MAT JOURNALS

Journal of Web Development and Web Designing
Volume 3 Issue 1

A Survey on Spatial Indexing

Shaik Abdul Nusrath Begum, K. P. Supreethi
Department of Computer Science and Engineering
JNTUH College of Engineering Hyderabad (Autonomous)

Abstract

Spatial information processing has been a centre of attention of research in the previous decade. In spatial databases, data related with spatial coordinates and extents are retrieved based on spatial proximity. A large number of spatial indexes have been proposed to make ease of efficient indexing of spatial objects in large databases and spatial data retrieval. The goal of this paper is to review the advance techniques of the access methods. This paper tries to classify the existing multidimensional access methods, according to the types of indexing, and their performance over spatial queries. K-d trees out performs quad tress without requiring additional memory usage.

Keywords: *spatial data, spatial index, Object Directed Decomposition*

INTRODUCTION

The main purpose of spatial access methods is to support efficient selection of objects based on spatial properties. Spatial access methods are also used to implement efficiently map overlays and spatial joins.

Conventional databases are related to non-spatial data which are not referenced (directly or indirectly) to multidimensional space such as maps, geometric information etc. Generally, non-spatial data are one dimensional and independent. The multidimensional data cannot be stored in conventional database approaches. So the requirement of spatial database systems has emerged.

A spatial database system (SDS) [1] is a software system used for storage and retrieval of data and provides tools for referencing the spatial data (providing at least spatial indexing and spatial join methods).In spatial databases, data is associated with spatial coordinates and is retrieved based on spatial proximity.

A spatial data is a data that is present in 2-D or 3D or even a higher dimensional space. (Or) A spatial data object may be composed of a single point or several thousands of polygons, arbitrarily distributed across space. A spatial data has following characteristics [2]

- Complex structure
- Dynamic
- No standard algebra defined on spatial data
- Spatial operations are not closed
- Computational cost vary among spatial database operations (Expensive)

Spatial indexing and clustering methods must be taken in account. Without a spatial index, each object in the database has to be tested to tell whether it meets the spatial selection criterion; a 'full table scan' in a relational database. Full Table Scan is also known as Sequential Scan. In Full Table Scan the database is scanned where each row of the table is read in a sequential (serial) order and the columns encountered are tested for the validity of a condition.

As spatial datasets are typically very large, such checking is not agreeable in practice for interactive use and most other applications. Therefore, a spatial index is requirement to find the required objects

efficiently without considering every object. [3]
In some cases the total dataset fits in main memory, it is enough to know the address of the requested objects. As the main memory storage allows random access and does not have significant delays. However, most spatial datasets are so large that they cannot reside in the main memory of the computer and must be stored in secondary memory, such as its hard disk.

Clustering is desirable for group those objects which are often requested together. Otherwise, many different disk pages will have to be fetched which results in slow response. In a spatial context, clustering infers that objects which are closer in reality are also stored closer in memory. Many strategies for clustering objects in spatial databases adopt some form of 'SFC' by ordering objects according to their arrangement along a path that traverses all parts of the space.

Most of the indexes are based on the principle of divide and conquer. Indexing structures typically follow hierarchical approach. This approach is obviously suitable for a database system where the memory space is limited, and the pruning of a search is performed such that the more detail to be examined. Hierarchical structures are proficient in range searching.

Indexing in spatial database is different from conventional databases as the data stored in SDS is multi-dimensional objects and its associated coordinates. Search operation also varies as the spatial object properties (Geometry: location, size, shape. Spatial relationships: distribution, neighborhood, proximity) are considered not the attribute values.

Principles of Spatial Data Access and Search

The key principle leading the searching algorithm is dividing of the search space into regions. Considered simply, this consists of placing data into uniquely identifiable boxes or cells. These methods are categorized by considering spatial indexing because with each block the information is stored such that the block is occupied by the spatial object or part of the object is known.

Jones (1997) distinguishes two types of space decomposition or space partitioning: [4] [55] as shown in fig 1.

- Regular decomposition
- Object-directed decomposition.

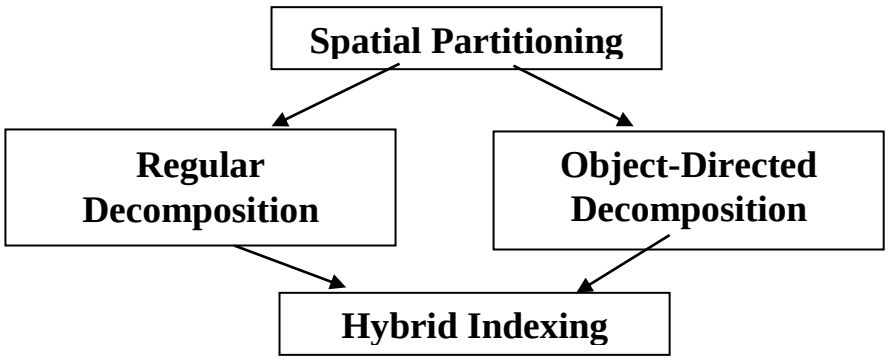

***Fig 1**: Space Decomposition*

REGULAR DECOMPOSITION

It is also known as space driven index. The space is divided in a regular or semi-regular manner independent of the distribution of objects (i.e., indirectly related to the objects in the space). Objects are mapped to the cells according to some geometric criterion.

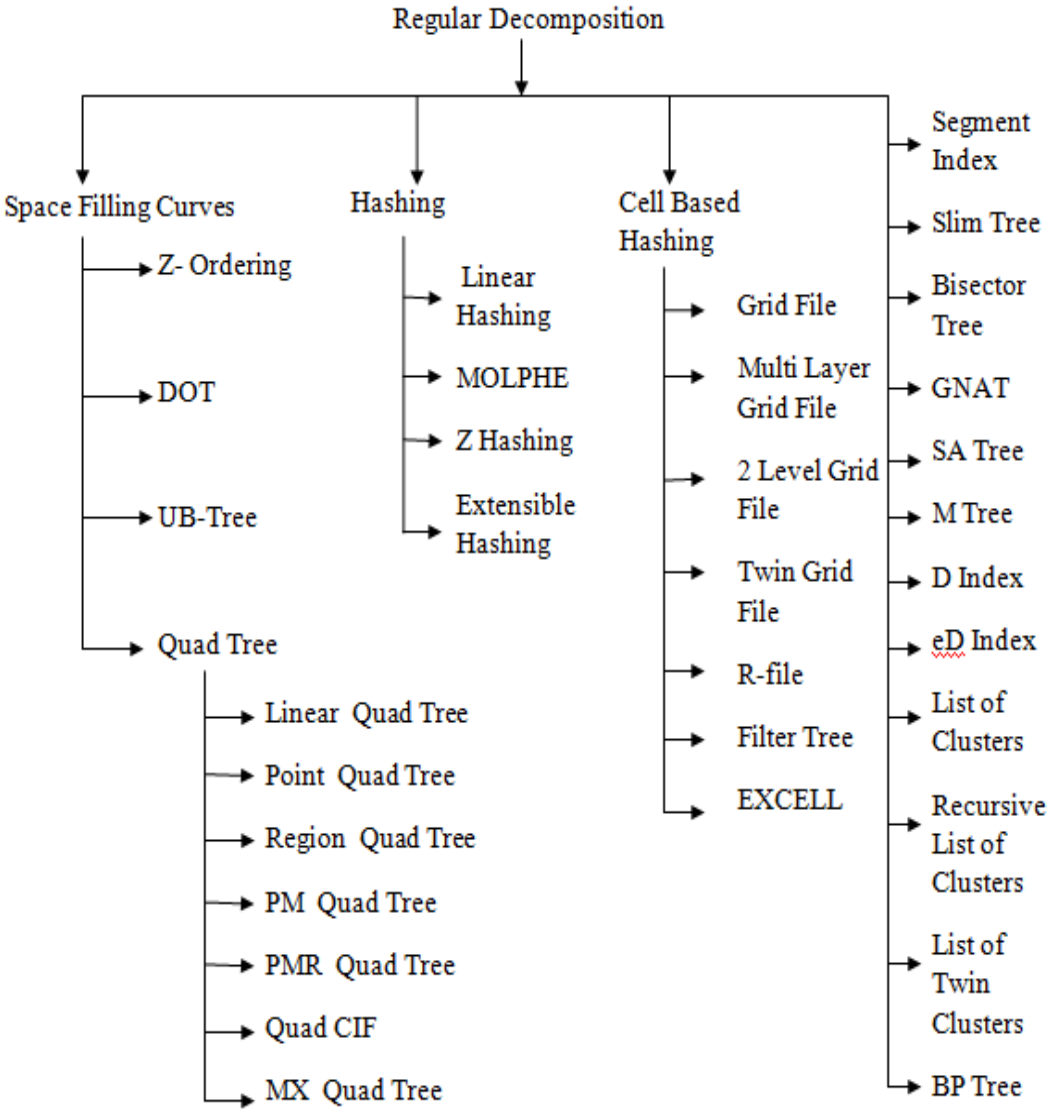

Fig 2: *Space Driven Index*

Object Directed Decomposition

Object Directed Decomposition is also known as data driven index. The division of the index space is determined directly by the objects. This technique divides the space by means of the coordinates of individual data points or of the extents or bounding rectangles or spatial objects which are to be stored. There is a multiplicity of object-directed decomposition search methods. The most common are:

