NAIST-IS-DT9661201

Doctor's Thesis

Studies on Spatial Index Schemes for High-Dimensional Data Sets

Yasushi Sakurai

September 30, 1999

Department of Information Systems
Graduate School of Information Science
Nara Institute of Science and Technology

Doctor's Thesis
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
DOCTOR of ENGINEERING

Yasushi Sakurai

Studies on Spatial Index Schemes for High-Dimensional Data Sets*

Yasushi Sakurai

Abstract

This dissertation focuses on similarity search for high-dimensional data. Fast content-based retrieval is necessary to provide excellent human interface for large-scale multimedia databases. Multimedia database systems are required to support not only retrieval using attached text description but also content-based retrieval using feature vectors, which are extracted from multimedia data. For example, content-based retrieval in image database systems needs selecting one or more images which have features similar to the feature of a query image. Many applications with multimedia data retrieval require spatial search techniques in various dimensions since retrieval processing using feature vectors involves high cost, especially, for large data set.

In this research, a new indexing scheme, the Subspace Coding Method (SCM), is presented. The motivation of the SCM is based on the comparison and analysis of the two best access methods proposed so far: the SR-tree and the VA-File. Since no result on the comparison between the SR-tree and the VA-File is available, this dissertation presents experiments comparing these two access methods

*Doctor's Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DT9661201, September 30, 1999.

using synthetic and real data. The result shows that the SR-tree offers better performance up to 40 dimension. However, as dimensionality increases, large volume of entries in non-leaf nodes causes fewer fanout, which consequently degrades the search performance of the SR-tree.

Based on the analysis of the experimental result, the SCM, a new indexing scheme applicable to any tree indices employing MBR (Minimum Bounding Rectangle) and/or MBS (Minimum Bounding Sphere) is introduced. The basic idea of the SCM is the introduction of *Virtual Bounding Rectangle (VBR)* and *Virtual Bounding quasiSphere (VBS)*, which contains and approximates MBR and MBS, respectively. And also, the subspace code compactly represents VBRs/VBSs, hence allows larger fanout of non-leaf nodes in indices. Unlike the approximation of absolute vector positions used in the VA-File, the subspace code represents the relative position of VBRs/VBSs in terms of parent's VBR/VBS. This feature is effective especially for non-uniformly distributed vectors which are commonly found in real applications. Nearest neighbor search is performed using the tree index of which non-leaf nodes contain VBRs/VBSs.

The experimental results using real data set demonstrate the effectiveness of the SCM. When applied to the SR-tree, the SCM outperforms both the original SR-tree and the VA-File in all range of dimensionality up to 56 dimension, which is the highest dimension in the experiments. The SCM achieves 71.9 % (74.7 %, resp.) savings in page accesses compared to the VA-File (the original SR-tree, resp.) for 56-dimensional vector data.

Keywords:

similarity search, high-dimensional data, spatial index scheme, multimedia database, subspace code

Contents

List of Figures

List of Tables

Acknowledgments

I am grateful to Professor Shunsuke Uemura for being my adviser and mentor. His technical and general guidance has assisted me in my personal and professional growth.

I would like to thank Professor Minoru Ito for providing me with valuable and beneficial suggestions in this research. The technical guidance and constant encouragement of Professor Naokazu Yokoya through the master course and the doctor course, is gratefully acknowledged.

I wish to thank Associate Professor Masatoshi Yoshikawa for continuous guidance, constructive criticisms and numerous suggestions. In addition, his inspiring thoughts have helped me in conducting my research.

I wish to express my thanks to Media Creation Project Manager Noboru Sonehara and Video Interface Group Leader Haruhiko Kojima of NTT Cyber Space Laboratories for valuable advice. My thanks also go to all my officemates in NTT Cyber Space Laboratories for useful support.

My thanks are due to Dr. Yoshiharu Ishikawa of University of Tsukuba for giving me the motivation of this research and precious comments. I would like to thank Dr. Toshiyuki Amagasa and Dr. Kenji Hatano of Uemura Laboratory at Nara Institute of Science and Technology for valuable comments.

Finally, I wish to express my gratitude to all members of Uemura Laboratory for considerable discussion and help.

