Abstract: In spatial databases search operations take an important role. These operations consist of the point query (find all objects that contain a given search point), the range query (find all objects that overlap a given search range) [GAE98] and the nearest neighbor query (find k objects (k >= 1) that are closest to a given object). They are very costly operations. Their performance is affected by both CPU-time and IO-cost. The long history of researches on spatial databases has resulted in many multidimensional access methods to efficiently support such operations. Each of them has strengths and deficiencies as well. This paper preliminary develops taxonomy for these multidimensional access methods and describes some prominent multidimensional access methods, which have been recently introduced as well as their comparative studies. Moreover, the important role of multidimensional access methods for supporting real-world applications in the next decade is also discussed.

Keywords: multidimensional access method (MAM), similarity search, spatial database, multidimensional database, bounding sphere (BS), minimum bounding rectangle (MBR), feature vector, similarity indexing.

1. INTRODUCTION

Spatial databases (SBs) have some typical properties that are different from the conventional databases. They are briefly discussed below to show indispensability of multidimensional access methods (MAMs, for short) in such database systems.
First, spatial databases are complex in structures and relationships. The spatial description of objects is especially extensive because a spatial data object may be composed of a single point or several thousands of polygons arbitrarily distributed across space [GAE98]. Each data object has more than one associated coordinate. Store such data objects in a single relational table with a fixed tuple size is usually not possible and not efficient.

Second, spatial databases tend to be large in quantity. Many SDs of real world interests are very large with sizes can be up to millions objects [OOI91].

Third, spatial databases are often dynamic. It means that updates, insertions and deletions can be interleaved. Data structures used to support these dynamic operations must not deteriorate querying performance.

Forth, spatial database operators are generally more expensive than standard relational operators are. One class of important operators in SDs is search operators. These operators usually base on spatial location of data objects. They require fast execution time as well as minimizing the number of accessed disk pages.

Because of large volume of the spatial databases, it is typically not efficient to pre-compute and store spatial relationships among all data objects [OOI91]. They are dynamically constructed during the query processing instead. To efficiently support the search operators of spatial objects, an index structure on their spatial locations is necessary. The main problem when designing such index structures is that there exists no total order among spatial data objects so that spatial proximity is still preserved. This effect cannot be simply obtained from employing some single index structures at the same time over spatial tables. In the next sections, some prominent multidimensional index structures [1] are summarized to clear this problem and to give more details how beneficial one can get from them.

This paper is organized as follow. Section 2 summarizes some well-known modern MAMs as well as comparative studies that had been shown in their papers. A new taxonomy is also developed to categorize MAMs into four classes: KD-tree based techniques, R-tree based techniques, hybrid

---

1 In this paper, it is also considered multidimensional access methods
techniques of both KD-tree and R-tree and other techniques. Section 3 gives conclusions and prediction of MAMs in the future.

2. MULTIDIMENSIONAL ACCESS METHODS

There are some criteria can be used to classify MAMs as structure of the directory nodes \(^2\), e.g. MBRs or BSs, MAMs are static or dynamic, exact or approximate, etc. (see [GAE98] for more details). In this paper, however, we consider MAMs as Data Partitioning (DP)-based/R-tree-based or Space Partitioning (SP)-based/KD-tree-based index techniques [CHA99], [KHA01]. Moreover, some MAMs that integrate both of them are classified into a new categorization called hybrid techniques. In details, we categorize MAMs into four classes as follows:

- KD-tree based techniques
- R-tree based techniques
- Hybrid techniques of both KD-tree and R-tree
- Other techniques

