dimensional minimum bounding box of the objects stored in v. For each object in turn, v stores only its Minimum Bounding Box (MBB) and a reference to the complete object description. The following are the properties of R-Tree according to [46].

i. Every node contains between *m* and *M* entries unless it is the root. The lower bound *m* prevents the degeneration of trees and ensures an efficient storage utilization. Whenever the number of a node's descendant's drops below *m*, the node is deleted and its descendants are distributed among the sibling nodes (*tree condensation*). The upper bound *M* can be derived from the fact that each tree node corresponds to exactly one disk page.
ii. The root node has at least two entries unless it is a leaf.
iii. The R-tree is height-balanced; that is, all leaves are at the same level. The height of an R-tree is at most ceiling of $\log_m (N)$ for *N* index records (*N* . 1).

Searching in R-Tree is similar to the B-Tree. R-Trees cannot give any performance guarantee for the basic operations, since they do not partition the multidimensional space in disjoint parts, but allow overlapping rectangles. Successors of the R-Tree like the R*-Tree [47] and the X-Tree [48] use complicated algorithms or even introduce buckets of varying size to minimize overlaps. However, complicated algorithms cannot overcome this problem in general. Introducing buckets of varying size may cause the index to degenerate. So the basic problem of R-Trees still remains.

F. Grid File

The Grid File [49] is a typical representative of an access method based on hashing. The grid file superimposes a d-dimensional orthogonal grid on the universe. Because the grid is not necessarily regular, the resulting cells may be of different shapes and sizes. A grid directory associates one or more of these cells with data buckets, which are stored on one disk page each. Each cell is associated with one bucket, but a bucket may contain several adjacent cells. Since the directory may grow large, it is usually kept on secondary storage. To guarantee that data items are always found with no more than two disk accesses for exact match queries, the grid itself is kept in main memory, represented by d one-dimensional arrays called scales. The grid file suffers from a super linear growth of the directory even for data that are uniformly distributed [50]. Grid-files give a two-access-guarantee for retrieval, but have an extremely bad worst-case behavior for updates: Inserting a point may result in a non-local split of the grid and thus require a reorganization of the grid-file. Furthermore, grid files have problems with dependencies in the multidimensional data distribution. For linearly dependent data the grid may require more storage than the tuples stored in the grid.

G. *Bang File*

Reference [51] proposed a new structure called the BANG (Balanced and Nested Grid) file to obtain a better adaption to given data points. Albeit it differs from the grid file in many facets. Analogous to the grid file, it partitions the universe into intervals (boxes). However the difference is that in BANG file, bucket regions may intersect. This can't occur in the regular grid file. Precisely, one can form nonrectangular bucket regions by taking the geometric difference of two or more intervals (nesting). To increase storage utilization, it is possible during insertion to reallocate points between different buckets. To manage the directory, the BANG file uses a balanced search tree structure. In combination with the hash-based partitioning of the universe, the BANG file can therefore be viewed as a hybrid structure. In order to achieve a high storage utilization, the BANG file performs spanning splits that may lead to the displacement of parts of the tree. As a result, a point search may in the worst case require the traversal of the entire directory in a depth-first manner. To address this problem, [52] later proposed different splitting strategies, including forced splits as used by the k-d-B-tree. These strategies avoid the spanning problem at the possible expense of lower storage utilization. Reference [25] made a similar proposal based on the BD-tree and called the resulting structure a *G-tree* (grid tree). The structure differs from the BD-tree in the way the partitions are mapped into buckets. To obtain a simpler mapping, the G-tree expenses the minimum storage utilization that holds for the BD-tree.

H. *B-Tree*

The B-Tree is due to [53] and its variants are the de-facto indexing structure for modern relational databases. They enjoy logarithmic performance of the basic operations of insertion, delete and exact match query with the exception of the range query. The performance setback of b-trees is that they work perfectly for indexing single attribute but performance deteriorate for multiple attributes. A popular approach to handling multidimensional search queries consists of the consecutive application of such single key structures, one per dimension. Unfortunately, this approach can be very inefficient [54]. Since each index is traversed independently of the others, we cannot exploit the possibly high selectivity in one dimension to narrow down the search in the remaining dimensions. In general, there is no easy and obvious way to extend single key structures in order to handle multidimensional data [4].