- Binary tree
- R-tree

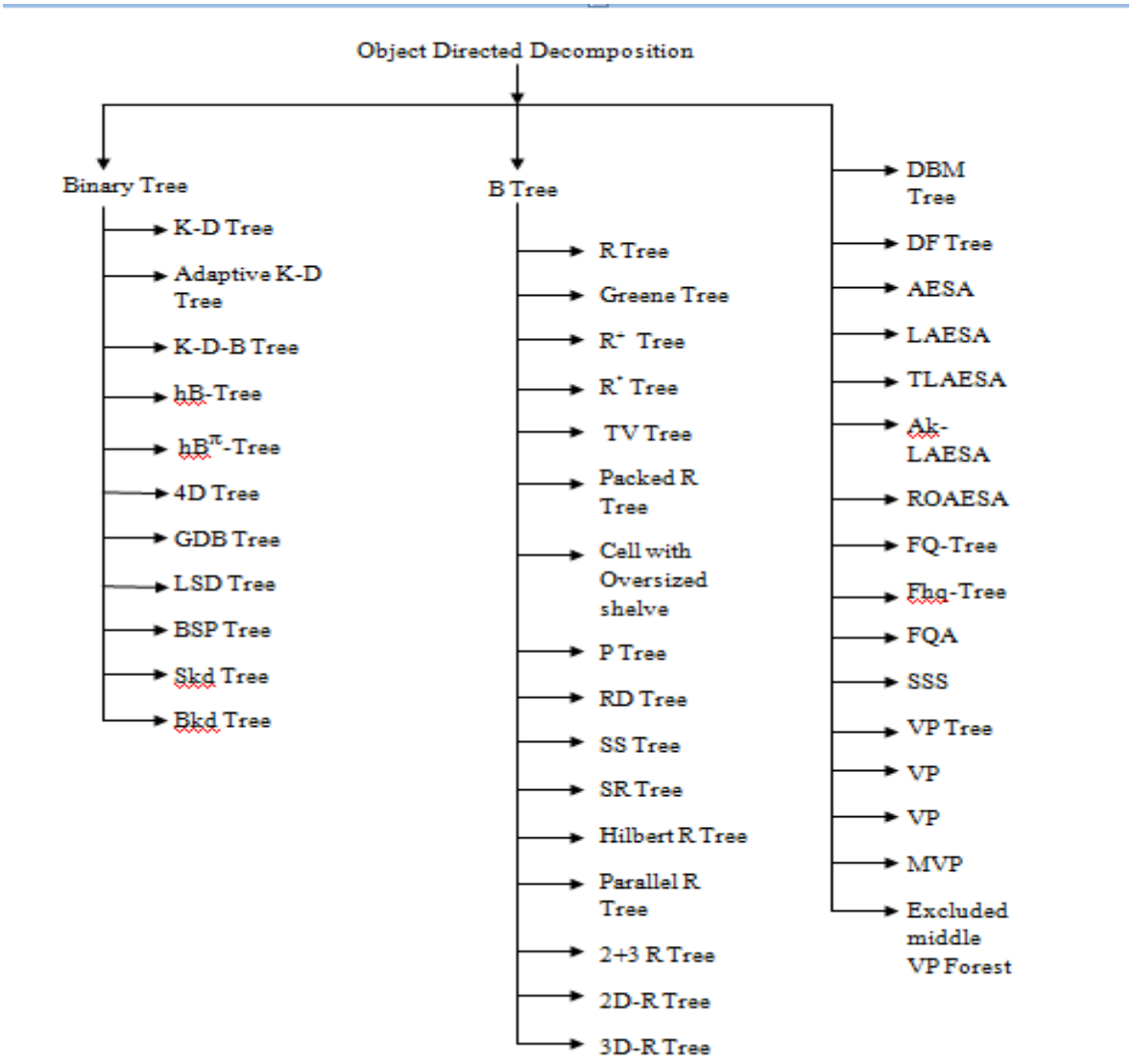

Fig 3: *Data Driven Index*

Object-Directed Decomposition (Data Driven Indexes)

Data Driven Indexing methods also known as Object-Directed Decomposition. Here, the objects decide the division of space (e.g. the 2D space containing the lines) into regions called buckets. They are commonly known as bucketing methods. There are some principal methods decomposing the space from which the data is drawn.

One approach to data driven indexes is for partition the data by relating divide-and-conquer strategies where individual data points or lines can be selected to subdivide the data space into consecutively smaller half-spaces (Binary-tree). Another method buckets the data based on the idea of a minimum bounding (or enclosing) rectangle (R-tree).

Such strategies generate hierarchical or tree data structures, in which traversing down each branch of the tree should result in reducing the volume of data at each stage. The branching factor defines the number of branches at each split and the number of leaves at the end of each branch.

Binary-Tree based Indexing Techniques
The binary search tree is a basic data structure for representing data items whose index values are organized by some linear order. The idea of repeatedly dividing the data space has been adopted and generalized in many sophisticated indexes.

K-D-Tree [5] [54] a binary search tree that stores points of the k-dimensional space. At each intermediate node, the k-d-tree divides the k-dimensional space in two parts by a (k-1)-dimensional hyperplane. The direction of the hyperplane, i.e. the dimension based on which the division is made, alternates between the k possibilities from one tree level to the next. Each splitting hyperplane contains at least one point, which is used as the hyperplanes representation in the tree.

Adaptive K-D-Tree [6] A better version of k-d tree is the *adaptive k-d-tree.* While dividing, the adaptive k-d-tree selects a hyperplane that divides the space in two sub-spaces with equivalent number of points. The hyperplanes are still parallel to axes, but they do not contain a point, and they do not have to firmly alternate. Interior nodes of the tree hold the dimension (e.g. x or y), and the coordinate of the corresponding split. All points are kept at the leaves, and a leave can contain up to a fixed number of points, if this number is exceeded, a split takes place.

K-D-B-Tree [7] pools properties of both the adaptive k-d-tree and the B-tree. It uses hyperplanes to split the space arbitrarily more than one hyperplanes divide a tree node (depending on the tree's storage utilization) in a equivalent number of disjoint regions. All nodes of the tree relate to disk pages. A leaf node stores the data points that are placed in the respective partition the leaf states. Like the B-tree, the K-D-B-Tree is perfectly balanced, however, it cannot guarantee storage utilization.

hB-Tree [8](holey brick B-tree)is a new multi-attribute index structure. It allows the data space to be holey, allowing removal of any data subspace from a data space. The concept of holey bricks has been used in as an effort to improve the clustering of data in a kd-tree known as the BD-tree. The hB-tree structure is based on the K-D-B-tree structure, but it allows the data space associated with a node to be non-rectangular and it uses kd-trees for space representation in its internal nodes. It is a height-balanced tree. Here, the leaf nodes are known as *data nodes* and the internal node are known as *index nodes*. An index node data space is a union of its child node subspaces which are obtained through kd-tree recursive division.

hB$^{\pi}$-Tree [9]The hB-tree is extended to allow for concurrency and recovery by transforming it in such a way that it becomes a special case of the π-tree. Consequently, the new structure is called the hB$^{\pi}$-tree. As a result of these modifications, the new structure can directly take advantage of the π-tree node consolidation algorithm. The lack of such an algorithm has been one of the major drawback of the hB-tree. Furthermore, the hB$^{\pi}$-tree corrects a flaw in the splitting/posting algorithm of the hB-tree that may occur for more than three index levels. The essential idea of the correction is to impose limitations on the splitting/ posting algorithms, which in turn affects the space occupancy.

4D-Tree [10] is used to store a bounding box by keeping the minimum and maximum points together in one 4D point. This technique can be used to simplify other geometric data structures that are originally suitable only for storing and retrieving points.

GBD-Tree [11] The BD-tree is prolonged to a balanced multi-way tree called the GBD-tree (generalized BD-tree). In addition to a DZ expression, a bounding rectangle is used to demonstrate data space that confines the objects whose centroids fall inside the region defined by the DZ expression. Centroids of objects are used to specify placement of objects in the correct bucket. While a DZ expression is used to confirm the position in the tree structure where an entity is found based on its centroid, a bounding rectangle is used in connection search.

LSD-tree [12] As an enhancement to the fixed size space partitioning of the grid files, a binary tree, called the Local Split Decision tree (LSD-tree) that supports arbitrary split position was proposed. A split position can be chosen such that it is optimal with respect to the current cell. The directory of an LSD-tree is similar to that maintained by the kd-tree. Each node of the LSD-tree represents one split and stores the split dimension and position, and each leaf node points to a data bucket.

BSP Tree [13] a data structure which is not built on a rectangular division of space. It uses the line segments of the polylines and the edges of the polygons to split the space in a recursive manner. The BSP-tree (Binary Space Partitioning) imitates this recursive division of space. Each time a (sub) space is divided into two subspaces by a so-called splitting primitive, a equivalent node is added to the tree. The BSP-tree implies an organization of space by a set of convex subspaces in a binary tree. This tree is beneficial during spatial search and other spatial operations.

Skd-Tree [14] Spatial kd-tree (skd-tree) in an attempt to avoid object duplication and object mapping. At each node of a kd-tree, a value (the discriminator value) is selected in one of the dimensions to partition a *k*-dimensional space into two subspaces. The skd-tree allows regions to overlap.

Bkd-tree [15] is an indexing technique for large multidimensional point data sets. The Bkd-tree is an I/O-efficient dynamic data structure based on the kd-tree. The Bkd-tree consists of a set of balanced kd-trees. Each kd-tree is laid out (or *blocked*) on disk similarly to the way the K-D-B-tree is laid out. To store a given kd-tree on disk, we first modify the leaves to hold points, instead of just one. In this way, points are packed in blocks.

