BM^{+}-tree: A Hyperplane-based Index Method for High-Dimensional Metric Spaces

Xiangmin Zhou[1], Guoren Wang[1], Xiaofang Zhou[2] and Ge Yu[1]

[1]College of Information Science and Engineering, Northeastern University, China
[2]School of Information Technology & Electrical Engineering, University of Queensland, Australia

Abstract. In this paper, we propose a novel high-dimensional index method, the BM^{+}-tree, to support efficient processing of similarity search queries in high-dimensional spaces. The main idea of the proposed index is to improve data partitioning efficiency in a high-dimensional space by using a rotary binary hyperplane, which further partitions a subspace and can also take advantage of the twin node concept used in the M^{+}-tree. Compared with the key dimension concept in the M^{+}-tree, the binary hyperplane is more effective in data filtering. High space utilization is achieved by dynamically performing data reallocation between twin nodes. In addition, a post processing step is used after index building to ensure effective filtration. Experimental results using two types of real data sets illustrate a significantly improved filtering efficiency.

Keywords: Similarity search, Multidimensional index, Binary hyperplane, Range query, K-NN query.

1 Introduction

With the rapidly growth of various types of multimedia information, the need for fast processing of content-based similarity search queries in large databases has increased dramatically, and will increase at a much faster pace in the future. Since retrieving multidimensional data always incurs very high, and sometimes prohibitively high, costs for large datasets, the search for effective index structures to support high dimensional similarity query has been at the frontiers of database research in the last decade[13]. Most efficient multidimensional indexing structures originated from managing low-dimensional geographical data (such as R-tree and it variants[1–4]) are not efficient in managing high dimensional data.

The approach supporting multidimensional similarity search can be classified into two categories: *position-based* indexes and *metric-based* indexes. The R-tree and its variants are representatives of the former, which deal with the relative positions in a vector space. The second type of indexes, on the other side, include the VP-tree[5], the MVP-tree[7], the M-tree[6, 8], and its optimized indexes[9–11]. These indexes manage the data based on the relative distances between objects. Among the metric-based indexes, VP-tree is the first hierarchical

index structure which supports similarity search by utilizing the relative distance between objects and triangular inequality. It is of great significance for reducing the cost of similarity search. However, the query performance of VP-tree is severely suffered from a large quantity of distance calculation due to the small fanout (thus a very tall index tree). It should be pointed out that distance calculation in this type of application is typically very complex and CPU intensive. The MVP-tree is proposed to overcome these problems, by introducing multiple vantage points instead one. This idea significantly lowered its height. Both the VP-tree and the MVP-tree are constructed in a top-down way. That means they cannot support insertion and deletion of data once the index is created.

The M-tree represents a significant step forward, and is representative of metric-based indexes. It is a paged and balanced tree which adopts the bottom-up construction strategy with node promotion and split mechanisms. Therefore, it is suitable as a secondary storage index structure and can handle data updates gracefully without reconstructing the whole index when a media object is inserted or deleted. The M-tree is also the first one to recognize the high cost of distance calculation, and most distances are pre-computed and stored in the index tree, thus query-time distance calculation can be avoided. The large extent of subspace overlapping among M-tree sibling nodes, however, is a noticeable problem. Different from other high dimensional indexes, the M^+-tree has subspace overlapping minimization and tree height minimization as its aim. It improves the M-tree from the following points: (1) the concept of key dimension is proposed to eliminate the overlapping between twin nodes and to reduce the overlapping across subspaces; (2) the concept of twin nodes are introduced to lower the height of tree; (3) the idea of key dimension shift is proposed to achieve optimal space partitioning; and (4) a brand new idea of associating an index entry with twin subtrees for more efficient filtering during search.

This paper proposes a binary M^+-tree, called BM^+-tree, which improves the data partitioning method used in the M-tree. Like M-tree, BM^+-tree is a dynamically paged and balanced index tree. It inherits the node promotion mechanism, triangle inequality and the branch and bound search techniques from M-tree. BM^+-tree also fully utilizes the further filtering idea as used in M^+-tree. However, BM^+-tree uses a rotatable binary hyperplane, instead of a key dimension, to further partition the twin subspaces and to perform filtration between them. This novel idea, as we shall discuss in this paper, can improve the query processing performance significantly comparing to M-tree and M^+-tree.