Instead of maintaining a single index structure for multiple attributes key, a total of d (d is the number of attributes of the table) indexes must be managed and updated upon insertion and deletion of objects. Also in a range query that requires these n attributes, the d indexing must be accessed. This is highly computationally expensive. Also, Multidimensional searching with several indexes has additive behavior. B-Trees do not also guarantee physical proximity of tuples and pages with respect to the dimensions (attributes) [55,56, 57, 58].

I. *Multidimensional Access Methods(MAMs)*

Loosely speaking, MAMs are set of methods to model spatial databases (stores spatial objects) for fast access. MAMs are categorized into Point Access Methods (PAMs) and Spatial Access Methods (SAMs). PAMs have primarily been designed to perform spatial searches on point databases (i.e., databases that store only points). The points may be embedded in two or more dimensions, but they do not have a spatial extension. Spatial access methods, however, manage extended objects, such as lines, polygons, or even higher-dimensional polyhedra. In literature, one often finds the term *spatial access method* referring to what we call m*ultidimensional access method*. Other terms used for this purpose also includes *spatial index* or *spatial index structure* [4]. MAMs can also be categorized into primary and secondary storage structures. Primary memory structures are used to manage multidimensional data in the main memory whereas the secondary storage structures are used for efficient management of large database in the secondary storage [4].

Reference [4] gives a comprehensive survey of MAMs. This work focuses on PAMs. Reference [4] also categorizes PAMs into: Techniques based on *hashing* (grid files [59], EXCELL [60], multi-level grid files [61], twin grid files [62] and multidimensional hashing [63,64]), *hierarchical access methods* (K-D-B-Tree [16], LSD-Tree [65], Buddy Tree [66], BANG File [51], hB-Tree [17], R-Trees [46, 67,68,48]) and *space filling curves in combination with one-dimensional access methods* [69,70,71,72,73]. Another technique is to use a *blend of several one-dimensional* methods such as inverted files [74,75] or bitmap index intersection [76].

Another paradigm that is quite promising in access methods is to map the multi-dimensional data onto a one-dimensional space filling curve (SFC) [77] like the Z-curve or the H-curve and use the properties of this curve for efficient retrieval. The thrilling thing about SFCs over the techniques described before is that they allow a disjoint partitioning of the multidimensional space (i.e. the partitions of the multi-dimensional space do not overlap). Another advantage is that the storage requirements do not degenerate for any data distribution and also it preserves the spatial proximity of multidimensional points in one dimensional space. Well known one-dimensional indexing methods can be applied and multidimensional search problems are reduced to linear search problems. Hence multidimensional insertion, deletion and point query algorithms inherit the complexities of the corresponding one-dimensional access method. Using B-Trees as one-dimensional access method allows to give logarithmic performance guarantees for the basic operations of insertion, deletion and point queries.

Most approaches based on SFCs were designed for spatial data, e.g., the zk-d-B-Tree [70], XZ-Ordering [48], DOT [78].

Reference [79] applies the Hilbert curve for indexing of multidimensional data and provides algorithms for the basic operations. It also analyses all basic curves and operations required for query processing. He provides alternative and optimized algorithms for the calculation of Z-curve, Gray-Code-curve, and Hilbert-curve. For the Hilbert-curve he proposes compressed state diagrams which work for up to 8 dimensions in order to speed up the calculation of the curve. For higher dimensional universes they are not suitable anymore due to their size requirements. Performance

measurements was also carried out with three million randomly generated data points in 3 to 16 dimensional spaces with a grid size of 2^{32} points. The page size was adapted to have the same capacity with respect to the number of records for each dimensionality. In practice there were 8450+/-150 pages which result in ≈355 tuples per page. He measured data file creation by random inserts, partial match queries, and range queries for the Hilbert-curve, the Moores Curve (a variation of the Hilbert-curve, [80]), Gray-Code-curve, Z-curve, and Grid-file [59].
The simple design of the Z-curve shows a considerable advantage over the Hilbert curve and Gray-Code-curve for both classes of address calculations and retains its cost linear to the address length with growing dimensionality. His preliminary measurements also show that the Z-curve is superior with respect to elapsed time for all measured queries, being faster by a factor of up to 4 for some cases. However it was loading up to 15% more pages in some case and ≈10% in average for these queries.