B-tree based Indexing Techniques

B+-trees have been commonly used in data intensive systems to facilitate query retrieval. The widespread acceptance of the B +-tree is its height-balanced characteristic, making it best for disk I/O where data transfer is in the unit of page. It has become an underlying structure for many new indexes. In this section, we discuss indexes based on the concept of the hierarchical structure of B +-trees.

R-Trees [16] [54] [55] are hierarchical data structures, intended for efficient indexing of multidimensional objects with spatial extent. R-trees are used to store, rather than the original space objects, their *minimum boundary boxes* (MBBs). The MBB of an n-dimensional object is defined to be the minimum n-dimensional rectangle that contains the original object. Similar to B-trees, the R-trees are balanced and they ensure proficient storage utilization. Each R-tree node relates to a disk page and an n-dimensional rectangle. Each non-leaf node holds entries of the form (*ref, rect*), where *ref* refers to the address of a child node and *rect* refers to the MBB of all entries in that child node. Leaves hold entries of the same format, where *ref* points to a database object, and *rect* is the MBB of that object.

Greene's [17] The coverage of covering rectangles and overlaps between them in the R-tree are affected by the objects are being dividing into groups by its splitting algorithm. In Greene's splitting algorithm, two most distant rectangles are selected and for each dimension, the separation is calculated. Each separation is normalized by dividing it with the interval of the covering rectangle on the same dimension, instead of by the total width of the entire group. Along the dimension with the largest normalized separation, rectangles are ordered on the lower coordinate. The list is then divided into two groups, with the first (M+1)/2 rectangles into the first group and the rest into the other.

R^+-Tree [18] [55] is introduced as a way to overcome the problem of inefficient searching that arises when sibling nodes overlap in the R-tree. As a direct solution to these problems they use *clipping*, i.e. there is no overlap between in-between nodes of the tree at the same level, and objects that intersect more than one MBB at a specific level are clipped and stored on several different pages. As a result, point queries on the R+-tree require traversing only one path of the tree. The price to pay is the increase of storage requirement of the tree.

R*-Tree [19] [55] this variety presents a new insertion policy, that significantly improves the performance of the tree. The key objective of this policy is to minimize the overlap region among sibling nodes in the tree. A straightforward benefit of this is the minimization of the tree paths that are traversed at an object search.

TV Tree [20] [54] (Telescopic Vector) tree is dynamically contracting and spreading feature vectors. Like any other tree, it arranges the data in a hierarchical structure: Objects (i.e. feature vectors) are grouped into leaf nodes of the tree, and the explanation of their Minimum Bounding Region (MBR) is stored in its parent node. Parent nodes are recursively grouped too, until the root is formed.

Packed R-Tree [21] [55] Packed R-trees were introduced to minimize the coverage and overlap of rectangles by building an R-tree statically. It was shown that for point data, it is possible to divide points into groups such that the bounding rectangles of these groups do not overlap. However, to achieve zero overlap may require rotating the orientation of the entire database, which may not be possible or beneficial. Further, zero overlap is achievable only at the leaf level of the R-tree with static construction. For bounding rectangles associated with the non-leaf nodes, overlap is sometimes unavoidable. The main objective of the algorithm is to reduce the storage space, the coverage and overlap of rectangles, in order to improve the search efficiency.

Cell Tree with Oversized shelves [22] The fragmentation effect is even more serious when the database becomes more populated. Each split of a node leads to a decrease in the node data space but to an increase in the number of nodes per object. To overcome the fragmentation and duplication problems in cell tree, proposed to store oversized objects which may greatly increase the number of object identifiers being stored in the leaf nodes in separate "oversize shelves". These oversize shelves are data nodes related to interior nodes in the cell-tree, in one way, creating the tree to be not height-balanced. The assignment of a new object in the sub-tree or oversize shelf requires some optimization. The oversize page shelf can be spread out and a split on this shelf is essential.

P Tree [23] (J) In various applications, intervals are not a good estimate of the data objects enclosed. In order to combine the elasticity of polygon-shaped containers

with the ease of the R-tree autonomously proposed different variations of polyhedral trees or P-trees.

RD Tree [24] is a variant of R-Tree, a popular access method for spatial data. RD stands for "Russian Doll", which describes the transitive containment relation that is fundamental to the tree structure.

SS-Tree [25] is an index structure designed for similarity indexing of multidimensional point data. It is an enhancement of the R*-tree and improves the performance of nearest neighbor queries by altering the following respects. Initially, it employs bounding spheres rather than bounding rectangles for the region shape. Secondly, the SS-tree modifies the forced reinsertion mechanism of the R*- tree. When a node or a leaf is full, the R*- tree reinserts a portion of its entries rather than splits it, unless reinsertion has been made on the same tree level. On the other hand, the SS-tree reinserts entries unless reinsertion has been made at the same node or leaf. This promotes the dynamic reorganization of the tree structure.

SR-Tree [26] The structure of the SR-tree is based on that of the R-tree, in common with the R*-tree and the SS-tree, and corresponds to the nested hierarchy of regions. However, the distinctive feature of the SR-tree is that it specifies a region by the intersection of the bounding sphere and the bounding rectangle of underlying points. Incorporating bounding rectangles permits neighborhoods to be partitioned into smaller regions than the SS-tree and improves the disjointness among regions. This enhances the performance on nearest neighbor queries especially for high dimensional and non-uniform data which can be practical in actual image/video similarity indexing.

Hilbert R-Tree [27] uses the center point Hilbert value of the MBR to organize the objects. When grouping objects (based on their Hilbert value), they form an entry in their parent node which contains both the union of all MBRs of the objects and the largest Hilbert value of the objects. Again, this is repeated on the higher levels until a single root is obtained.

Parallel R-Tree [28] The underlying file structure is the R-tree. It is a server for spatial objects designed on a parallel architecture to achieve high throughput, under concurrent range queries. The first step is to decide on the hardware architecture. Then to distribute an R-tree over multiple disks.

2+3 R-Tree [29] The two-dimensional points would represent the current spatial information about the data points, whereas the three-dimensional lines would the represent (piecewise) the historical information. In the 2+3 R-tree, even though the end time of an object's position is unknown, it is indexed under a two-dimensional R- tree, keeping the start time of its position together with its id. Note that the original R- tree (or any of its derivatives) keep only the object's id (or a pointer to the actual data record) and its MBR in the leaf nodes. The two-dimensional R-tree used in this method is thus marginally altered.

2D R-Tree [30] The 2DR-tree is a height-balanced, hierarchical spatial data structure that uses two-dimensional nodes. An MBR is stored in an appropriate location with respect to all other MBRs in the node. Using two-dimensional nodes allows spatial relationships to be preserved. The spatial relationships supported in the 2DR-tree are north, northeast, east, southeast, south, southwest, west and northwest. A spatial relationship is defined between two objects using the centroids of their MBRs.

3D R-Tree [31] A spatiotemporal access structure. It uses standard R-trees to index multimedia data. It is widely used for indexing of spatial data in several applications, such as Geographic Information Systems (GIS), CAD and VLSI design, etc. Here R-trees are adapted in order to index either spatial, temporal or spatio-temporal occurrences of actors and relationships between them.

DBM-Tree (Density-Based Metric tree) [32] The DBM-tree is a dynamic MAM that grows bottom-up. The objects of the dataset are grouped into fixed size disk pages, each page corresponding to a tree node. An object can be stored at any level of the tree. Its main intent is to organize the objects in a hierarchical structure using a representative object as the center of each minimum bounding region that covers the objects in a sub-tree. An object can be stored in a node if the covering radius of the representative covers it.

DF-Tree [33] a new and efficient MAM. The main approach is to use global representatives for the whole tree. These global representatives will work together with the representatives of the nodes in order to decrease the number of distance calculations.

Approximating and Eliminating Search Algorithm [34] **(AESA)** makes use of metric properties of given distance. The algorithm consists of

- An efficient elimination rules and
- An appropriate search for available rules by which maximum efficiency is maintained.

For N prototypes, it uses pre-computation of triangular array of $(N^2-N)/2$ distances between prototypes. By which Nearest Neighbour (NN) Search is carried out through a very small number of distance computations. Furthermore, for large N, this number of computations tends to be independent of N (asymptotic constant time- complexity). However, the great storage complexity and preprocessing time ($O(N^2)$), severely limit the practical use of the AESA for large sets of prototypes.

LAESA [35] **(Linear AESA)** a new version of the AESA**,** which uses a linear array of distance, but is strictly based on metric arguments like the original AESA. The technique starts by choosing "Base Prototypes" (BP) from the given set of prototypes which are comparatively a small set there by calculating the distances between these BP's and the finishing set of prototypes.

Tree LAESA [36] **(TLAESA)** is based on the LAESA which reduces its average time complexity to sub-linear. The TLAESA algorithm consists of two parts: (1) preprocessing and (2) search. In the preprocessing algorithm, two structures are built: a binary search tree storing all prototypes and a table of distances. The search algorithm is essentially a traversal of the binary tree where the table of distances is used in order to avoid the exploration of some branches of the tree.

Approximating k-LAESA [37] **(Ak-LAESA)** is a fast classifier for general metric spaces (no vector space required) based on the LAESA algorithm The aim is to achieve classification rates similar to those of a k-NN classifier, using k neighbors that may not be the k-NNs and preserving the main properties of LAESA (distance computations and time and space complexities). It gets error rates very close to those of a k-NN classifier, while computing a much lower number of distances (the number of distances of the Ak-LAESA algorithm is accurately the same that the calculated by the LAESA algorithm).