Several summaries of MAMs have been published previously in [GAE98], [OOI91], etc. The latest publication on this topic is in [GAE98]. It summarized the history of MAMs in 30 years from 1966 to 1996. In the next subsections, we present prominent MAMs introduced from 1996 to 2001. Related basic index techniques, however, are given as well. In details, section 2.1 and its subsections are devoted to introduce the KD-tree [BEN75] and index techniques based it as VAMSplit-tree [WHI96a], LSDh-tree [HEN98]. Section 2.2 presents the R-tree [GUT84] and the R-tree based techniques, which consist of TV-tree [LIN94], SS-tree [WHI96], X-tree [BER96], SR-tree [KAT97], M-tree [CIA97], A-tree [SAK00]. Section 2.3 describes hybrid techniques, which are derived from both the KD-tree and the R-tree as Hybrid tree [CHA99], SH-tree [KHA01]. Section 2.4 introduces techniques that cannot classify into any three ones above like pyramid technique [BER98], VA-File [WEB98]. Besides, application range of each of these MAMs is also discussed. Section 2.5 will give the comparative studies that have been shown in the related papers. Figure 1 shows our classification schema of these index techniques, which will be discussed in the subsections.

\(^2\) Also called internal nodes
2.1. KD-Tree Based Techniques

2.1.1. The KD-tree

The KD-tree [BEN75] was published in 1975. It is one of the first MAMs. The KD-tree is a binary search tree composed by recursively splitting d-dimensional data space into two subspaces by means of (d-1)-dimensional hyperplanes. Its pros-and-cons have been shown in literature. The KD-tree has constant fan-out without depending on dimension number of the space. It is fast for insertion and has no overlap between the subspaces. Nevertheless, the KD-tree depends on order of inserted data objects. It means that the KD-tree is not robust for sorted data. Besides, the KD-tree also bears dead space problem and does not appropriate for secondary (tertiary) storage.

2.1.2. The VAMSplit-tree

The VAMSplit-tree [WHI96a] is a static MAM, i.e. all data objects must be available at the time of creating the tree. The VAMSplit-trees are rather similar to the KD-trees but have some discriminated aspects. It chooses split dimension that has the largest variance instead of the
maximum spread dimension. In addition, the split position in the selected split dimension is chosen to minimize the used bucket number while still approximating a median split. However, the VAMSplit-trees do not clearly show benefit of variance-based split and the size of a VAMSplit-tree is limited by available main memory.

2.1.3. The LSDh-tree

The LSDh-tree [HEN98] has been really derived from the LSD-tree [HEN89], but the LSD-tree has been depended on the KD-tree so the LSDh-tree is classified into this categorization. The LSDh-trees have two-level directory: the first one (internal directory) in memory and the second one (external directory) is kept in secondary storage. The internal node splits just depend on objects located in these internal nodes themselves (so the tree is called LSDh-tree – LSD stands for Local Split Decision). It inherits positive aspects of the KD-tree-based MAMs like fast insertion, deletion. The dead space problem that is inherited from the KD-tree is overcome by using the coded actual data regions (CADR). This technique is applicable to many MAMs. The LSDh-tree is designed for managing both point objects and extended data objects (EDO) as polygons, lines, etc. However, the LSDh-tree is still sensitive to pre-sorted data.

2.2. R-Tree Based Techniques

2.2.1. The R-tree

In [GUT84], Guttman introduced the R-tree, which is a multidimensional generalization of the B-tree. The R-tree is one of the earliest proposed MAMs for extended spatial objects and is balanced in height. Each R-tree is a hierarchical structure of minimum bounding rectangles (MBRs). Each entry of internal nodes consists of a MBR of its descendants and a pointer pointing to its child node. Each entry in a leaf node contains a MBR of a spatial data object \(^{[3]}\) and its object identifier. The algorithms for inserting, deleting and querying are concretely presented in [GUT84]. The main weakness of the R-tree is overlap of the MBRs in high dimensional space, which causes the performance of the R-tree to be deteriorated seriously. Nevertheless, the R-tree and some its variants have been widely used in many commercial database systems, typically as in Oracle.