The rest of the paper is organized as follows. In Section 2, we give some definitions for similarity searches. Section 3 introduces a new partition strategy. We describe BM^+-tree in Section 4, including its key techniques and algorithms. Section 5 presents performance evaluations. Section 6 concludes this paper.

2 Similarity Queries

In this section, we follow the conventions used in [8] and give basic definitions related to the BM^+-tree, including r-neighbor search, k-nearest neighbor search.

R-neighbor search and k-nearest neighbour search are two basic types of similarity queries. Commonly, the former is to obtain all objects within certain distance from the query object, while the latter is to find k objects which have the minimum distances to a given query object. They can be defined as follows.

Definition 1. *r-neighbor search.* Given a query object $q \in O$ and a non-negative query radius r, the r-neighbor search of q is to retrieve all the objects $o \in O$ such that $d(q, o) \leq r$.

Definition 2. *k-nearest neighbor search.* Given a query object $q \in O$ and an integer $k \geq 1$, the k-NN query is to retrieve k objects from O with the shortest distances from q.

The purpose of indexing a data space is to provide an efficient support for retrieving objects similar to a reference (query) object (for r-neighbor search or k-NN search). Here, for a given query, our main objective is to minimize the number of distance calculations, I/O operations and priority queue accesses, which are usually very expensive for many applications.

3 Data Partitioning Using Binary Hyperplane

The Strategy of data partitioning using binary hyperplanes is the main idea of BM$^+$-tree. This section will introduce this technique, including how to choose the binary hyperplanes, how to use the binary hyperplanes for data partitioning, and how to use the binary hyperplanes for filtering during the search process.

3.1 Construction of Binary Hyperplanes

Generally speaking, because of different data distributions, different dimensions carry a different weight in distance computation. Based on this fact, M$^+$-tree partitions a subspace into twin subspaces according to the selected *key dimension* which is the dimension that affects distance computation most. Different from M$^+$-tree, BM$^+$-tree uses a *binary hyperplane* to partition a subspace into twin subspaces. The binary hyperplane idea extends the (single) key dimension concept of the M$^+$-tree to make use of two key dimensions. This new data partition strategy is mainly based on the following observation.

Observation 1 *For most applications, except for the first key dimension, there is commonly another dimension which may also contains a large quantity of information. Cancelling the second dimension would cause great loss of information. Just as in the process of dimensionality reduction, we usually maintain the first few dimensions, instead of just the first one. Therefore, when performing further data partitioning, it is advisable to keep two dimensions that have maximal value variances and construct a binary hyperplane by using two dimensions.*

Figure 1 gives an example set of data (the shaded area). Obviously, the extent of objects along $x1$ is longer than that along x (which is in turn longer than that along y). The key dimension based data partitioning will divide data along x dimension. It is clear that much more information can be obtained if the partitioning is done using a binary hyperplane vertical to $x1$.

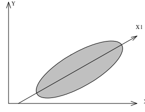

Fig. 1. A sample of data distribution.

The selection of a hyperplane has to conform to the following rules in order to achieve optimal data partitioning, i.e., trying to keep the objects having the minimal distances in the same subspace while minimizing the overlap between twin subspaces. In the process of binary hyperplane construction, the selection of two key dimensions and the decision of the coefficients of them are the two major issues. We consider two binary hyperplane construction strategies: m-RAD-2 based strategy and max-distance based strategy. The former is based on the fact that in the M-tree the query performance of index is optimal when it adopts the m-RAD-2 partitioning strategy. This strategy ensures the maximum of the radii of the two subspaces split minimal. The latter one, on the other side, is based on our observation that the distances between objects along the max-distance dimension can usually keep maximal quantity of information.

The BM$^+$-tree adopts the following steps to determine the hyperplane:

1. Choose two reference points in the following way. Use the center points of the two subspaces according to the m-RAD-2 partition strategy [8], and when these two points have the same feature value, one can compute the distances among objects in the subset, and choose two points from the subset such that the distance between the selected points is the maximal;
2. Compute the distance along each dimension between these two points; and
3. Choose two dimensions which have the biggest absolute values as the key dimensions, and consider the differences between the two center points of the two key dimensions as the coefficients of them respectively.

3.2 Binary Hyperplane based Further Data Partition

Data partition strategy is one of the most important issues which directly affect the performance of indexes. Before introducing the binary hyperplane based data partitioning, we need to introduce another concept.