J. UB-Tree

The UB-tree is due to Rudolf and Markl [82] in their work on Mistral [81]. The UB-tree as described in [82] is a structure to index multi-dimensional data with linear complexities i.e. using a structure that has linear complexities. The UB-Tree exploits the capabilities of B-tree and Z-curve [79]. Each multi-dimensional data tuple is transformed into an integer (Z-address), which is inserted into the B-tree. Each node is a pair of integer ([α:β]) denoting the lower bound (α) and the upper bound (β) of a region on the plane respectively. It suffices to note at this point that the entire plane is regarded as a Z-region (Super-Z-region). For consistency, all regions will be regarded as simply Z-region. A leaf of the UB-tree which is mapped to a Z-region of the curve holds data (points) or link to the data. Usually, a region mapped to a disk block (or page). Range query can be handled by retrieval points in regions that are perfectly subsumed by the query box or that intersect the query box. Figure 1 shows (a) typical 2-D 8by8 space with 6 Z-regions (b) UB-tree-nodes corresponding to the z-regions in the space. The inner nodes of UB-tree recursively divides the space, such that a hierarchy of nested Z-regions is formed.

The original UB-Tree range query algorithm is exponential albeit it was ameliorated to a linear time. Reference [82] argued that the best underling B-Tree for the UB-Tree is the B+Tree because of the chaining of the leaves for range query. He also proposed a new split point algorithm which displaces the redundancy introduced by the split point tree as used in the original UB-Tree algorithm. Analysis shows that the bit interleaving operation is negligible and that the UB-Tree and most structures that utilizes SFC do not suffer of the *curse of dimensionality*. The Z-curve is preferred above other SFCs because the computation of point addresses is pretty cheap.

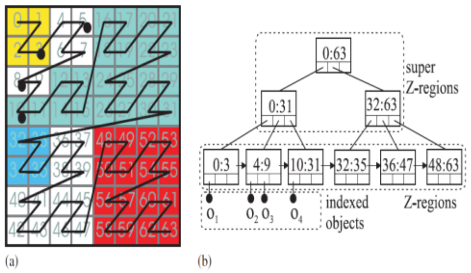

Figure 1: Z-curve with Z-regions and UB-Tree

The cost of insertion, deletion and point query operations of the UB-tree is the same as the underlying one-dimensional index structure (B-tree) but the address of the tuples in question must first be calculated. The range query is a more time-intensive operation of the UB-tree. Proposed algorithms to minimize the cost of range query operation can be found in [82,84]. Bit-interleaving is used to calculate the Z-address from the coordinates.

IV. CONCLUDING REMARKS

The paper discusses various structures that are used for indexing data in databases utilizing computational geometric techniques. Most of the structures from A to H either are main memory structures or single attribute structures. Most of the main memory structures have been extended to secondary storage structures but suffer of the curse of dimensionality (i.e. they deteriorate by increase in the number of attributes in the query). Most structures that utilizes SFCs do not suffer of the curse of dimensionality. Therefore they are the next generation indexing structures for OLAP applications that are characterized with complex queries spanning several dimensions.