\(^{3}\) It can be a point object or an EDO (extended data object). An EDO is sometimes called extended spatial object in this paper.
There is a considerable variant of the R-tree that is the R*-tree [BEC90]. Its design introduced some important improvements as forced reinsert policy, a new insertion algorithm and so on. Its performance improvement has been reported up to 50% compared to the basic R-tree.

2.2.2. The TV-tree

The TV-tree (Telescopic-Vector tree) [LIN94] has the R*-tree-like structure (and the R-tree also). It improves the performance of the R*-tree by using a variable number of dimensions for indexing, adapting to the number of objects to be indexed and to the current level of the tree [LIN94]. This idea is motivated by softening phenomenon called “dimensional curse” in querying high dimensional spatial objects of MAMs, i.e. the performance of MAMs is degenerated when going to higher dimensions. Dimensions are ranked by their importance and only a few first dimensions, which are “more” important than others, are used for indexing. For nodes that are close to the root, just a small number of dimensions are used, thus these nodes can get high fan-out. More dimensions must be employed for indexing as descending the tree to discriminate the objects.

As mentioned in [WHI96] and [KAT97], the TV-tree depends on two assumptions: (1) dimensions can be ranked by their importance and (2) there exist feature vectors that allow the shift of index dimensions. For real-valued feature vectors, the second condition is not always true due to diversity of their coordinate. Therefore, the performance of the TV-tree is depended on applications.

2.2.3. The SS-tree

The SS-tree (Similarity Search tree) [WHI96] has been designed for similarity indexing of spatial objects [4]. It improves the performance of R*-tree by employing bounding spheres (BSs) instead of MBRs for the region shape. The insertion algorithm of the SS-tree also uses forced insert policy as in the R*-tree but a node is split if reinsertion has been made on it before. Because BSs have short diameters than MBRs [KAT97], the performance of nearest neighbor queries in the SS-tree is better than one of the R*-tree. As declared, the SS-tree can scale up to large databases and index high dimensional feature vectors. Nevertheless, BSs of the SS-tree highly tend to overlap in high

---

[4] The data structures of the SS-tree are designed for both spatial points and extended spatial objects but the test results have been shown only for spatial points.
dimensional spaces. This causes the SS-trees to be deteriorated in the high dimensional spaces as the R-tree. To overcome weaknesses of both the SS-tree and the R-tree, the SR-tree has been published afterwards.

2.2.4. The X-tree

In [BER96], the X-tree (eXtended node tree) has been introduced as a variant of the R*-tree. It improves the performance of the R*-tree by using two techniques: (1) the X-tree introduces an overlap-free split algorithm, which depends on the split history of the X-tree, (2) if the overlap-free split algorithm results in under-filled nodes, the X-tree omits the split and the corresponding internal node becomes a supernode. The supernodes are internal (directory) nodes that can be enlarged dynamically. Although the performance of the X-tree is better than the R*-tree as shown in the paper, the dynamic construction of a X-tree is very time-consuming [BER00].

2.2.5. The SR-tree

The SR-tree (sphere/rectangle tree) [KAT97] has been recently introduced for indexing spatial point data. It is an improvement of the SS-tree and the R-tree where both concepts, bounding rectangle of the R-tree and BS of the SS-tree, are integrated to form the SR-tree. Katayama et al. show that bounding rectangles can divide points into small volume regions but have much longer diameters than BSs. Furthermore, BSs permit to divide points into short diameter regions but they tend to have larger volumes than bounding rectangles. By applying both of BSs and bounding rectangles, the SR-tree shows the better performance when comparing to the SS-tree and one of the most successful variants of the R-tree, say the R*-tree. The main weakness of the SR-tree is fan-out problem. The fan-out of the SR-tree is only one third of the SS-tree and two thirds of the R*-tree [KAT97]. This problem may cause more nodes to be read on queries and the query performance of the SR-tree to be reduced.