Definition 3. *Twin nodes.* In the M$^+$-tree and the BM$^+$-tree, an internal entry has two pointers pointing to two subtrees. These two subtrees are called twin subtrees, and the roots of the twin subtrees are called twin nodes.

In Figure 2 (a) and (b), subspaces 1 and 2 correspond to the twin nodes of the tree. Figures 2(a), (b) and (c) show the data partitioning of the BM$^+$-tree, the M$^+$-tree and the M-tree respectively. The M-tree adopts distance-based data partition strategy which partitions a data space into two subspaces according to the distances between objects. Among the proposed partitioning methods in the M-tree, partitioning by m-RAD-2 is proved to be the best. The M$^+$-tree improves the data partition methods of the M-tree by adopting two steps data

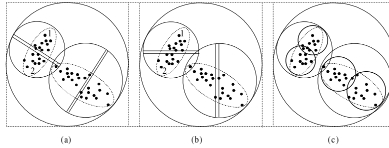

Fig. 2. Data partitioning for BM$^+$-tree, M$^+$-tree and M-tree.

partition strategy, i.e., partitioning with m-RAD-2 way proposed in the M-tree first and then further partitioning the two subspaces into two pairs of twin nodes according to the selected key dimensions (which can be different for different subspaces). The twin nodes are expressed through two boundary values of a key dimension which are the maximal key dimension value of the left twin space and the minimal key dimension value of the right twin space.

The BM$^+$-tree also adopts two steps data partition strategy. Different to the M$^+$-tree, it uses a rotatable binary hyperplane rather than the key dimension to further partition the twin subspaces, i.e. the binary hyperplane can be different for the two data spaces and the direction of it can change according to the data distribution in the data subspace considered. The binary hyperplane construction strategy proposed in this paper ensures the binary hyperplane rotatable, deciding the key dimensions and coefficients of them according to the data distribution of a subspace. That is, the key dimensions and their coefficients are not fixed, but modifiable in the whole data space.

The data partitioning strategy in the BM$^+$-tree can be described as follows: (1) the twin spaces are regarded together as a whole space and then it is partitioned with the m-RAD-2 way, as in the M-tree. As a result, two new sub-spaces are produced; and (2) each subspace is further partitioned into twin sub-spaces according to the selected binary hyperplane.

Figure 2 shows that to achieve the same level of data grouping, the M-tree needs three levels of partitioning, while the BM$^+$-tree and the M$^+$-tree only need two levels. Meanwhile, it is obvious that the data partition strategy in the BM$^+$-tree has the better clustering effect. The distance between objects along the data partitioning direction of the BM$^+$-tree is much bigger than that of the M$^+$-tree. This figure gives the intuition that the binary hyperplane based data partitioning can keep much more distance information.

3.3 Data Filtering Using Binary Hyperplanes

The filtering using binary hyperplanes is carried out on the basis of distance from an object to a hyperplane and the triangular inequality property. Comparing the cost of calculating the distance between two points in multidimensional space, the cost of computing the distance to a binary hyperplane is quite trifling. Some inactive subtrees can be filtered out according to the hyperplanes, thus avoiding some distance calculations. The process of filtering by binary hyperplane is not complex. We sketch the process below.

Let N_l and N_r be twin nodes, k_1 and k_2 be the key dimensions of the subspace, C_1 and C_2 be the coefficients of hyperplane in k_1 and k_2 dimensions respectively. Let L_{max} and R_{min} be the maximal value of the binary hyperplane for the left node and the minimal value of that for the right node respectively. Let $C = \sqrt{C_1^2 + C_2^2}$. Then the hyperplane of this subspace is as equation (1)

$$X_{k1} * C_1 + X_{k2} * C_2 = HP \tag{1}$$

Then, for the left part, HP is equal to L_{max}. While for the right, HP is R_{min}.