ROAESA [38] (Reduced Overhead AESA), related to both AESA and LAESA, that decreases this overhead cost by using a heuristic to limit the set of objects whose lower-bound distances d_{lo} are updated at each step of the algorithm.

Fixed Queries Trees [39] **(FQ Trees)** are an evolution where the same pivot is used for all the nodes of the same level of the tree. In this case the pivot does not need to belong to the sub-tree. The main goal of this structure is to minimize the number of element comparisons, as opposed to other data structure operations (such as tree traversal). This is especially important in applications where computing distances between two elements is a much more expensive operation than, say following pointers. This tree structure differs from other tree structures where the keys on each level of the tree are all the same, so we have just one key per level. In other words, which comparisons we make does not depend on the results of previous comparisons up the tree. Thus the name fixed-queries tree or FQ-tree. A major strength of FQ- trees is that the data structure and the search algorithms are independent of the distance function, as long as it satisfies the triangle inequality.

Fixed Height fq-tree [40] **(fhq-tree)** A variant called FQ Tree where all the leaves are at the same depth h, regardless of the bucket size

Fixed Queries Array [41] **(FQA)** has two interesting properties. First, it is the first data structure which is able to achieve a sub linear (in the database size) number of side computations without using any extra space. Next, it is capable to trade number of pivots k for their precision, so as to enhance the usage of the available space. FQA are based in a common idea: k pivots are picked and each object is plotted to k coordinates which are its distances to the pivots. Later, the query q is also plotted and if it varies from an object in more than r along some coordinate then the element is filtered out by the triangle inequality. That is, if for some pivot p_i and some element v of the set it holds $|d(q, p_i) - d(v, p_i)| > r$, then we know that $d(q, v) > r$ without need to evaluate d(v, q). The elements that cannot be filtered out using this rule are directly equated.

Sparse Spatial Selection [42] **(SSS)** is a new pivot-based technique. The main characteristic of this method is that it guarantees a good pivot selection more efficiently. In addition, SSS adapts itself to the dimensionality of the metric space we are working with, without being necessary to specify in advance the number of pivots to use. Furthermore, SSS is dynamic, that is, it is capable to support object insertions in the database efficiently, it can work with both continuous and discrete distance functions, and it is suitable for secondary memory storage.

Vantage Point Tree [43] **(VP Tree)** Like the KD-Tree, each VP-Tree node cuts/divides the space. A VP-Tree node employs distance from a selected vantage point rather than using coordinate values. Near points make up the left/inside subspace while the right/outside subspace consists of far points. A binary tree is formed by recursion. Each of the tree nodes identifies a vantage point, and for its children (left/right), the node contains bounds associated to subspace by the vantage point. Other forms of the VP-Tree include additional subspace bounds and may employ buckets near leaf level.

VP^S-Tree [43] an element of S is compared with the vantage point belonging to each of its ancestors in the tree. This information is also not captured in the simple tree. It consists of a database element identifier 'id', and a list 'hist' of distances from the item to each vantage point tracing back to the root. A list of these structures is

initialized with the history set to null from the entire database. The algorithm then recursively splits the list into two lists L and R, while adding to the history of each item. Each resulting tree node contains a list 'bnds' which have a range interval for its corresponding subspace which is seen by each ancestral vantage point.

VP^{SB}-Tree [43] Here we consider one way to form buckets of leaves in order to save space while preserving the notion of ancestral history present in the VP^{S}-Tree. Buckets are formed by descending sub-trees near leaf level into a flat structure. Each bucket contains say no element records. Each record must specify an id, and in addition holds distance entries for every ancestor. We call the resulting structure a VP^{SB}-Tree.

Multi-Vantage Point Tree [44] **(MVP Tree)** A distance based index structure called multi-vantage point (MVP) tree for similarity queries on high-dimensional metric spaces. The MVP Tree uses more than one vantage point to partition the space into spherical cuts at each level. It also utilizes the pre-computed (at construction time) distances between the data points and the vantage points.

Excluded Middle Vantage Point Forest [45] is a variant of VP-trees. It is intended for radius-limited nearest neighbor search, that is, where the nearest neighbor is restricted to be within some radius *r** of the query object. This technique is based on the insight that most of the complexity in performing search in methods is based on binary partitioning, such as the VP-tree, is due to query objects that lie close to the partition values, thus causing both partitions to be processed.

Regular Decomposition (Space Driven Indexes)

Applying the regular decomposition methods, the data space is partitioned in a regular or semi-regular way. The subdivision of space should be specified and then object will be addressed in the new structure. The geometry of the object is distributed between several adjacent cells (or regions). The objects details are generally kept intact, whereas the spatial index cells store locations of the database positions of the entire objects that intersect them. The data related with each cell are stored in one or more records and the address of which is given in terms of the coordinates of the lower corner of the cell.

For the regular decomposition of space, cells commonly have three different shapes:

- **Triangle:** convenient for representing approximately spherical surfaces. Triangles have the advantage that they can be regularly partitioned any number of times.
- **Rectangle**: most appropriate because its edges can be aligned with the axis of a coordinate system. Rectangles simplify inclusion analysis within rectangular search window.
- **Hexagon:** suitable for mapping statistical properties since their neighboring centers are equidistant in all six directions.

Space Filling Curves [3] order the points in a discrete two-dimensional space. This method is also called tile indexing. It converts a two-dimensional problem into a one-dimensional one, so it can be used in combination with a familiar data structure for one-dimensional storage and retrieval.

- The **row ordering** simply numbers the cells row by row, and within each row the points are numbered from left to right
- The **row prime** (or snake-like, or

boustrophedon) ordering is a variant in which alternate rows are traversed in opposite directions

- **Column and column prime orderings** in which the roles of row and column are transposed.
- Bitwise interleaving of the two coordinates results in a one-dimensional key, called the **Morton key**. The Morton key is also known as Peano key, or N- order, or Z-order.
- **Hilbert ordering** is constructed using the classic Hilbert-Peano curve.
- **Gray ordering** is achieved by bitwise interleaving the Gray-codes of the x and y coordinates. As Gray codes have the property that successive codes vary by exactly one bit position, a 4-neighbour cell only differs in one bit
- The **Cantor-diagonal ordering** is that the numbering of the points is adapted to the fact that we are dealing with a space that is bounded in all directions
- The **Spiral ordering**
- The **Sierpinski curve**, which is based on a recursive triangle subdivision.

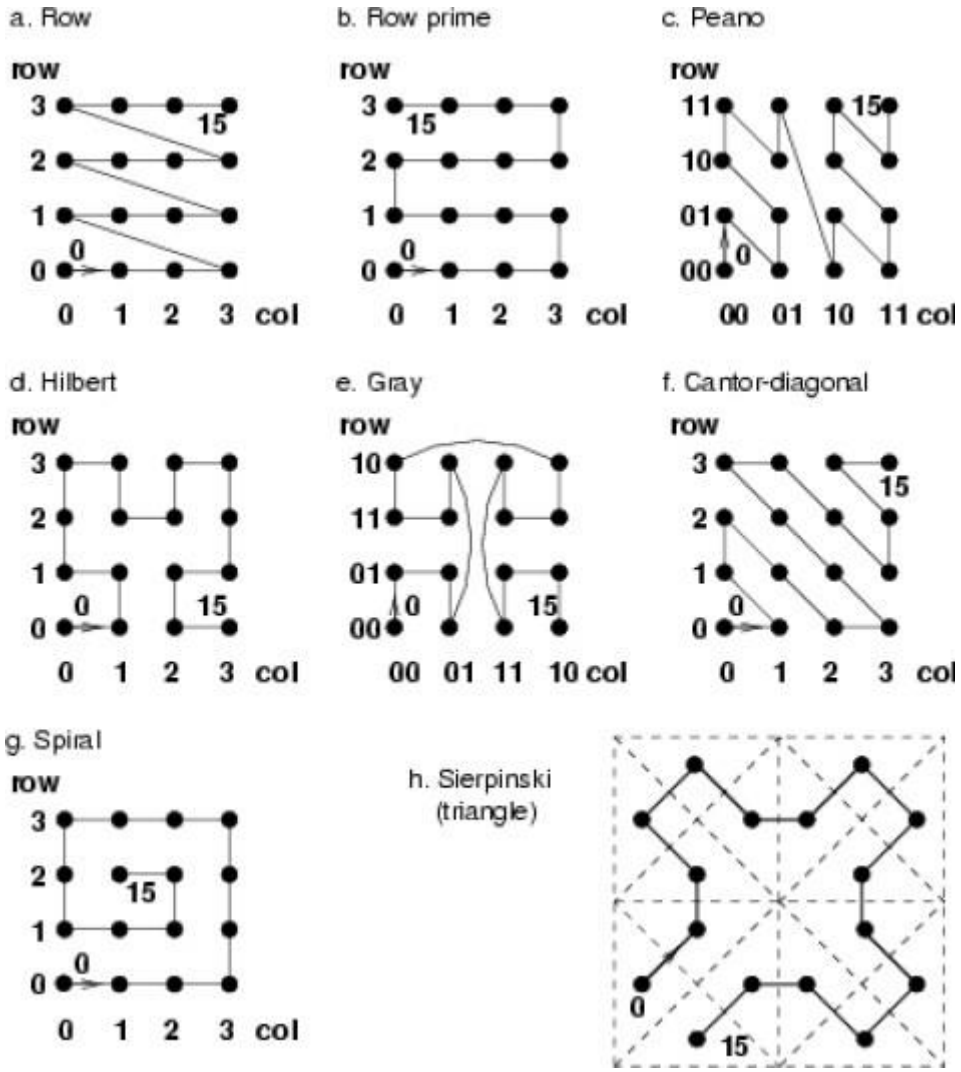

Fig 4: *Space Filling Curves*

DOT [46] (DOuble Transformation) approach has two transformations: first, a covering rectangle of an object in a *k*-dimensional space is mapped into a point in a *nk*-dimensional space; second, the n*k*-dimensional point is then mapped onto a point in a one-dimensional space using the distance preserving mapping concept. The second transformation can be any mapping, such as Hilbert or Peano mapping.