Chapter 1

Introduction

1.1 Data Retrieval in High-Dimensional Space

Fast content-based retrieval is necessary to provide excellent human interface for large-scale multimedia databases. It needs selecting one or more objects in multimedia database which have features similar to the feature of query data. In many cases, feature vectors are extracted from multimedia data to perform content-based retrieval. For instance, features extracted from images include color, texture, structure and so on. Image database systems are required to support not only retrieval using attached text description but also content-based retrieval based on similarity search using these features. Since retrieval of feature vectors incurs high cost for high-dimensional and large data set [NN96], the need for spatial indices and search methods with the object of efficient data retrieval is envisaged. Developing techniques for storage and retrieval of high-dimensional data is an important research issue. This dissertation is focused on similarity search for multi-dimensional data, especially, high-dimensional data of which search cost is higher.

Many applications of multimedia data retrieval systems require spatial search techniques in various dimensions such as GIS (geographical information system) [Li97] which handles 2-D or 3-D data, video library system with feature vectors of which dimensionality is ten or tens [WKSS96], or image retrieval system by hundreds dimension [Dim97]. Above all, on account of the reduction of computational cost and the raise of object recognition ratio, many recognition methods [MN95] [PMS94] [TP91] [MCM92] or systems [WKSS96] using feature vectors of which dimensionality is ten or tens are presented. Therefore, the main issue of this dissertation is indexing and searching for feature vectors of which dimensionality is ten or tens.

Various spatial indices [SRF97] [GG98] have been proposed so far, however, they have various drawbacks. In this dissertation, a high-speed spatial search method is proposed to accelerate search performance.

1.2 Contribution of the Dissertation

In this dissertation, a new indexing scheme named Subspace Coding Methods (SCM) [SYU99] [SYU98] is proposed. The SCM is applicable to any tree indices employing MBR (Minimum Bounding Rectangle) and/or MBS (Minimum Bounding Sphere).

The introduction of the SCM is motivated by the comparison and analysis of two excellent methods, the SR-tree and the VA-File. The SR-tree [KS97] has a unique feature in that it uses both MBRs and MBSs, and reported to offer good performance. The other one is The VA-File [WSB98] which is a simple yet powerful scheme. In the VA-File, data objects are approximated by the cell beforehand, and then this approximation file is used for search to select out candidate data objects. Since no result on the comparison between these two access methods is available, the experiments comparing the two access methods are performed.

The result shows that the SR-tree offers better performance than the VA-File for real data (color histogram vectors of images) up to 40 dimensions. However, as dimensionality increases, the search performance of the SR-tree degrades. The analysis of further experimental results reveals that large size of entries in non-leaf nodes[1] in high dimension causes fewer fanout and larger number of non-leaf node accesses, which results in the performance degradation of the SR-tree.

To overcome this drawback, the SCM, which is a general indexing scheme based on an approximated and relative representation of MBRs/MBSs is proposed in this dissertation. The basic idea of the SCM is the introduction of *Virtual Bounding Rectangle (VBR)* and *Virtual Bounding quasiSphere (VBS)*, which contains and approximates MBR and MBS, respectively. And also, the *subspace code* which compactly represents VBRs/VBSs is introduced. Hence, the SCM allows larger fanout of non-leaf nodes in indices. Unlike the approximation of absolute vector positions used in the VA-File, the subspace code represents the relative position of VBRs/VBSs in terms of parent's VBR/VBS. This feature is effective especially for non-uniformly distributed vectors which are commonly found in real applications. Spatial search is performed using the tree index of which non-leaf nodes contain VBRs/VBSs. This dissertation gives the algorithms for search, insertion and deletion for the SCM. The SCM is not "a yet another tree index", but a general indexing scheme applicable to any tree indices employing MBR and/or MBS (e.g. R-tree, R*-tree, X-tree, SS-tree, SR-tree). Since non-leaf nodes of the SR-tree have both rectangles and spheres, the SCM will be most effective when applied to the SR-tree.

In this dissertation, performance evaluations using both synthetic and real

[1]In this dissertation, leaf node means a node which contains data objects on the lowest level of tree structure, and non-leaf node means other nodes.

data are presented. The result demonstrates the effectiveness of the SCM in high-dimensional nearest-neighbor search. When applied to the SR-tree, the SCM outperforms both the original SR-tree and the VA-File in all range of dimensionality up to 56 dimension, which is the highest dimension in the experiments. The SCM achieves 71.9 % (74.7 %, resp.) savings in page accesses compared to the VA-File (the original SR-tree, resp.) for 56-dimensional vector data. As far as we know, 56 is the highest dimension of real data used for performance evaluation in high-dimensional access methods with the only exception of 100 dimensional eigen-face data used in [WJ96b].