Suppose that the n dimensional query object is $O(x_1 \cdots x_n)$ and the search radius is r. Then, the distance from O to the left and right binary boundary hyperplanes can be expressed as (2)(a) and (b) respectively.

$$d_L = |HP - L_{max}|/C \quad (a) \qquad d_R = |HP - R_{min}|/C \quad (b) \tag{2}$$

For the filtering process, if $HP \geq L_{max}$, the query object is outside the area of the left; thus it is possible to filter out the left. Likewise, if $HP \leq R_{min}$, the right can be filtered out. Therefore, we only consider the following case.

$$d_L^{'} = (HP - L_{max})/C \quad (a) \qquad d_R^{'} = (R_{min} - HP)/C \quad (b) \tag{3}$$

If $d_L^{'} \geq r$, the left node does not contain any query results, and can be pruned. Likewise, if $d_R^{'} \geq r$, the right can be filtered out. Obviously, the cost to compute $d_L^{'}$ or $d_R^{'}$ is much less than computing the distance in the high dimensional space. It is similar to process k-NN following the filtering discussions above.

4 BM$^+$-tree

There are two types of node objects in a BM$^+$-tree: routing objects and leaf objects. Each leaf entry has the the same structure as that of M-tree. A routing object includes the following parts: the feature value of the routing object O_r; the covering radius of O_r, $r(O_r)$; the distance of O_r to its parent, $d(O_r, P(O_r)$; an array D_{NO} which contains two key dimension numbers; another array C containing the coefficients corresponding to the two key dimension numbers respectively; the pointers $lTwinPtr$ to the left twin sub-tree, and $rTwinPtr$ to the right twin; the maximal value of binary hyperplane in the left twin sub-tree M_{lmax} and the minimal value of binary hyperplane in the right M_{rmin}.

4.1 Building the BM$^+$-tree

To insert an object into BM$^+$-tree, the appropriate node should be found first by performing the subtree choosing algorithm . If the node is not full, the object can be inserted directly. If one of the twin nodes is full, the entries will be reallocated between the twins. If the twins are both full, they will be considered as a whole and split by performing distance based splitting and binary hyperplane splitting.

When a new node is inserted, how to choose this appropriate node is vital for the performance of the index. The subtree selection follows the optimal principal:

(1) Choosing the node of which the distance from query object to routing object is minimal if the covering radius of the subtree need not increase; (2) Choosing the subtree of which the covering radius increases most slightly, if no subtree can keep the same when an object is inserted; and (3)Trying to keep the gap between twins maximal while the subtree choosing between them are performed.

BM$^+$-tree grows in a bottom-up way by adopting a bottom-up split strategy. It shares the promotion strategy with M-tree and adopts a two-steps split strategy: first, splitting using the m-RAD-2 strategy; and second, splitting by the binary hyperplane. In the node splitting, the BM$^+$-tree adopts two strategies to choose binary hyperplane: (1) m-RAD-2 based binary hyperplane choosing strategy; and (2) Max-Distance based binary hyperplane choosing strategy.

```
SPLIT(entry(O_n), PEntry)
 1  Let S be a set, N_p be the parent node
 2  S←entries(PEntry→lTNode)⋃ entries(PEntry→ rTNode)⋃ entry(O_n)
 3  Reallocate a new node N'
 4  PromoteEntriesAndPartitionByDist
 5  ChooseBHAndPartitionByHyperplane
 6  if N_p is not a root
 7    then switch
 8           case TwinsOf(N_p) full   : Split
 9           case N_p is not full   : InsertIntoParentNode
10           case DEFAULT :    ReallocateTheTwins
11    else if N_p is full
12           then SplitRootByBHyperplane
13           else InsertIntoParentNode
```

```
CHOOSEBINARYHYPERPLANE(D_1NO, D_2NO, C_1, C_2)
1  GetTwoObjectsBymM_Rad
2  GetTwoMaxDiffByComputingAndSorting
3  if Diff[0] == Diff[1]
4    then GetTwoObjectsByMax_Distance
5         GetTwoMaxDiffByComputingAndSorting
6  SetMaxDimNoAndCoefToThem;
```

4.2 Query Processing

BM$^+-$tree supports two types of similarity search: r-range search and k-NN search.The range search starts from the root of the BM$^+-$tree and implements the process recursively until the leaf of the tree, and keeps all matching objects.

For a certain request, in non-leaf nodes, range search needs to perform two steps filtering. First, filtering according to the distances between objects among sibling nodes; and Second, filtering according to the binary hyperplane between twin nodes. For leave nodes, a two-steps filtering operations is used. The whole

process is similar to that used by M^+-tree. But it is different from the M^+-tree for filtering according to a binary hyperplane instead of a key dimension.