UB Tree [47] is based on the grouping of one dimensional index structures and space-filling curves. i.e., combines the B-Tree and the Z-curve. Together with its sophisticated query processing algorithms it has proven its performance advantages in numerous application domains. As the UB-Tree is based on the standard B-Tree, which is the basic index structure in almost every commercial DBMS, the task of integrating this MAM into an existing kernel becomes less complex and less costly.

Z-ordering [48] [55] is based on the Peano curve. A simple algorithm to obtain the z-ordering representation of a given extended object can be described as follows. Starting from the (fixed) universe containing the data object, space is split recursively into two subspaces of equal size by (*d*-1)-dimensional hyperplanes.

Quad-tree Based Structures

Quadtrees [49] are one of the first data structures for higher-dimensional data. A quadtree is a rooted tree in which every internal node has four children. Every node in the quadtree corresponds to a square. If a node *v* has children, then their corresponding squares are the four quadrants of the square of *v* ¾ hence the name of the tree. This infers that the squares of the leaves together form a portion of the square of the root. We call this subdivision as the *quadtree subdivision*. The children of the root are labeled NE, NW, SW, and SE to indicate to which quadrant they belong to; NE stands for the north-east quadrant, NW for the north-west quadrant, and so on.

Point Quadtree [49] [54] resembles the KD-tree. The modification is that the space is divided into four rectangles instead of two. The input points are stored in the internal nodes of the tree.

Region Quadtree [50] is used to store a rasterized estimate of a polygon. First, the area of interest is bounded by a square. A square is repetitively divided into four squares of equal size until it is entirely inside or outside the polygon or until the maximum depth of the tree is reached.

PM Quadtree [51] A polygonal map, a collection of polygons, can be represented by the PM Quadtree. The vertices are stored in the tree similar as in the PR Quadtree. The edges are divided into q-edges which entirely fall within the squares of the leaves. There are seven classes of q-edges. The first class of q-edges is those that interconnect one boundary of the square and meet at a vertex within that square. The other six classes intersect two boundaries and are named after the borders they intersect: NW, NS, NE, EW, SW and SE. For each non-empty class, the q-edges are stored in a balanced binary tree.

PMR Quadtree [52] (for PM Random) is based on the observation that any rule that divides up the line segments among quadtree blocks in a reasonably uniform fashion can be used as the basis for a PM-like quadtree. The PMR quadtree uses a couple of rules, one for splitting and one for merging, to dynamically organize the data.

Quad-CIF-Tree [53] (CIF - Caltech Intermediate Form) was proposed for representing a set of small rectangles for VLSI applications. It is organized in a way

similar to the region quad-tree. A region is recursively partitioned until the resulting quadrants do not contain any rectangle. During the subdivision, all rectangles that intersect with either of the two dividing lines are associated with the dividing lines. The rectangles that are associated with a quadrant must not belong to any ancestor quadrant. It is assumed that no two rectangles overlap.

MX Quadtree [54] is probably the simplest way of representing line data, and is a region quadtree in which lines are represented by regions which are one pixel wide. It can be viewed as a quadtree representation of a chain code.

Linear QuadTree [55] The key remark is that once the entries [*mbb*, *oid*] has been assigned (as for a regular quadtree) to a quadtree leaf with label *l* , and stored in a page with address *p*, then we index in a B+ tree the collection of pairs (*l* , *p*) keyed on the leaf label *l*.Such a organization provides an ice packing of quadtree labels into B+ tree leaves. The packing is dynamic; that is, it persists when inserting and deleting objects in the collection.

Hashing based Structures

Linear Hashing[56][57] Linear hashing divides the universe [A, B) of possible hash values into binary intervals of size $(B-A)/2^k$or $(B-A)/2^{k+1}$ for some $k \geq 0$.Each interval corresponds to a bucket, that is, a collection of records stored on a disk page. T € [A, B) is a pointer that separates the smaller intervals from the larger ones: all intervals of size $(B-A)/2^k$are to the left of t and all intervals of size $(B-A)/2^{k+1}$ are to the right of t.

Multidimensional Linear Hashing. Unlike multidimensional extendible hashing, multidimensional linear hashing uses no or only a very small directory. It therefore occupies relatively little storage compared to extendible hashing, and it is usually possible to keep all relevant information in main memory.

MOLHPE [58] A variation of linear hashing called Multidimensional Order-Preserving Linear Hashing with Partial Expansions (MOLHPE). This organization is based on the idea of partially extending the buckets without increasing the file size at the same time. MOLHPE outperforms for uniformly distributed data. It fails for non-uniform distributions, mostly because the hashing function does not adapt elegantly to the given distribution.

Z-Hashing [59] uses a space-filling curve technique called z-ordering to guarantee that points located close to each other are also stored close together on the disk.

Extendible Hashing [60] As does linear hashing, extendible hashing organizes the data in binary intervals, here called cells. Overflow pages are avoided in extendible hashing by using a central directory. Each cell has an index entry in that directory; it firstly corresponds to one bucket. If during an insertion a bucket at maximal depth exceeds its maximum capacity, all cells are split into two. New index entries are created and the directory doubles in size. Since each bucket was not at full capacity before the split, it may now be possible to fit more than one cell in the same bucket. In that case, adjacent cells are regrouped in data regions and stored on the same disk page.

Cell Methods based on Dynamic Hashing

Grid Files [61] [55] A file data structure aimed to manage a disk allocation storage in terms of fixed size units like disk blocks, pages or buckets depending on level of description. It is a variant of grid method where the requirement is cell

division lines must be equidistant. Hashing method for multidimensional points. It is an extension of extensible hashing.

Multi-layer Grid File [62] To enhance the search performance of the grid file, a *multi-layer grid file* which neglects object mapping was proposed. In such a structure, a map space may consist of several grid files that cover the same space. When a grid file is divided, all objects that are not cut by the dividing hyperplane are scattered among the two new subspaces and objects that are cut are stored in the next layer grid file. There may be several layers of grid files that store un-partitioned objects, and each layer has different dividing hyperplanes. At the maximal layer, the objects are clipped if they overlap the dividing hyperplane.

Two-Level Grid File [63] The basic idea of it is to use a second grid file to manage the grid directory. The first level is called the *root directory*. Entries of the root directory contains pointers to the directory pages of the lower level, which in turn contain pointers to the data pages. By having a second level, splits are often limited to the subdirectory regions without affecting too much of their surroundings.

Twin Grid File [64] It is other hashing method. It tries to increase space utilization compared to the original grid file by introducing a second grid file. The relationship between these two grid files is not hierarchical but somewhat more balanced. Both grid files span the whole space (universe). The distribution of the data among the two files is performed dynamically.

R File [65] The R-file is based on the concept of multi-layer grid files. The R-file is different from the multi-layer grid file in that the R-file has only one layer and is intended for non-zero sized objects. In the R-file, cells are divided using the dividing strategy of the grid file and a cell is split when overflowed. In order for cells to robustly contain the spatial objects, cells are divided recursively by repeated halving till the smallest cell that encloses the spatial objects is obtained. Spatial objects that are completely contained in a cell are stored in its related data page, and those that intersect the dividing line are stored in the original cell. If the number of spatial objects that intersect a division is more than what can be stored in a data page, partitioning line along the other dimensions will be used. If all records lie on the cross point of partitioning lines, they cannot be partitioned by any partitioning lines, and in such a case a chain of buckets is used.

Filter Tree [66] is a hierarchical organization that tends to separate spatial entities by size, placing larger entities at the higher levels of the Filter Tree, and smaller entities at lower levels. Within each level, index entries for the entities are ordered by a space-filling curve (Hilbert curve). This allows the algorithms to use bulk I/O requests, exploiting the locality in the index information, and minimizing the number of I/O transfers from disk. Filter Trees engage a recursive binary partition of the data space in each dimension. Entities related with a particular level are all grouped together. Each entity is placed at the lowest-level of the tree at which it is entirely enclosed by a single cell of the division at that level. This method of determining the level at which an entity is stored tend to cause larger entities to be stored high in the tree (because they can be enclosed only in large cells), whereas smaller entities tend to sink to lower levels of the tree since they fit into smaller cells. Sometimes small entities will be caught at higher levels in the tree because they happen to lie across the boundary between two large cells. However, under reasonable statistical assumptions about where entities are placed, the fraction of such entities is

small.

EXCELL method [67] (Extendible CELL) related to the grid file. It is a bintree with a directory in the form of an array providing access by address computation. It can also be viewed as an adaptation of extendible hashing to multidimensional point data. In contrast to the grid file, where the dividing hyperplanes may be spaced arbitrarily, the EXCELL method decomposes the space regularly ad all grid cells are of equal size.

Segment Index [68] approach combines features of the memory resident Segment Tree data structure with those of a class of database access methods that are based on paged, multi-way, tree-structured indexes. The Segment Tree data structure stores line segments in a binary tree by storing the segment endpoints in the leaf nodes, and then associates each interval with the highest level no& N that spans the values corresponding to the left and right children of N.