1.3 Outline of the Dissertation

The remainder of this dissertation is organized as follows. In Chapter 3, the background of the research is presented. Chapter 4 presents a comparison and analysis of the SR-tree and the VA-File based on experiments using real and synthetic data. Based on the analysis, Chapter 5 describes the motivation, the definitions and the algorithms of the proposed method, the subspace coding method. Chapter 6 presents the result of performance evaluation of the proposed methods and conventional access methods, and the discussion. Finally, Chapter 7 presents final discussion and concludes the dissertation.

Chapter 2

Experimental Configuration

The experimental evaluation for the techniques and the algorithms shown in this dissertation, is presented in each of Chapter 3, 4 and 6. In this chapter, the experimental conditions on this dissertation are described.

2.1 Characteristics of the Data Sets

In this dissertation, the performance evaluation is based on three kinds of data sets, real data sets, uniformly distributed data sets and cluster data sets. The real data sets consist of feature vectors that are hue histograms extracted from images using in [SAT99] and the uniformly distributed data sets are uniform random point sets distributed in the range [0.1) on each dimension. As non-uniformly distributed data set, not only real data sets but also cluster data sets are created and used for the evaluation. For cluster data sets, the number of cluster is 100 in each data set and the center of cluster is distributed uniformly in the range [0.10). In addition, as the number of objects is N, $N/100$ objects are gathered according to Gaussian distribution around the center of cluster. Figure 2.1 shows the distribution of cluster data for two dimension as the size of data

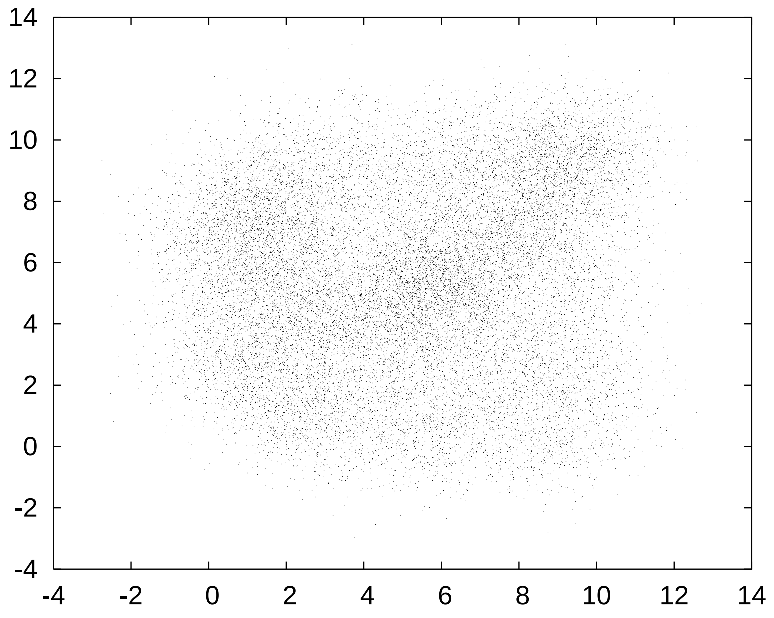


Figure 2.1. Distribution of cluster data for two dimension.

set is 20,000.

Table 2.2, Table 2.1 and Table 2.3 show the average and the variance for the distance between each nearest neighbor objects in three data sets as the size of all data sets is 100,000. Unlike real data sets and uniformly distributed data sets, cluster data sets are not distributed in the range [0.1) on each dimension. Therefore, on Table 2.3, the average and the variance for nearest neighbor distance are calculated by the location of each data object whose distribution is normalized in the range [0.1) on each dimension.

These tables reveal the difference between the data sets as regards data distribution. As shown in Table 2.1, nearest neighbor distance increases as dimensionality grows for uniformly distributed data sets. Despite the fact that the distance of non-uniformly distributed data set also rises as shown in Table 2.2 and Table 2.3, the distance is extremely small when compared with uniformly distributed

Table 2.1. Average and variance of nearest neighbor distance in uniformly distributed data sets.

Dimensionality	Average	Variance
4	0.034690	0.000098
8	0.200570	0.001122
16	0.596733	0.003665
24	0.948192	0.005843
32	1.255998	0.007390
40	1.531881	0.007947
48	1.786691	0.008196
56	2.012652	0.008377

Table 2.2. Average and variance of nearest neighbor distance in real data sets.