2.2.6. The M-tree

The M-tree (paged metric tree) [CIA97] [ZEZ96] has been proposed for indexing objects, which are unnecessary to be fit into feature vector-based representations, where the Lp metric is being employed like many MAMs. One of such examples is the edit distance for string similarity, i.e. the
minimum character number that must be inserted, deleted or replaced to transform a string S1 into a string S2. Generally, the structure of the M-tree is similar to that of the R-tree based techniques; especially it is similar to the SS-tree. However, the M-tree is more flexible and sophisticated as well as its application range is wider. In the high dimensional spaces, like the SS-tree, the M-tree also incurs the overlap problem between its covering regions.

2.2.7. The A-tree

In fact, the A-tree (Approximate tree) [SAK00] is motivated by comparison and analysis of the SR-tree and the VA-file and then improves their weaknesses. Nevertheless, it carries the most important characteristics of the SR-tree so we classify it into the class of the R-tree-based MAMs. Each node of the A-tree contains a MBR and representation of relative approximation of its children. Besides, in each node, centroid of data objects is also stored for update operations.

In accordance with [SAK00], the usage frequency of MBSs is decreased because MBSs occupy much larger volume than MBRs in high dimensional spaces. Therefore, the A-tree just employs MBRs and the centroids of objects as mentioned above. In addition, the A-tree has introduced the relative approximation (in the VA-file is absolute approximation) in which bounding regions or data points are approximated by their relative positions in terms of parent’s bounding region only.

2.3. Hybrid techniques of both KD-tree and R-tree

Hybrid techniques have been introduced in many MAMs as skd-tree [OOI87], UB-tree [BAY96], R-files, G-tree, etc. (see [GAE98] for a survey), Hybrid tree [CHA99], SH-tree [KHA01] and so forth. Generally, these techniques employ more than one concept from some others ones to form new MAMs. The UB-tree (Universal B-tree), for example, is formed by integrating the z-ordering technique [ORE84] and ideas of the B-tree together. In subsections below, however, we just present some typical techniques that derive from both the R-tree and the KD-tree or their variants.

2.3.1. The Hybrid tree

The Hybrid tree [CHA99] is combined by using representation of internal nodes like the KD-tree and treating the indexed subspaces as bounding regions in the R-tree while traversing the tree. As in
the skd-tree [OOI91], the Hybrid tree also allows overlap between subspaces, which is different from the pure concept of the KD-tree. To eliminate dead space in the subspaces, the Hybrid tree employs a technique introduced in the LSDh-tree [HEN98] called CADR (Coded Actual Data Region). Another considerable aspect of the Hybrid tree is, as feature-based MAMs, support of queries with arbitrary distance measures.

2.3.2. The SH-tree

The SH-tree (Super Hybrid tree) [KHA01] is motivated from alleviating the fan-out problem of the SR-tree, together with trying to conserve the data clustering aspect, which is a weakness of the pure SP /KD-tree based index techniques. In fact, the SH-tree is a well-combined structure of both the KD-tree and the SR-tree based techniques. Its structure has three kinds of nodes: the internal nodes are organized as those of the KD-tree but the balanced nodes are the same as those of the SR-tree. Besides, its leaf nodes are just below the balanced nodes and they keep feature vectors information (and some other additional information) of the objects. Note that, in the SH-tree, the balanced nodes are not nested like in the SR-tree. This structure has overcome the fan-out problem of the SR-tree to improve its search performance.

2.4. Other Techniques

In addition to the R-tree-based and the KD-tree-based MAMs, there are MAMs that cannot be categorized into either of them or their hybrid class. These MAMs can be derived from hashing techniques, space-filling curves and so on (see [GAE98], [OOI91] for an overview). In this section, we introduce some of the most prominent MAMs like that, which have been recently published.