Slim-tree [69] a dynamic tree for organizing metric datasets in pages of fixed size. The Slim-tree uses the "fat-factor" which provides a simple way to quantify the degree of overlap between the nodes in a metric tree. It is well-known that the degree of overlap directly affects the query performance of index structures. There are many suggestions to reduce overlap in multi- dimensional index structures, but the Slim- tree is the first metric structure explicitly designed to reduce the degree ofoverlap.

Bisector Tree [38][70] (BST) it is often common to augment the gh-tree by including for each pivot the maximum distance to an object in its sub-tree yielding what are, in effect, *covering balls*. The resulting data structure is called a bisector tree (BST). The motivation for adding the covering balls is to speed up the search by enabling the pruning of elements whose covering balls are farther from the query object than the current candidate nearest neighbor (the farthest of the *k* candidate nearest neighbors) or are outside the range for a range query.

GNAT [71] (Geometric Near-neighbor Access Tree) is a generalization of the GH- Tree, where more than two pivots may be chosen to partition the data set at each node. In particular, given a set of pivots *P= {p1, ..., pm}*, we split *S* into *S*1, *: : : , Sm* based on which of the objects in *P* is the closest. It is also based on Voronoi cell- like partitioning.

SA-Tree [72] (*Spatial Approximation Tree*.) was inspired by the Voronoi diagram, a widely used method for nearest neighbor search in point data. thesa-tree attempts to approximate the structure of the Delaunay graph.

M-tree [38] [73] [74] is based on a hierarchical organization of data objects Oi € S according to a given metric d. It is a distance-based indexing method designed to address this deficiency. Like other dynamic and paged trees, the M-tree structure consists of a balanced hierarchy of nodes. The nodes have a fixed capacity and a utilization threshold. Within M-tree hierarchy the objects are clustered into metric regions. The leaf nodes contain ground entries of the indexed data objects while routing entries (stored in the inner nodes) describe the metric regions. Its design goal was to combine a dynamic, balanced index structure similar to the R-tree (which, in turn, was inspired by the B-tree) with the capabilities of static distance-based indexes.

D-Index [75] is an access organization for similarity search. It is a multi-level metric structure, consist of search-separable buckets at each level. The structure supports simple insertion and restricted search costs for the reason that at most one

bucket desires to be accessed at each level for range queries up to a predefined value of search radius ρ. At the same time, the applied pivot-based approach considerably reduces the number of distance computation in accessed buckets.

eD-Index [76] An access structure for similarity self-join**.** The idea behind the eD- Index is to change the ρ-split function so that the exclusion set and separable sets overlap of distance. The objects which belong to both the separable and the exclusion sets are replicated. This principle, called the *exclusion set overloading*, ensures that there always exists a bucket for every qualifying pair $(x, y)|d(x, y) \leq \mu \leq €$where the pair occurs.

List of clusters [77] The LC splits the space into zones. Each zone has a center *c* and stores both its radius *rp* and the bucket *I* of internal objects, that is, the objects inside the zone. The LC splits the space into zones (or "clusters"). Each zone has a center c and a radius r_c and it stores the internal objects $I = \{x \in S, d(x, c) \leq r_c\}$, which are at distance at most r_c from c.

Recursive List of Clusters (RLC) [78] [79] which can be seen as a dynamic version of the LC. The RLC is composed by clusters of fixed radius, so the number of objects of each cluster can differ.

List of Twin Clusters (LTC) [80] a new metric index specially focused on the similarity join problem. The data structure considers two lists of overlapping clusters, which we can call twin clusters. Each cluster is a triple (center, effective radius, internal bucket). Considering the LC idea, every object being a center is not included in its twin bucket. So, when solving range queries, most of the relevant objects would belong to the twin cluster of the object that we are querying for.

Ball-and-Plane tree (BP-tree) [81] which is constructed by separating the dataset into compact clusters. It combines the advantages of both disjoint and non-disjoint paradigms in order to attain a structure of tight and low overlapping clusters, yielding considerably better performance. BP-tree does not split the data set into disjoint or non-disjoint groups. Instead, it is an index structure that combines the advantages of both those strategies. BP-tree is an unbalanced tree index generated by the hierarchical partitioning of the dataset. Like other metric trees, the objects of the data set are stored into fixed size disk pages. Each page holds a predefined maximum number of objects *K*.

Hybrid methods

Buddy-Tree [82] The buddy-tree can be measured as a compromise of the R-tree and the grid-file. It ignores the down splitting of the K-D-B-tree, the overlap problem of the R-tree and the dependency of structure upon the inclusion of data. The buddy-tree generalize the buddy system of the grid-file to arrange correlated data proficiently, by bounding the data points firmly using the bounding rectangle concept of the R-tree and arrange the directory as in the R-tree. Like grid-files, the non-zero sized data have to be mapped into high dimensions.

BANG File [83] (Balanced And Nested Grid) file is an interpolation-based grid file which is however different from the original grid file in that it allows two subspaces to intersect. The BANG file divides the data space into a hierarchy of sets of notational grid regions. Each of these grid regions can be identified by a unique pair (r, l), where *r* is the region number, and *l* is the granularity or level number. A space is obtained by recursively halving along some selected dimensions. That is, the level of the hierarchy of grid

regions is generated from the previous higher level by dividing along a selected dimension.

BV-Tree [84] The BV-tree represents an effort to solve the *d*-dimensional B-tree problem, i.e.,, to find a generic generalization of the B-tree to higher dimensions. The BV-tree is not intended to be a concrete access method, but rather a conceptual framework that can be useful to a variety of existing access methods, including the BANG file or the hB-tree.

G-tree [85] (grid tree) is based on the BD-tree. The structure differs from the BD-tree in the way the partitions are mapped into buckets. To obtain a simple mapping, the G-tree sacrifices the minimum storage utilization that holds for the BD-tree.

Generalized Grid File [86] **(GGF)** It is a multi-level grid file. The GGF acts like a B+ tree for a single dimensional data. With such a hierarchical formation, the property of "two disk accesses" for exact-match queries is no longer applicable.

Pivoting M-tree (PM-tree) [87], exploiting pivot-based ideas for metric region volume reduction.

Metric Index (M-Index) [88] defines a worldwide mapping schema from a generic metric space to a real numeric domain. Significantly, this schema has the capability to conserve the proximity of data, i.e. it map similar metric objects to close numbers in the numeric domain. The M-Index indexing and searching mechanisms make use of a set of reference objects and synergically exploit basically all known metric based principles of data separation, pruning and filtering.

PLOP Hashing [89] *(Piecewise linear order-preserving*) hashing this structure can also be used as an access method for extended objects. A grid file extension was proposed for the storing of non-zero sized objects. The method is a multi-dimensional dynamic hashing scheme based on Piecewise Linear Order Preserving (PLOP) hashing. Like the grid file, the data space is partitioned by an orthogonal grid. However, instead of using *k* arrays to store scales that define dividing hyperplanes, *k* binary trees are to represent the linear scales. Each internal node of a binary tree stores a (*k*-1) dimensional partitioned hyperplane. Each leaf node of a binary tree is related with a *k*-dimensional subspace (a slice), where the interval along its related axis is a sub-interval and the other *k*-1 intervals assume the intervals of the global space. Each slice is addressed by an index *i* stored in its leaf node.

Space-filling curve and Pivot-based B+-tree (SPB-tree) [90] It stores complex objects in a separate random access file (RAF), and uses a B+-tree with minimum bounding boxes (MBB) to index objects after a two-stage pivot-and-SFC mapping. The SPB-tree is generic: it does not rely on the detailed representations of objects, and it can support any distance notion that satisfies the triangle inequality.

Other kinds of indexing

Bitmap Indices [91] [92] [93] [94] [95] Bitmap indices are a specialized type of index designed for easy querying on multiple keys, although each bitmap index is built on a single key. A bitmap index for a field F is a collection of bit-vectors of length n, one for each possible value that may appear in the field F. The vector for value v has 1 in position i if the i^{th} record has v in field F, and it has 0 there if not. For bitmap indices to be used, records in a relation must be numbered sequentially, starting from, say, 0. Given a number n, it must be easy to retrieve the record numbered n. This is particularly easy to achieve if records are fixed in size, and allocated on consecutive blocks of a file.

The record number can then be translated easily into a block number and a number that identifies the record within the block.

Join Index [96] [97] [98] a simple data structure for improving the performance of joins in the context of complex queries. A Join Index is a data structure used for processing join queries in databases. Join indices uses pre-computation techniques to speed up online query processing and are useful for datasets which are updated rarely. For most of the joins, updates to join indices incur very little overhead. Some properties of a join index are

- Efficient usage of memory and adaptiveness to parallel execution
- Compatibility with other operations (including select and union)
- Support for abstract data type join predicates
- Support for multi-relation clustering and
- Its use in representing directed graphs and in evaluating recursive queries.

STARjoin & STARindex [99] STARjoin is a high-speed, single pass, parallelizable multi-table join. It outperforms many join methods implemented by traditional OLTP RDBMSs as it can join more than two tables in a single operation. Red bricks RDBMS supports the formation of specialized indexes called STARindexes, to significantly accelerate join performance. The STARindexes differs from traditional index structures like B-tree or bitmapped indexes. STARindexes are formed on one or more foreign key columns of a fact table.