For k-NN search, BM^+-tree uses `PR`, a *priority queue* that contains pointers to active sub-trees, and `NN`, an array used to store the final search results.The BM^+-tree uses a heuristic criteria to select the priority node to access the priority queue and choose the next sub-tree to search. In this process, the binary hyperplane based filtering is used to reduce the I/O access and queue access of index. Due to our more effective data partitioning strategy, a binary hyperplane can be more discriminative than a key dimension.

The k-NN search using the BM^+-tree can be carried out according to the following steps: (1) keeping the root in the priority queue, `PR`, and the maximal distance in array `NN`; (2) choosing a priority node from `PR`; (3) searching the active subtree in the priority node. Here, the active subtree choosing follows (a) for a internal node, deciding the active subtree; (b) deciding when to update the result array `NN`; and (c)for a leaf node, deciding the matching objects; (4) repeating the subtree choosing process until the minimal distance of objects in `PR` from q is greater than `NN[`k`-1]` or `PR` is `null`. At the end of this query process, all pointers in `PR` will be removed and `NN` will be returned.

5 Performance Evaluation

This section presents an empirical study to evaluate the performance of BM^+-tree. Extensive experiments are conducted to compare the BM^+-tree with other competitors, including M-tree and M^+-tree. Our objectives of this study are:

(1) to study the scalability with number of dimensions;
(2) to evaluate the scalability with dataset sizes;
(3) to study the relative performance with the M-tree and the M^+-tree.

We use two types of real datasets: color histogram and color layout.

(1) The color histogram dataset contains vectors of Fourier coefficients of a set of images. We choose 10 sets of data with the dimensionality from 4 to 40;
(2) The color layout dataset comprises of color layout features from 20,000 images, which are 12-d data obtained by MPEG-7 feature extraction tool.

All the experiments were tested on a Pentinum IV 2.5GHz PC with 256MB of memory. All data are stored in an object database system XBase [12].

5.1 Effect of the Dimensionality

We performed experiments to evaluate the impact of dimensionality and compare the performance of BM^+-tree and that of M^+-tree. The experiments were performed on the 4 to 40-d histogram data, and the size of dataset is 20,000. As mentioned in M^+-tree[11], the number of queue accesses is also a key factor which affects the performance of k-NN search for these trees. Therefore, the

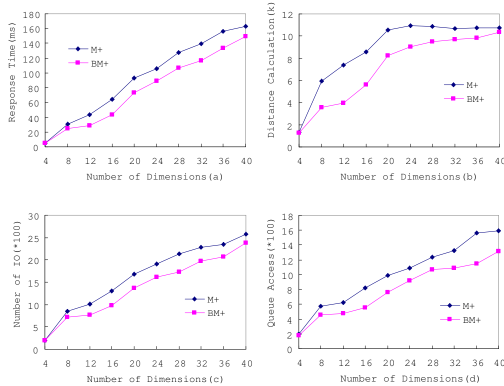

Fig. 3. The effect of dimensionality.

performance is measured by (1) the response time of a query; (2) the average number of distance calculations required to execute the query; (3) the average number of disk accesses for the query; and (4) the average number of queue accesses. The query is to find 10 nearest neighbors of the query object.

Figure (3) shows the performance of BM$^+$-tree comparing with that of M$^+$-tree. It is obvious that BM$^+$-tree responses more quickly than M$^+$-tree irrespective of the varying of dimensionality, and BM$^+$-tree outperforms M$^+$-tree by up to 35% . While the number of I/Os, distance calculations and queue accesses for BM$^+$-tree and M$^+$-tree also increase with increasing number of dimensions, those of BM$^+$-tree are growing at a much slower rate. BM$^+$-tree outperforms M$^+$-tree since the rotary binary hyperplane in BM$^+$-tree has a stronger filtering ability comparing against the key dimension in M$^+$-tree; consequently, some of sub-queries can be pruned and its search space covers fewer points. Moreover, Figure(3)(b) also shows the fact that, when the dimensionality is high enough, the number of distance calculations keeps steady irrespective of further increasing of dimensionality for a query has to scan the whole index for both of them.

5.2 Effect of Data Sizes

Now we compare the performance of the M$^+$-tree and the BM$^+$-tree with varying dataset sizes. The dataset size of the 10-d color histogram data was set from 10,000 to 90,000. Figure (4) shows the results.