Dimensionality	Average	Variance
4	0.004598	0.000023
8	0.023116	0.000326
16	0.042783	0.000654
24	0.053278	0.000724
32	0.058653	0.000709
40	0.062080	0.000668
48	0.063145	0.000627
56	0.063707	0.000616

Table 2.3. Average and variance of nearest neighbor distance in cluster data sets.

Dimensionality	Average	Variance
4	0.020631	0.000076
8	0.077410	0.000413
16	0.162310	0.000629
24	0.231455	0.000723
32	0.291108	0.000761
40	0.342610	0.000787
48	0.389892	0.000802
56	0.433240	0.000854

data sets. Especially, for real data (i.e. histogram data), the average is small, in addition, the variance for nearest neighbor distance is also small. Table 2.2 shows that neighbor objects are close each other for most of objects in the real data sets.

2.2 Experimental Conditions for Search Methods

The dimensionality for synthesis and real data varies from 4 to 56. The size of the data sets is 100,000. Page size is 4KB.

On search performance, the evaluation of page accesses and CPU-time are based on the average number for 1,000 query points. 20-nearest neighbor queries are used, and query data is not the point data included in indices, that is, query points are generated randomly independent of data points. CPU-time was measured on SUN UltraSPARC-II 296MHz. The search performances of R-tree family are measured by using the algorithm presented in [HS95] since the algorithm in [HS95] outperforms the branch-and-bound R-tree traversal algorithm [RKV95] as shown in Section 3.4.2.

As for the insertion, the average of 1,000 insertion cost is measured. These 1,000 objects are inserted into data sets of the size from 20,000 to 100,000.

Chapter 3

Background

3.1 Content-Based Retrieval for Multimedia Databases

Content-based retrieval in multimedia databases is important research issue [Fal96]. It needs selecting one or more objects in multimedia database which have exact or similar feature extracted by data processing. Examples of queries include '*collect pictures with similar color distribution from image database*', or '*find beach scenes in video databases using picture or sound*'.

Figure 3.1 shows a simple example for content-based retrieval system. Multimedia data set is transformed into feature data set by feature extraction, and then index structure is constructed using the feature data set in advance of query from users. On the other hand, through the user interface, query vector is calculated from the content of query. The vectors that are similar to the query vector, are collected efficiently by search algorithm using the index structure. Finally, required multimedia data are reported based on the collected vectors. In these descriptions, the scope of spatial search methods includes the indexing and the

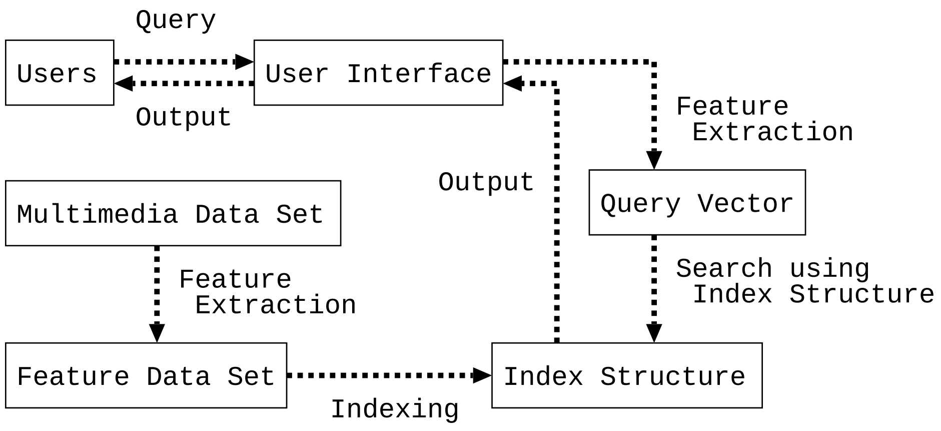


Figure 3.1. An example of content-based retrieval system.

search using the index structure.

Many multimedia database systems, such as QBIC [NBE+93] [FBF+94] [FSA+95], CORE [WNM+95], Virage [BFG+96] [HGH+97], present interfaces and mechanisms of content-based retrieval. Various spatial queries have been presented for adapting to many kinds of applications. And also, in order to accelerate search performance, many multi-dimensional search methods have been proposed so far.