OBSERVATIONS

Some of the observations are as follows:

- R^+-tree > R-tree > k-d-B-tree When less overlap between data rectangles
- R^*-tree > Variants of the R-tree. R^*-tree has best storage utilization and insertion times. For all data list and queries, only number of disk accesses is measured.
- Hilbert R-tree slightly better than R^*-tree
  - ❖ Hilbert codes can therefore be used for bulk insertion into dynamic R^*- tree.
  - ❖ Hilbert R-tree has better search result, while updates take about the same as for the R^*- tree.
- skd-tree > R-tree skd-tree requires more space than R-tree.
  - ❖ For large page size, the performance is in term of number of page accesses per search operation.
- PMR-quadtree = R^*-tree = R^+-tree
  - ❖ R^+- tree shows the best insertion Performance.
  - ❖ R^*- tree occupies the least space and is more compact in term of the data.
  - ❖ When use line segments as test data for indexing.
- R-file > R-tree
  - ❖ R-file has a 10-20% performance advantage over the R-tree on a data set with a high degree of overlap.
- K-d trees out performs quad tress without requiring additional memory usage.
- (Buddy tree, BANG file) > R-tree
  - ❖ For all data distributions in terms of measuring the number of page accesses.
- SPB Tree achieves low-cost index storage, construction, and manipulation, supports efficient query processing in metric spaces and manage efficiently a large set of complex objects.

CONCLUSION

In this paper we presented a short overview of the current state in the field of development of the access methods. During the last four decades the access methods have been developed towards plenty of modifications of small number

basic ideas. It is important to remark that the research has been provided on software as well as on hardware levels.

The survey of the access methods suggests that the context-free multi-dimensional access methods practically are not available. SPB tree outperforms in terms of low cost indexing and efficient query processing. K-d trees out performs quad tress without requiring additional memory usage.

REFERENCES

1. Indexing in Spatial Databases BC Ooi, R Sacks-Davis, J Han-Unpublished/Technical Papers, 1993.
2. Multidimensional Access Methods , VOLKER GAEDEIC-Parc, Imperial College, London and OLIVER GU¨ NTHER Humboldt-Universita¨ t, Berlin, ACM Computing Surveys, Vol. 30, No. 2, June 1998.
3. Spatial access methods P VAN OOSTEROM Geographical information systems, 1999.
4. A white paper on Spatial Partitioning and Indexing, Claudia Dolci, Dante Salvini, Michael Schrattner, Robert Weibel, GITTA(Geographic Information Technology Training Alliance).
5. Bentley, J. L., "Multidimensional Binary Search Trees Used For Associative Searching, "*Communications of the ACM*, 18(9), 509-517, 1975.
6. J. L. Bentley, J. H. Friedman. *Data structures for range searching*. ACM Comput. Surv. 11, 1979, 4, 397–409.
7. J.T. Robinson. The K-D-B-tree: A search structure for large multidimensional dynamic indexes. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 1981, pp. 10-18.
8. D.B. Lomet, B. Salzberg. *The hBtree: A robust multi attribute search structure*. In Proceedings of the Fifth IEEE International Conference on Data Engineering, 1989,pp. 296–304.
9. G. Evangelidis, D. Lomet, B. Salzberg. *The hBP- tree: A modified hB-tree supporting concurrency, recovery and node consolidation*. In Proceedings of the 21st InternationalConference on Very Large Data Bases, 1995, pp. 551–561.
10. J. B. Rosenberg: Geographical data structures compared: A study of data structures supporting region queries, *IEEE Trans. on Comp. Aided Design CAD-4*, 1, 53-67 (1985).
11. Y. Ohsawa, M. Sakauchi: A new tree type data structure with homogeneous nodes suitable for a very large spatial database. *Proc. IEEE 6th Int. Conf. on Data Engineering*, 296-303 (1990).
12. A. Henrich, H.-W. Six, P. Widmayer. *The LSD tree: Spatial access to multidimensional point and non-point objects*. In Proceedings of the Fifteenth International Conference on Very Large Data Bases, 1989, pp. 45–53.
13. H. Fuchs, Z. Kedem, B. Naylor. *On visible surface generation by a priori tree structures*. Computer Graph. 14, 3, 1980.
14. B.C. Ooi, K.J. McDonell, R. Sacks-Davis. Spatial kd-tree: An indexing mechanism for spatial databases. In Proceedings of the IEEE Computer Software and Applications Conference, 1987, pp. 433–438.
15. O. Procopiuc, P. K. Agarwal, L. Arge, J.-S. Vitter. Bkd-tree: A Dynamic Scalable kd-tree. In

Proceedings of International Symposium on Spatial and Temporal Databases, 2003.
16. A. Guttman: R-trees: A dynamic index structure for spatial searching. *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Boston, MA, 47-57 (1984).
17. D. Greene: An implementation and performance analysis of spatial data access methods. *Proc. 5th Int. Conf. on Data Engineering*, 606 – 615 (1989).
18. Sellis et al, 1987] T. Sellis, N. Roussopoulos, C. Faloutsos. The R+-tree: A dynamic index for multi-dimensional objects. In Proceedings of the Thirteenth International Conference on Very Large Data Bases, 1987, pp. 507–518.
19. N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In Proceedings of ACM SIGMOD International Conference on Management of Data, 1990, pp. 322–331.
20. K. I. Lin, H. V. Jagadish, C. Faloutsos. The tv-tree: an index structure for high dimensional data. The VLDB Journal, 3(4):517–542, 1994.
21. N. Roussopoulos, D. Leifker. Direct spatial search on pictorial databases using packed R-trees. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 1985, pp. 17–31.
22. O. Günther, H. Noltemeier. Spatial database indices for large extended objects. In Proceedings of the Seventh IEEE International Conference on Data Engineering, 1991, 520–526.
23. H.V. Jagadish. Spatial search with polyhedra. In Proceedings of the Sixth IEEE International Conference on Data Engineering, 1990, pp. 311–319.
24. Joseph M. Hellerstein, AviPfeffer, "The RD-tree: An Index Structure for sets". University of Wisconsin, Computer Science Technical report 1252, November 1994.
25. D. A. White, R. Jain. Similarity indexing with the ss-tree. In ICDE '96: Proceedings of the Twelfth International Conference on Data Engineering, pages 516–523, Washington, DC, USA, 1996. IEEE Computer Society.
26. N. Katayama, S. Satoh. The SR-tree: an index structure for high dimensional nearest neighbor queries. In SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data, pages 369–380, New York, NY, USA, 1997. ACM Press.
27. I. Kamel, C. Faloutsos. Hilbert R-tree: An improved R-tree using fractals. In Proceedings of the Twentieth International Conference on Very Large Data Bases, 1994, pp. 500–509.
28. I. Kamel, C. Faloutsos. Parallel R-trees. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 1992, pp. 195–204.
29. M. A. Nascimento, J.R.O. Silva, Y. Theodoridis. Evaluation of Access Structures for Discretely Moving Points. In Proc. of the Intl. Workshop on Spatio-Temporal Database Management, STDBM, pages 171–188, Sept. 1999.
30. W. Osborn, K. Barker. Searching through Spatial Relationships using the 2DR-tree. The IASTED Conference on Internet and Multimedia Systems and Applications Honolulu, Hawaii, USA August 14-16, 2006.
31. Y. Theodoridis, M. Vazirgiannis, T.

Sellis. Spatio- Temporal Indexing for Large Multimedia Applications. In Proc. of the IEEE Conference on Multimedia Computing and Systems, ICMCS, June 1996.
32. DBM-Tree: Trading Height-Balancing for Performance in Metric Access Methods Marcos R. Vieira, Caetano Traina Jr., Fabio J. T. Chino, Agma J.M. Traina.
33. Traina, C., Traina, A.J.M., Filho, R.F.S., Faloutsos, C.: How to improve the pruning ability of dynamic metric access methods. In: CIKM, pp. 219–226 (2002).
34. E. Vidal, "An algorithm for finding nearest neighbors in (approximately) constant average time," Pattern Recog. Lett., vol. 4, no. 3, pp. 145–157, 1986.
35. L. Mico, J.Oncina, E. Vidal An algorithm for finding nearest neighbours in constant average time with a linear space complexity Proceedings, 11th IAPR International Conference on Pattern Recognition. Vol. II. Conference B:
36. Pattern Recognition Methodoly and Systems Year; 1992.
37. L. Mico, J. Oncina, and R. C. Carrasco, "A fast branch & bound nearest neighbour classifier in metric spaces," Pattern Recog. Lett., vol. 17, no. 7, pp. 731–739, 1996.
38. A modification of the LAESA algorithm for approximated k-NN classification, Pattern Recognition Letters, Vol. 24, Issue: 1-3, January 2003, Pages: 47-53.
39. G. R. Hjaltason and H. Samet, "Index-driven similarity search in metric spaces," ACM Trans. Database Syst., vol. 28, no. 4, pp. 517–580, 2003.
40. R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In Proc. CPM'94, LNCS 807, pages 198–212, 1994.
41. Ricardo Baeza-Yates Gonzalo Navarro "Fast approximate string matching in a dictionary" Published in: String Processing and Information Retrieval: A South American Symposium, 1998. Proceedings IEEE.
42. E. Ch´avez, J.L. Marroquin, and G. Navarro. Fixed queries array: A fast and economical data structure for proximity searching. Multimedia Tools and Applications (MTAP), 14(2):113–135, 2001.
43. Spatial Selection of Sparse Pivots for Similarity Search in Metric Spaces,
44. JCS&T Vol. 7 No. 1, April 2007.
45. P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in Proc. Annu. ACM-SIAM Symp. Discrete Algorithms, 1993, pp. 311–321.
46. T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In Proc. SIGMOD'97, pages 357–368, 1997. Sigmod Record 26(2).
47. YIANILOS, P. N. 1998. Excluded middle vantage point forests for nearest neighbor search. Tech. rep., NEC Research Institute, Princeton, NJ. July. (Presented at the FirstWorkshop on Algorithm Engineering and Experimentation (ALENEX'99), Baltimore, MD, Jan.
48. C. Faloutsos, Y. Rong. DOT: A spatial access method using fractals. In Proceedings of the Seventh IEEE International Conference on Data Engineering, 1991, pp. 152– 159.
49. R. Bayer. The universal B-tree for multidimensional indexing. Tech. Rep. I9639, Technische Universitat Munchen, Munich, Germany. 1996.
50. J. Orenstein, T.H. Merrett. A class of data structures for associative searching. In Proceedings of the Third ACM SIGACT–SIGMOD