2.4.1. The Pyramid Technique

The Pyramid technique [BER98] has been developed as a new index method for high dimensional spaces without deriving from any other ones that have been proposed before. Its design philosophy depends on mapping from d-dimensional space into a one-dimensional space and then store these one-dimensional data objects into a B+-tree to take its advantages, which have been proven in the literature. The Pyramid technique partitions a d-dimensional space into 2d pyramids having the center point of the space (0.5, 0.5, … 0.5) as their top and a (d-1)-dimensional surface of the space
as their base. Afterwards, each of the 2d pyramids is divided into several sub-partitions shaped like peel of an onion (trapezoids), which can be stored in one data page. This approach, as declared, overcomes the curse of dimensionality problem, which other MAMs must be carried when going to the high dimensional spaces. As experiments show, the Pyramid technique is very efficient for almost boxing range queries.

2.4.2. The VA-file

The VA-file (Vector Approximation file) [WEB98] divides the data space into cells and addresses them with bit-strings of length i. All vectors inside each cell are approximated by the cell itself. The cells are kept in an array. Search operation scans entire this array to determine candidate cells (vectors). These candidates are checked to give the result vectors. As reported in the paper, the VA-file outperforms the R*-tree as well as the X-tree when the dimension number of the data space is greater or equal six. This kind of approximation is called absolute positions based approximation [SAK00]. It is independent of data distribution, thus a large number of dense data objects tend to be approximated by the same value. Hence, the VA-file is not efficient when skew data are approximated and indexed.

2.5. Comparative Studies

In this section, we briefly present theoretical and experimental results by comparison between multidimensional access methods. Through these results, we have a more exact assessment over MAMs, although their total order is very difficult to show absolutely.

Lin et al. [LIN94] had compared the TV-tree to the R*-tree for 16000 spatial data objects. In fact, the test data are words that are transformed into feature vectors of 32 dimensions. The experimental results have shown that the TV-tree outperformed the R*-tree up to 67-73% savings in total disk accesses for exact matches and about 40% savings for range queries. They also tested on various sizes of database objects. The results show that as the object size increases, the fan-out of the TV-tree decreases, making the TV-tree grows faster. However, the TV-tree still has improvement of about 60% over the R*-tree for exact match and 40% for range queries. Besides, the needed space of the TV-tree is less then that of the R*-tree as well.
Berchtold et al. [BER96] compared the X-tree to the R*-tree and the TV-tree for both real and synthetic data sets. Concerning the synthetic data set, the experimental results show speed-up of the search time for point queries of the X-tree over the R*-tree up to about 270 with 16-dimensional data. For lower dimensions, the X-tree is still superior to the R*-tree (for d=8, the speed-up is about 30). With respect to the real data set, say Fourier data set consists points in high dimensions; the speed-up for point queries is even higher than one of the synthetic data set. As reported by Berchtold et al., the speed-up is about 90 and 320 for 4- and 8-dimensional spaces, individually. Besides, for nearest neighbor queries, the speed-up of the X-tree is also consistently higher than that of the R*-tree, between about 10 for d=6 and 20 for d=20 (see [BER96] for a detailed description). In addition, the test results also showed superiority of the X-tree over the TV-tree with the speed-up ranging from 4 to 12, even for the rather small databases. They also gave an interesting result: the performance of the R*-tree is better than that of the TV-tree for dimensions that are smaller than 16.

White et al. [WHI96] compared the SS-tree to the R*-tree. The experimental results showed that the SS-tree provides faster nearest neighbor query performance than the R*-tree in almost tests in high dimensional spaces (d>5). Besides, White et al. [WHI96a] also tackled the comparisons between VAMSplit-trees and the SS-tree as well as the R*-tree. They turned out that the VAMSplit-trees are superior to both later ones on 20-nearest neighbor queries.

Ciaccia et al. [CIA97] compared the M-tree to the R*-tree in processing similarity queries. The empirical results, as reported, show the merits of the M-tree over the R*-tree in both IO-cost and CPU-time.