Many multimedia retrieval systems include different kinds of feature vectors. For instance, QBIC presents retrieval services based on color, texture and shape for image or video databases, and also all vectors are stored in one index structure together [NBE+93]. In [TY98], an index is created for each feature in order to perform spatial search based on weighted Euclidean distance. Pruning strategy is performed in each index based on assigned weight for the index. Unlike the algorithm using more than one index, this dissertation is focused on algorithms for efficient spatial search using a single index.

3.2 Queries for Multi-Dimensional Data

Many retrieval systems or applications with spatial data have need of various search operations [GG98] [LJF94] [Fal97]. The following are several examples of spatial query relative to the condition which data objects are points:

(1) Exact match query, Point query

Find all objects in the database that are equal to a given query object. More formally, let q be an query point and o be an object included in the database, find all objects o:

$$Query_{exact}(q) = \{o | Dist(o, q) = 0\},$$

where $Dist$ is the function reporting the distance for two given points. For example, *find the building in a given location from geographical database.*

(2) Region query, Range query

Find all objects in the database that are included in a given region. More formally, given a region R, find all objects o:

$$Query_{region}(R) = \{o | o \in R\}.$$

In particular, range query is the operation reporting all objects that are within the distance d from a given query point:

$$Query_{range}(q, d) = \{o | Dist(o, q) \leq d\}.$$

For example, *find all buildings within a given area; find all pictures that look like a sunset.*

(3) Nearest neighbor query

Find all objects in the database that are closer than the other objects [FBY92]. That is, given a query point q, find k objects o:

$$Query_{NN}(q,k) = \{o | \exists d \forall o' Dist(o,q) \le d \le Dist(o',q)$$
$$\wedge Number_{object}(Query_{range}(q,d)) = k\},$$

where $Number_{object}$ is the function reporting the number of objects. As $k=1$, the formula is simplified:

$$Query_{NN}(q) = \{o | \forall o' Dist(o,q) \le Dist(o',q)\}.$$

For example, *the image in the database that is most similar to the query image which shows Mt. Fuji.*

(4) Spatial join

Given a spatial predicate θ, find all object pairs that satisfy the predicate θ from two data sets [Gue93] [BKS93] [BKSS94] [MP99]:

$$A \underset{i\theta j}{\bowtie} B = \{(o,o') | o \in A \wedge o' \in B \wedge \theta(o_i, o'_j)\},$$

where A and B are given relations, and also $i\theta j$ is the spatial operator for the i-th entry of A and the j-th entry of B. For example, *given a map, find all houses that are within 500 meters from a railroad station.* This query example is based on two relations, *house* and *station*.

(5) Sub-pattern matching

Find all objects that include a given sub-pattern [Fal96] [Fal97]. That is, given a query pattern q that has k or more than k entries, find all objects that include q or a part of q which has k continuous entries:

$$Query_{sub-pattern}(q,k,\epsilon) = \{o | o_{\delta 1} \in o \wedge q_{\delta 2} \in q \wedge Number_{entry}(\delta 1) =$$
$$Number_{entry}(\delta 2) = k \wedge Dist(o_{\delta 1}, q_{\delta 2}) \le \epsilon\},$$

where $Number_{entry}$ is the function reporting the number of entries, and $\delta 1$ and $\delta 2$ has k continuous entries. In this formula, the query is allowed a tolerance ϵ. For example, *find images that contain a given sub-pattern.*

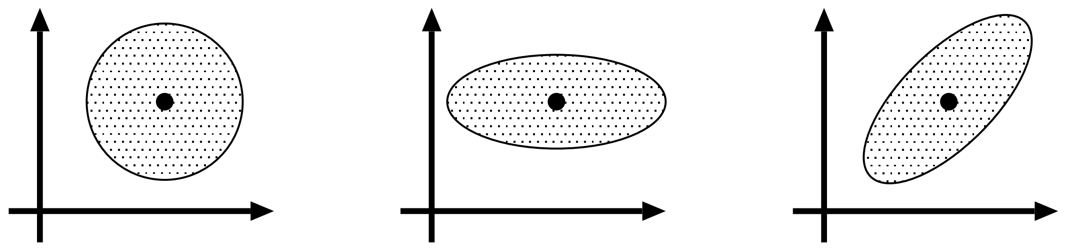
All these queries are fully workable on the structure of the method which is proposed in the dissertation. In the queries, the performance evaluation is based on nearest neighbor query which is one of the most useful query for many applications and requires relatively high search cost.