Symposium on Principles of Database Systems, 1984, pp. 181–190.
51. Samet, H. (1984). `The Quadtree and Related Hierarchical Data Structures', Computing Surveys, 16 (2), 187-260.
52. R. Finkel, J. L. Bentley. Quadtrees: A data structure for retrieval of composite keys. Acta Inf. 4, 1, 1974, pp. 1–9.
53. H. Samet, R. E. Webber. Storing a collection of polygons using quadtrees. ACM Trans. Graph. 4, 3, 1985, 182–222.
54. R.C. Nelson, H. Samet. A Consistent Hierarchical Representation for Vector Data. In Proc. of the ACM SIGGRAPH, pages 197–206, Aug. 1986.
55. KEDEM, G. 1982. The quad-CIF tree: A data structure for hierarchical on- line algorithms. In Proceedings of the Nineteenth Conference on Design and Automation, 352–357.
56. Principles of Multimedia Database Systems by VS Subrahmanian, Morgan Kaufmann Publishers, 1998.
57. Spatial Databases with Application to GIS by Philippe Rigaux, Michel Scholl, Agnes Voisard, Morgan Kaufmann Publishers (2002), pages 207-259.
58. W. Litwin. *Linear hashing: A new tool for file and table addressing*. In Proceedings of the Sixth International Conference on Very Large Data Bases, 1980, pp. 212–223.
59. P. A. Larson. *Linear hashing with partial expansions*. In Proceedings of the Sixth International Conference on Very Large Data Bases, 1980, pp. 224–232.
60. H.-P. Kriegel, B. Seeger. Multidimensional order preserving linear hashing with partial expansions. In Proceedings of the International Conference on Database Theory, LNCS 243, Springer-Verlag, Berlin/Heidelberg/New York. 1986.
61. A. Hutflesz, H.-W. Six, P. Widmayer. Globally order preserving multidimensional linear hashing. In Proceedings of the Fourth IEEE International Conference on Data Engineering, 1988, pp. 572–579.
62. R. Fagin, J. Nievergelt, N. Pippenger, R. Strong. Extendible hashing: A fast access method for dynamic files. ACM Trans. Database Syst. 4, 3, 1979, pp. 315–344.
63. J. Nievergelt, H. Hinterberger, K. Sevcik. The grid file: An adaptable, symmetric multikey file structure. In Proceedings of the Third ECI Conference, A. Duijvestijn and P. Lockemann, Eds., LNCS 123, Springer-Verlag, Berlin/Heidelberg/New York, 1981, pp. 236–251.
64. H. Six, P. Widmayer. Spatial searching in geometric databases. In Proceedings of the Fourth IEEE International Conference on Data Engineering, 1988, pp. 496–503.
65. K. Hinrichs. Implementation of the grid file: Design concepts and experience. BIT 25, 1985, pp. 569–592.
66. A. Hutflesz, H.-W. Six, P. Widmayer. Twin grid files: Space optimizing access schemes. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 1988, pp. 183–190.
67. A. Hutflesz, H.-W. Six, P. Widmayer. The R-file: An efficient access structure for proximity queries. In Proceedings of the Sixth IEEE International Conference on Data Engineering, 1990, pp. 372–

379.
68. K. Sevcik, N. Koudas. Filter trees for managing spatial data over a range of size granularities. In Proceedings of the 22th International Conference on Very Large Data Bases (Bombay), 1996, pp. 16– 27.
69. M. Tamminen. The extendible cell method for closest point problems. BIT 22, 1982, pp. 27–41.
70. C. Kolovson, M. Stonebraker. Segment indexes: Dynamic indexing techniques for multi-dimensional interval data. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 1991, pp. 138–147.
71. C. Traina Jr., A. Traina, B. Seeger, C. Faloutsos. Slim- trees: High Performance Metric Trees Minimizing Overlap Between Nodes. International Conference on Extending Database Technology (EDBT) 2000, Konstanz, Germany, March 27-31, 2000.
72. I. Kalantari, G. McDonald. A data structure and an algorithm for the nearest point problem. IEEE Trans. Software Eng., 9(5):631–634, 1983.
73. S. Brin. Near neighbor search in large metric spaces. In VLDB '95: Proceedings of the 21th International Conference on Very Large Data Bases, pages 574–584, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.
74. HJALTASON, G. R. AND SAMET, H. 2003a. Improved search heuristics for the SA-tree. *Patt. Rec. Lett. 24*, 15 (Nov.), 2785–2795.
75. P. Ciaccia, M. Patella, P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases, pages 426–435, San Francisco, CA.
76. Indexing metric spaces with m-tree. P Ciaccia, M Patella, F Rabitti, P Zezula
- SEBD, 1997.
77. V. Dohnal, C. Gennaro, P. Savino, P. Zezula: D-Index: Distance Searching Index for Metric Data Sets. To appear in ACM Multimedia Tools and Applications, 21(1), September 2003.
78. V. Dohnal, C. Gennaro, and P. Zezula, "Similarity join in metric spaces using eD-Index," Database Expert Syst. Appl., vol. 2736, pp. 484–493, 2003.
79. E. Chavez and G. Navarro, "A compact space decomposition for effective metric indexing," Pattern Recog. Lett., vol. 26, no. 9, pp. 1363–1376, 2005.
80. M. Mamede. Recursive lists of clusters: A dynamic data structure for range queries in metric spaces. In Proc. 20th Intl. Symp. on Computer and Information Sciences (ISCIS'05), LNCS 3733, pages 843–853, 2005.
81. G. Navarro and N. Reyes, "Dynamic list of clusters in secondary memory," in Proc. 7th Int. Conf. Similarity Search Appl., 2014, pp. 94–105.
82. R. Paredes and N. Reyes, "Solving similarity joins and range queries in metric spaces with the list of twin clusters," J. Discrete Algorithms, vol. 7, no. 1, pp. 18–35, 2009.
83. J. Almeida, R. D. S. Torres, and N. J. Leite, "BP-tree: An efficient index for similarity search in high-dimensional metric spaces," in Proc. ACM Int. Conf. Inf. Knowl. Manage., 2010, pp. 1365–1368.
84. B. Seeger, H.-P. Kriegel. The buddy-tree: An efficient and robust access method for spatial data base systems. In Proceedings of the Sixteenth International Conference on Very Large Data Bases, 1990, pp.

590– 601.
85. M. Freeston. The BANG file: A new kind of grid file. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 1987, pp. 260–269.
86. M. Freeston. A *general solution of the n-dimensional B-tree problem*. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 1995, pp. 80–91.
87. A. Kumar. *G-tree: A new data structure for organizing multidimensional data*. IEEE Trans. Knowl. Data Eng. 6, 2, 1994, pp. 341- 347.
88. H. Blanken, A. Ijbema, P. Meek, B. Van den Akker. The generalized grid file: Description and performance aspects. In Proceedings of the Sixth IEEE International Conference on Data Engineering, 1990, pp. 380– 388.
89. T. Skopal, J. Pokorny, and V. Snasel, "PM-tree: Pivoting metric tree for similarity search in multimedia databases," in Proc. ADBIS, 2004, pp. 803– 815.
90. D. Novak, M. Batko, and P. Zezula, "Metric Index: An efficient and scalable solution for precise and approximate similarity search," Inf. Syst., vol. 36, no. 4, pp. 721–733, 2011.
91. H.-P. Kriegel, B. Seeger. PLOP-hashing: A grid file without directory. In Proceedings of the Fourth IEEE International Conference on Data Engineering, 1988, pp. 369–376.
92. L. Chen, Y. Gao, X. Li, C. S. Jensen, and G. Chen, "Efficient metric indexing for similarity search," in Proc. IEEE 31st Int. Conf. Data Eng., 2015, pp. 591–602.
93. Database System Concepts, Silberschatz−Korth−Sudarshan: Sixth Edition, McGraw-Hill Publishers.
94. Database Systems The Complete Book, Molina, Ullman Second Edition 2008.
95. Fundamentals of Database Systems book, Elmasri, Navathe, Sixth Edition.
96. Bitmap index design and evaluation CY Chan, YE Ioannidis - ACM SIGMOD Record, 1998.
97. Multi-table joins through bitmapped join indices P O'Neil, G Graefe - ACM SIGMOD Record, 1995.
98. Join indices P Valduriez - ACM Transactions on Database Systems (TODS), 1987.
99. Efficient join-index-based spatial-join processing: A clustering approach Shashi Shekhar, Chang Tien Lu, Sanjay Chawla, Sivakumar Ravada.
100. D. Rotem. "Spatial Join Indices". In IEEE Transactions on Knowledge and Data Engineering, Kobe, April 1991.
101. "Data Warehousing, Data Mining and OLAP", Alex Berson and Stephen J.Smith, Tata McGraw – Hill Edition 2008.