REFERENCES

[1] M. D. Berg, O. Cheong, M. v. Kreveld and M. Overmars, Computational Geometry: Algorithms and Applications, Verlag Berlin Heidelberg: Springer, 2008.
[2] E. F. Codd, "A Relational Model of Data for Large Shared Databanks," *ACM*, vol. 13, no. 6, pp. 377-387, 1970.
[3] J. D. Ullman, Database and Knowledge Based Systems Volume I, Rockville, MD: Computer Science Press, 1988.
[4] O. Gunther and V. Gaede, "Multi-dimensional Acess Methods," *ACM Computing Surveys*, vol. 30, no. 2, 1998.
[5] J. L. Bentley and R. A. Finkel, "Quad trees: a data structure for retrieval on composite keys.," *Acta Inform.*, vol. 4, p. 1–9, 1974.
[6] H. Samet, "An overview of quadtrees, octrees, and related hierarchical data structures," Theoretical Foundations of Computer Graphics and CAD. NATO ASI Series F, vol. 40, pp. 51-68, 1988.
[7] H. Samet, Applications of Spatial Data Structure, Addison-Wesley, 1990.
[8] H. Samet, "The Design and Analysis of Spatial Data Structures," MA, Addison-Wesley, 1990.
[9] H. Samet, "Foundations of Multidimensional and Metric Data Structures," San Mateo, CA, Morgan Kaufmann, 2006.
[10] S. Aluru, "Quadtrees and octrees," Chapman & Hall/CRC, 2005.
[11] H. Samet, "The Quadtree and related hierarchical data structure," *ACM Computing Survey*, vol. 16, no. 2, pp. 187-260, 1984.
[12] H.Samet and R.E. Webber, "Hierarchical data structures," *2nd International Electronic Image Week, Nice*, vol.2, pp. 577-584, 1985.

[13] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *ACM Communication*, vol. 18, p. 509–517, 1975.

[14] J. L. Bentley and J. H. Friedman, "Data structures for range searching.," *ACM Computing Survey*, vol. 11, no. 14, p. 397–409, 1979.

[15] M. Tamminen, "Comment on quad- and octrees," *ACM*, p. 204–212, 1984.

[16] J. Robinson, "The K-D-B-Tree: A Search Structure for large multidimensional dynamic Indexes," in *ACM SIGMOD Conference*, 1981.

[17] D. B. Lomet and B. Salzberg, " The hBtree: A multiattribute indexing method with good guaranteed performance," in *ACM Transaction of Database Systems*, 1990.

[18] D. B. Lomet and B. Salzberg, "The hBtree: A robust multiattribute search structure," in *Fifth IEEE International Conference on Data Engineering*, 1989.

[19] T. Matsuyama, L. V. Hao and M. Nagao, "A file organization for geographic information systems based on spatial proximity," *Int'l journal of Computing Vis. Graph. Image Process*, vol. 26, no. 3, pp. 303-318, 1984.

[20] Y. Ohsawa and M. Sakauchi, "BD-tree: A new n-dimensional data structure with efficient dynamic characteristics," in *Ninth World Computer Congress, IFIP* , 1983.

[21] B. C. Ooi, R. Sacks-Davis and K. J. Mcdonell, " Spatial indexing by binary decomposition and spatial bounding," *Information System Journal*, vol. 16, no. 2, pp. 211-237, 1991.

[22] Y. Ohsawa and M. Sakauchi, "A new tree type data structure with homogeneous node suitable for a very large spatial database," *Sixth IEEE International Conference on Data Engineering*, p. 296–303, 1990.

[23] A. Henrich, H.-W. Six and P. Widmayer, "The LSD tree: Spatial access to multidimensional point and non-point objects," in *Fifteenth International Conference on Very Large Data Bases*, 1989.

[24] P. Oosterom, " Reactive data structures for geographic information systems.," 1990.

[25] A. Kumar, "G-tree: A new data structure for organizing multidimensional data," *IEEE Transaction of Knowledge and. Data Engineering.*, vol. 6, no. 2, p. 341–347, 1994.

[26] S. Bachtold and D. A. Kiem, "High-Dimensional Index Structure: Database support for next decades's in Applications," 2000.

[27] J. L. Bentley, "Decomposable searching problems," *Information Processing Letter*, vol. 8, pp. 244-251, 1979.

[28] G. S. Lueker, "A data structure for orthogonal range queries. In Proc.," in *19th Annual IEEE Symposium Foundation Computer Science*, 1978.

[29] D. E. Willard, "The super-b-tree algorithm," Cambridge, MA, 1979.

[30] D. T. Lee and C. K. Wong, "Quintary trees: A file structure for multidimensional database systems," *ACM Transction Database System*, vol. 5, p. 339–353, 1980.

[31] B. Chazelle and L. J. Guibas, "Fractional cascading: I. A data structuring technique," *Algorithmica*, vol. 1, p. 133–162, 1986.

[32] B. Chazelle and L. J. Guibas, "Fractional cascading: II. Applications," *Algorithmica*, vol. 1, p. 163–191, 1986.