From Figure (4), we can see that both the BM$^+$-tree and the M$^+$-tree incurred higher I/O cost and CPU cost with increasing data set sizes. As before, the performance of the BM$^+$-tree degrades much slower than that of the M$^+$-tree, and the BM$^+$-tree remains superior over the M$^+$-tree. Noticeably, compared with the M$^+$-tree, The BM$^+$-tree saves a large number of distance calculations, I/Os and queue accesses, which combined improve the query performance. This improvement based on using the BM$^+$-tree originates from the stronger filtering

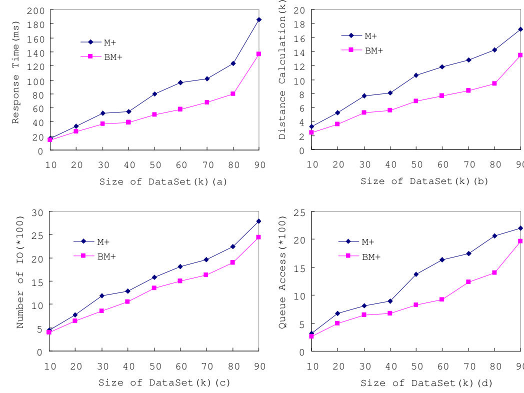

Fig. 4. The effect of dataset sizes.

ability of the rotary binary hyperplane, thus the number of IO can be reduced greatly. As a result, despite that, for a single operation, the comparison between binary hyperplane is a little bit slower than that of key dimension, the BM$^+$-tree still responses more quickly up to 40% than the M$^+$-tree for a k-NN search.

5.3 Comparison with M$^+$-tree and M-tree

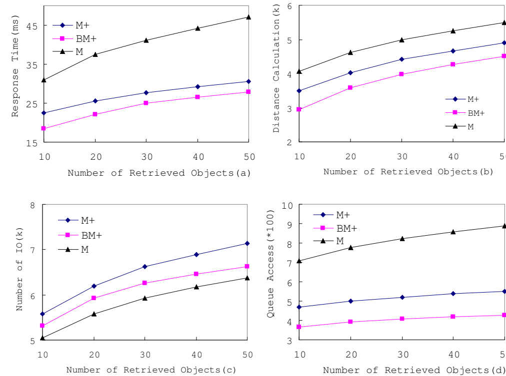

Fig. 5. k-NN search using BM$^+$-tree, M$^+$-tree and M-tree.

Next we examine the experiments to compare the k-NN search and range search using the BM$^+$-tree, the M-tree and the M$^+$-tree by using a 12-d real dataset which consists of the color layout information of 20,000 images. Figure

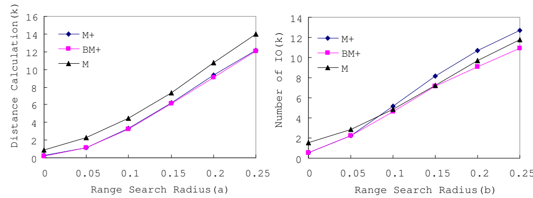

Fig. 6. Range search using BM$^+$-tree, M$^+$-tree and M-tree.

(5) shows the results. First, we can see that, from the response time, for k-NN search, the BM$^+$-tree performs the best, with the M$^+$-tree following it, and the M-tree performs significantly worse than the other two. Second, from distance calculations and queue access, the BM$^+$-tree outperforms its competitors to different extent, followed by the M$^+$-tree which is superior than the M-tree remarkably. Third, we note that the BM$^+$-tree and the M$^+$-tree need more I/O than the M-tree due to the heuristic criteria used in k-NN search to select the priority node to access the priority queue and choose the next sub-tree to search. The heuristic criteria set the search radius from maximal value, which leads to inferior filtering effect for both of them. Since the filtering ability of binary hyperplane degrades to a lower extent than that of key dimension, the BM$^+$-tree needs fewer I/O when comparing against the M$^+$-tree. All factors considered, the BM$^+$-tree outperforms the M$^+$-tree . For the M-tree, although it needs fewer I/O operations, the differences between the BM$^+$-tree and the M-tree is small (no more than 5%), while taking other decisive factors into account, the BM$^+$-tree saves up to 30% of distance calculations and half of the queue accesses, thus have much better query performance which is shown on Figure(5)(a).