3.3 Distance Functions

Many multimedia applications use search algorithms based on Euclidean distance, on the other spatial queries based on general quadratic distance are useful. For Euclidean distance, the search area is formed the shape of a sphere of which center point is the query point, moreover, Euclidean distance is the independence of dimension. By contrast, in order to reflect correlations of dimension, a distance model that has the power of model dependencies between different components of feature, is provided by elliptical distance functions [SK97]. An elliptical distance is a distance that has equi-distance surfaces with ellipsoidal shape [Ish99]. That is, for given two N-dimensional vectors x and y, and $N \times N$-matrix A, the general quadratic distance between x and y is described as follows:

$$d_A^2 = (x - y) \cdot A \cdot (x - y)^T.$$

This formula represents the Euclidean distance function when A is equal to the identity matrix, and also A is called similarity matrix [SK97]. And also, the formula represents the weighted Euclidean distance when the similarity matrix A is diagonal. In the case that A is positive definite matrix, the formula includes the elliptical distance between x and y on A (cf. Figure 3.2).



(a) Euclidean distance (b) Weighted Euclidean distance (c) Elliptical distance

Figure 3.2. Query ranges for distance functions.

The general quadratic distance functions mentioned above are required for various applications. For instance, in [HSE+95], image retrievals based on general quadratic distance with arbitrary similarity matrix are performed. The similarity matrix is determined by the users on each query, however, it is difficult for them to decide on the similarity matrix, especially, the correlations of dimension or the weight of dimension. MindReader [ISF98] supports application users in the attempt to determine their favorite similarity matrix using relevance feedback techniques [Har92] in the information retrieval field.

Since spatial queries based on general quadratic distance are useful [HSE+95] [ISF98], search algorithms for efficient processing of ellipsoid queries using tree structures have been proposed [SK97] [ABKS98] [Ish99]. The search cost on [SK97] is seriously high since testing whether a data object is contained in the query ellipsoid requires $O(n^2)$ time, where n is the dimensionality, and testing the intersection of a MBR in the index structure and the query ellipsoid takes $O(n^2 \cdot i)$ for a small iteration factor i. The algorithms of [ABKS98] and [Ish99] improve in search cost by using an approximation-based approach. These algorithms can partly avoid calculating elliptical distance and lower search cost for CPU-time by using approximation of elliptical distance. The Query region of the approximation distance by the algorithms contains the exact query region, thus the algorithms guarantee no false drops. The algorithm of [ABKS98] approximates query region using MBR and MBS that contain the exact query region. And also, the algorithm of [Ish99] approximates using ellipsoidal shape inscribed in the MBS that approximates the exact query region. In [Ish99], the computation of the ellipsoidal shape is simplified in connection with the dimensions of which eigen-value [Oja83] [And84] [Fuk90] is relatively large.

The performance evaluation is based on Euclidean distance, however, all these spatial search algorithms based on elliptical Distance are fully workable on the structure of the method which is proposed in the dissertation.

3.4 Spatial Search Methods

3.4.1 Classification of Spatial Search Methods

In order to perform the queries mentioned above efficiently, various spatial indices [SRF97] [GG98] have been proposed so far. In these methods, the grid file [NHS84], the kdb-tree [Rob81] and the R-tree [Gut84] are one of the methods which was the base for all further developments in the field of spatial search. The field of spatial search can be classified in two broad categories, cell-based files and tree structures.

Cell-Based Files

As cell-based file, first, the grid file is discussed. The grid file [NHS84] using hasing technique covers whole data space with grid cells that are partitioned by each dimensional axis. Each dimensional axis is divided by the scale for itself and the division leads to generated grid cells as a result. For non-uniformly