[33] K. Mehlhorn and S. Naher, "Dynamic fractional cascading," *Algorithmica*, vol. 5, pp. 215-241, 1990.

[34] B. Chazelle. Filtering search: A new approach to query-answering. *SIAM J. Comput.*, 15:703–724, 1986.

[35] B. Chazelle, "Lower bounds for orthogonal range searching, II: The arithmetic model," *ACM*, vol. 37, p. 439–463, 1990.

[36] B. Chazelle, "Lower bounds for orthogonal range searching, I: The reporting case," *ACM*, vol. 37, p. 200–212, 1990.

[37] H. W. Scholten and M. H. Overmars, "General methods for adding range restrictions to to decomposable searching problems," *Symbolic Computing*, vol. 7, pp. 1-10, 1989.

[38] D. E. Willard and G. S. Lueker, "Adding range restriction capability to dynamic data structures," *ACM*, vol. 32, p. 597–617, 1985.

[39] M. H. Overmars, "Efficient data structures for range searching on a grid," *Journal of Algorithms*, vol. 9, p. 254–275, 1988.

[40] D. E. Willard, "New trie data structures which support very fast search operations," *Computer System Science*, vol. 28, pp. 379-394, 1984.

[41] R. G. Karlsson, "Algorithms in a restricted universe.," Waterloo, ON, 1984.

[42] R. G. Karlsson and M. H. Overmars, "Scanline algorithms on a grid. BIT," *BIT*, vol. 28, pp. 227-241, 1988.

[43] E. McCreight, "Priority search trees," *SIAM J. Comput.*, vol. 14, no. 1, pp. 257-275, 1985.

[44] M. Edward, "Efficient algorithms for enumerating intersecting intervals and rectangles," CSL-80-9, Xerox Palo Alto Res. Center, Palo Alto, CA, 1980.

[45] H. Edelsbrunner, "Dynamic data structures for orthogonal intersection queries," *Inst. Informationsverarb., Tech. Univ*, 1980.

[46] A. Guttman, "R-Trees: A dynamic Index Structure for spatial Searching," in *ACM SIGMOD*, 1984.

[47] N. Beckmann, H. Kriegel, R. Schneider and B. Seeger., "The R*-Tree. An efficient and robust Access Method for Points and Rectangles," in *ACM SIGMOD*, 1990.

[48] S. Berchtold, D. Keim and H.-P. Kriegel, "The X-Tree. An Index Structure for high dimensional Data," in *22nd VLDB*, 1996.

[49] J. Nievergelt, H. Hinterberger and K. Sevcik, "The grid file: An adaptable, symmetric multikey file structure.," in *LNCS 123, Springer-Verlag, ork*, Berlin/Heidelberg/New Y, 1981.

[50] M. Regnier, "Analysis of the grid file algorithms," in *BIT*, 1985.

[51] M. Freeston, "The BANG File: A new Kind of Grid File," in *ACM SIGMOD*, San Francisco, CA, 1987.

[52] M. Freeston, "Advances in the design of the BANG file," in *Third International Conference on Foundations of Data Organization and Algorithms* , Berlin/Heidelberg/New York, 1989.

[53] B. Rudolf and E. McCreight, "Organization and Maintenance of Large Ordered Indexes," *ACTA Information*, vol. 1, no. 3, pp. 173-189, 1972.

[54] H. P. Kriegel, "Performance comparison of index structures for multikey retrieval," in *ACM SIGMOD International Conference on Management of Data*, 1984.

[55] R. Bayer, "The universal B-tree for multidimensional indexing: general concepts," in *International Conference on Worldwide Computing and Its Applications, Springer*, Berlin, 1997.

[56] M. Franklin, "B+-Trees," wikipedia, 15 06 2015. [Online]. Available: https://www.cs.umd.edu/class/spring2006/cmsc424/notes/B+-Trees.ppt. [Accessed 26 06 2015].

[57] D. Comer, "The Ubiquitous B-Tree," *ACM Computing Surveys*, vol. 11, no. 2, pp. 121-137, 1979.

[58] D. Knuth, The Art of Computer Programming, Vol. 3: Sorting and Searching., Addison Wesley, 1973.