Figure (6) shows the performance of the BM$^+$-tree, the M$^+$-tree and the M-tree for range search. From this figure, we can see that, comparing with the M$^+$-tree, the BM+-tree saves about 20% of I/Os while only 5% of distance calculation. Compared with the M-tree, the BM+-tree needs much less distance calculations, even a quarter of that for M-tree, while the slight improvement of I/Os. In additon, with the increase of search radius, the filtering ability of key dimension degrades rapidly, thus the number of I/Os needed by the M$^+$-tree increases noticeably and exceeds that of the M-tree. The BM$^+$-tree remains the least of I/Os for the stronger filtering ability of binary hyperplane. Therefore, the BM$^+$-tree always outperforms the M$^+$-tree and the M-tree taking the I/O and distance computation into account. The superiority of the BM$^+$-tree is clear.

6 Conclusions

In this paper, we have proposed an improved high dimensional indexing method, the BM$^+$-tree, which is a dynamically paged and balanced metric tree. This index method partitions a subspace into two non-overlapping twin subspaces

by utilizing two binary hyperplanes. An optimized method has been proposed for choosing binary hyperplanes for data partitioning. We have also given the algorithms to use the BM$^+$-tree based on rotary binary hyperplanes to perform effective filtering between twin nodes.

Experimental results obtained from using the two types of real datasets show that the BM$^+$-tree has a significantly better query processing performance. Comparing with the M-tree, the query efficiency has been improved by more than 30% on average, and up to 10 times in some cases. Comparing with the M$^+$-tree, the BM$^+$-tree reduces about 10-20% of I/O operations, about 5% distance calculations, and about 20% of queue access for K-NN search queries.

Acknowledgment: This research was supported by the National Natural Science Foundation of China (Grant No. 60273079 and 60473074), the Foundation for University Key Teacher and the Teaching and Research Award Program for Outstanding Young Teachers in High Education Institution of Chinese Ministry of Education, and the Australian Research Council (Grant No. DP0345710).

References

1. N. Berkmann, H.-P. Krigel. R. Schneider, and B. Seeger. (1990) "The R*-tree: an Efficient and Robust Access Method for Points and Rectangles" ACM SIGMOD 90, pages 322-331.
2. N. Katayama and S. Satoh. (1997) "The SR-tree: an Index Structure for High-dimensional Nearest Neighbor Queries ACM SIGMOD 97, pages 369-380.
3. D. A. White and R. Jain.(1996) "Similarity Indexing with the SS-tree" ICDE 96, pages 516-523.
4. K.-I. Lin, H. V. Jagadish, and C. Faloutsos. (1994)"The TV-tree: An Index Structure for High-Dimensional Data" VLDB Journal, Vol. 3, No. 4, pages 517-542.
5. J. K. Uhlmann.(1991) "Satisfying General Proximity/ Similarity Queries with Metric Trees", Information Processing Letters, vol 40, pages 175-179.
6. P. Zezula, P. Ciaccia, and F. Rabitti.(1996) "M-tree: A Dynamic Index for Similarity Queries in Multimedia Databases" TR 7, HERMES ESPRIT LTR Project.
7. T. Bozkaya, M. Ozsoyoglu.(1997) "Distance-based Indexing for High-dimensional Metric Spaces" ACM SIGMOD 97, page 357-368.
8. P. Ciaccia, M. Patella, P. Zezula.(1997) "M-tree: An Efficient Access Method for Similarity Search in Metric Spaces" VLDB 97, Greece.
9. M. Ishikawa, H. chen, K. Furuse, J. X. Yu, N. Ohbo (2000) "MB$^+$tree: a Dynamically Updatable Metric Index for Similarity Search" WAIM 2000, page 356-366.
10. C. Traina Jr, A. Traina, B. Seeger, C. Faloutsos (2000) "Slim-trees: High Performance Metric Trees Minimizing Overlap Between Nodes. EDBT 2000, pages 51-65.
11. X. Zhou, G. Wang, J. X. Yu, G. Yu (2003) "M$^+$-tree: A New Dynamical Multidimensional Index for Metric Spaces" Proc of 14th Australasian Database Conference (ADC2003), pages 161-168.
12. G. Wang, H. Lu, G. Yu, Y. Bao (2003). "Managing Very Large Document Collections Using Semantics." Journal of Computer Science and Technology, 18(3): 403-406.
13. C. Böhm, S. Berchtold, D. A. Keim (2002), "High-dimensional Spaces - Index Structures for Improving the Performance of Multimedia Databases". ACM Computing Surveys, 2002.