Katayama et al. [KAT97] compared the SR-tree to the SS-tree, the VAMSplit-tree and the R*-tree for 21-nearest neighbor queries. For uniform and real data set, the experimental results show that the SR-tree reduces the CPU-time to 91% and 67% of the SS-tree and the disk access number to 93% and 68% of the SS-tree, individually. Specially, the SR-tree outperforms the VAMSplit-tree for the real data set with a notice that the VAMSplit-trees are static index structures while the SR-trees are dynamic ones. They also confirmed that both the SR-tree and the SS-tree is better than the R*-tree with respect to processing the nearest neighbor queries.

Berchtold et al. [BER98] compared the Pyramid technique to the X-tree and other MAMs. Their evaluation comprises both real and synthetic data sets. They conducted the evaluations on boxing
range queries. For the synthetic data, the Pyramid technique with 2 million objects in 100-dimensional space performs the range queries 879 times faster than the X-tree in total elapsed time, which comprises CPU-time and IO-time. With this kind of data, the Pyramid technique also scale well to higher dimensions. They tackled experiments on 8, 12, 16, 20 and 34 dimensions with database size is fixed at 1.000.000 objects. The results show that the Pyramid technique is up to 2500.7 times faster than the X-tree in terms of total elapsed time. For the real data sets, the experimental results also show superiority to the X-tree by orders of magnitude.

Henrich [HEN98] compared the LSDh-tree to the X-tree. The performance tests are performed with n-nearest neighbor queries (n=2, 4, 8, … 1024). For the uniform data, the performance of the LSDh-tree with CADRs (coded actual data regions) is slightly better than that of the X-tree. However, when not using CADRs, the LSDh-tree’s performance is worse than the performance of the X-tree. For the real data sets that is used in the test, say CAR and TIGER data sets [HEN98], the LSDh-tree outperforms the X-tree by far.

Weber et al. [WEB98] compared the VA-file to the R*-tree and the X-tree. Their experimental results also confirm their theoretical analyses. For spaces that have dimension number $d \geq 6$, the VA-file outperforms both the X-tree and the R*-tree. They also indicate that the X-tree has lost its advantages in high dimensional spaces.

Recently, Chakrabarti et al. [CHA99] have introduced the Hybrid tree. It has been reported to consistently outperform the SR-tree, the hB-tree [LOM90] and sequential scan for nearest neighbor queries and easily scale to the high dimensional spaces.

Sakurai et al. [SAK00] have compared the A-tree to the SR-tree and the VA-file. For non-uniform data, the A-tree outperforms both the VA-file and the SR-tree up to 64-dimensional space, which is the highest dimension in their tests. For 64-dimensional real data, the A-tree requires 77.3% (77.7%) fewer page accesses than the SR-tree (the VA-file).

The last index technique of our schema in figure 1 is the SH-tree [KHA01]. The performance of this MAM has not been reported so far. However, in our recent performance tests, the SH-tree is superior to the SR-tree and the Hybrid tree for uniform data sets with both nearest neighbor queries and range queries in the total elapsed time needed to process the queries.
3. CONCLUSIONS

Researches in database community for over 35 years have resulted in many MAMs. Each of them has advantages and weaknesses as well. This paper shows an overview over recent published well-known MAMs, say from 1996 to 2001 [5]. Unfortunately, there has not any standard to rank MAMs so far because there are so many different criteria to define optimality and so many parameters to determine performance of MAMs [GAE98]. Therefore, one still has to depend on experiences to select suitable MAMs for their applications. Application range of MAMs is various like Geographical Information System (GIS), multimedia databases [SEI97], time-series databases [FAL94], CAD/CAM systems [BER97], medical image databases [KOR96], etc. All of them are important application fields that are both in the past and for the future. This again asserts that MAMs are indispensable in spatial/multidimensional databases or in other words, they are really the important factor for current and next decade’s applications in such databases.

In our perspective, MAMs tend to focus on supporting specific application domains in the near future. The benchmark may be introduced but it needs time to verify and to make one accept it as an official standard.

REFERENCES


3 Except for the TV-tree and the basic MAMs