[59] J. Nievergelt, H. Hinterberger and K. Sevcik, "The Grid-File," *ACM TODS*, vol. 9, no. 1, pp. 38-71, 1984.

[60] M. Tamminen, "The extendible cell method for closest point problems," *BIT*, vol. 22, pp. 27-42, 1982.

[61] K. Hinrichs, " Implementation of the Grid File: Design Concepts and Experience," *BIT*, vol. 25, pp. 569-592, 1985.

[62] A. Hutflesz, H. Six and P. Widmayer, "Twin Grid Files: Space Optimizing Access Schemes.," in *ACM SIGMOD*, 1988.

[63] C. Faloutsos, "Multi-attribute Hashing Using Gray Codes," in *ACM SIGMOD*, 1985.

[64] C. Faloutsos, "Gray Codes for Partial Match and Range Queries," *IEEE TSE*, vol. 14, no. 10, pp. 1381-1393, 1988.

[65] A. Hutflesz, H.-W. Six and P. Widmayer, "The LSD-Tree: Spatial Access to Multidimensional Point and non-Point Objects," in *VLDB Conference*, Amsterdam, Netherlands, 1989.

[66] B. Seeger and H. Kriegel, "The Buddy Tree: An Efficient and Robust Access Methods for Spatial Database Systems," in *14th VLDB Conference*, 1988.

[67] T. Sellis, N. Roussopoulos and C. Faloutsos, "The R+-Tree: A Dynamic Index for Multi- Dimensional Objects," in *13th VLDB Conference*, Brighton, England, 1987.

[68] N. Beckmann, H. Kriegel, R. Schneider and B. Seeger., "The R*-Tree. An efficient and robust Access Method for Points and Rectangles," in *ACM SIGMOD*, 1990.

[69] H. Tropf and H. Herzog, "Multidimensional Range Search in Dynamically Balanced Trees," *Angewante Informatik*, 1983.

[70] J. A. Orenstein and T. Merret, "A Class of Data Structures for Associate Searching," in *ACM SIGMOD-PODS*, Portland, Oregon, 1984.

[71] H. Jagadish, "Linear Clustering of Objects with multiple Attributes," in *ACM SIGMOD*, 1990.

[72] D. Abel and J. Smith, "A Data Structure and Algorithm based on a linear Key for a Rectangle Retrieval Problem.," *Computer Vision*, vol. 24, pp. 1-13, 1983.

[73] C. Faloutsos and S. Roseman, "Fractals for Secondary Key Retrieval," in *8th ACM SIGMOD-PODS*, 1989.

[74] Y. Lum, "Multi-Attribute Retrieval with Combined indexes," *ACM*, vol. 13, no. 14, pp. 660-665, 1970.

[75] D. Haderle, Y. Wang and J. Cheng, "Single Table Access Using Multiple Indexes Optimization, Execution and Concurrency Control Techniques," in *International Conference on Extending Database Technology*,, 1990.

[76] P. O´Neill and D. Quass, "Improved Query Performance with Variant Indexes," in *ACM SIGMOD*, Tucson, Arizona, 1997.

[77] H. Sagan, "Space Filling Curves," Berlin/Heidelberg/New York, 1994.

[78] C. Faloutsos and Y. Rong, "Dot: A spatial access method using fractals," in *Seventh International Conference on Data Engineering, IEEE Computer Society*, Kobe, 1991.

[79] J. Lewder, "The Application of Space-the Storage and Retrieval of Multi-dimensional Data," London, 1999.

[80] D. Moore, "Hilbert Curve," 2000.

[81] "MISTRAL Project," 1999. [Online]. Available: http://mistral.informatik.tu-muenchen.de.

[82] V. Markl, "Processing relational queries using a multidimensional acces technique," *Dissertations in Database and Information Systems-Infix*,, vol. 59, 1999.

[83] P.Z. Piah, "An Efficient Range Searching Algorithm in Complex Geometric Space", PhD thesis, University of Port Harcourt, 2015.

[84] T. Skopala, M. Kratkyb, J. Pokornya and V. Snaselb, "A new range query algorithm for Universal B-trees," *Information Systems*, vol. 31, p. 489–511, 2006.