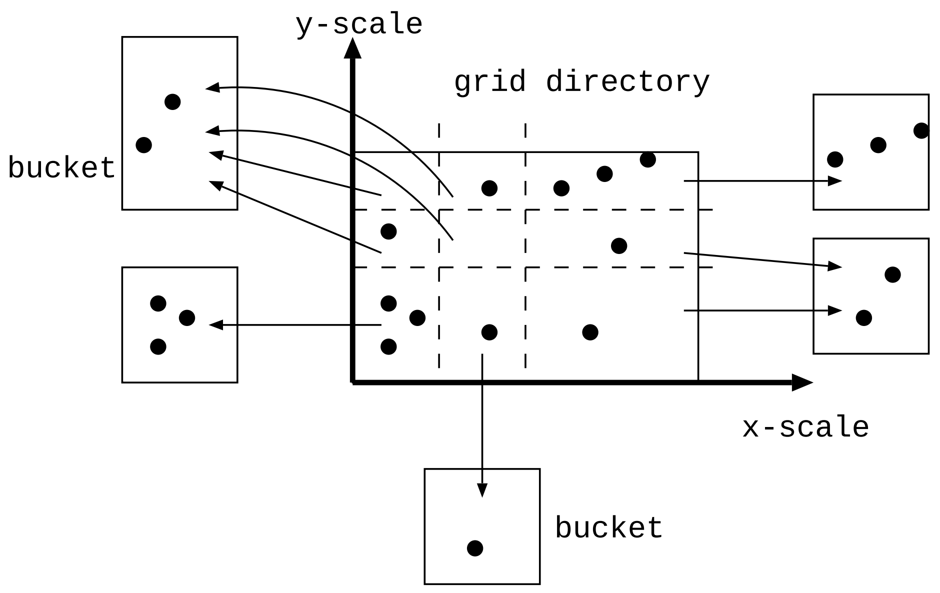


Figure 3.3. The grid file.

distributed data set, split for grid block can be frequent in dense space and the volume of grid cells in the space will became smaller than the other cells. On the other hand, volume of cells gets larger in sparse space.

A grid directory consists cells, and each cell is associated with one bucket. There is one-to-one correspondence between bucket and disk page block which contains data records. One or more of adjacent cells are stored in one bucket for efficient page utilization. Figure3.3 shows an example of grid file in two-dimensional space. The grid directory is divided into nine grid cells by the x-scale and y-scale, and also, each bucket contains one or more than one cell in this figure.

On the grid file, split for bucket is performed by hyperplane. The problem in the grid file is that split is not local operation, that is, large part of grid directory can be influenced by bucket split and it can lead to superlinear directory growth.

The twin grid file [HSW88b] [HSW88a] is one of the variants [Fre87] [KS88]

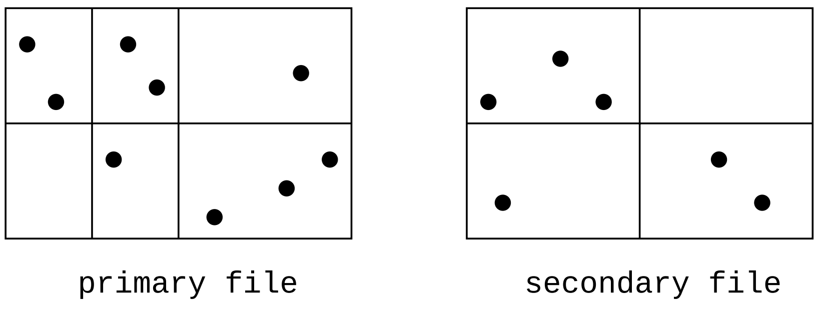
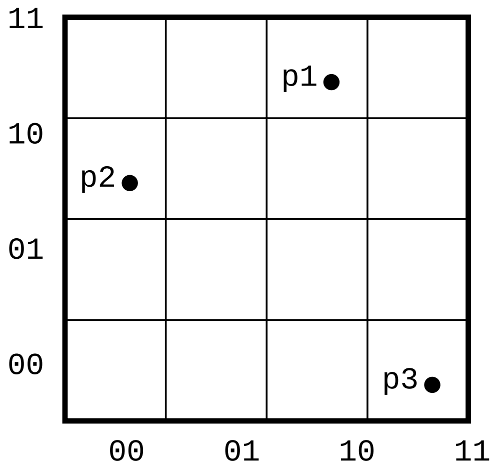


Figure 3.4. The twin grid file.

[Ouk85] [RLS93] on the grid file. In this method, two grid files, primary file and secondary file are used and set of points are distributed among these grid files in order to increase space utilization as shown in Figure 3.4. As insertion or deletion occurs, objects can be redistributed among two files. The redistribution leads to lowering the total number of buckets and higher space utilization. In [HSW88b], 90 % space utilization for the twin grid file and 69 % utilization for the traditional grid file are presented. This strong point, high utilization is useful for range queries.

Cell-Based File for Nearest Neighbor Queries

Grid file family is not suitable for nearest neighbor query, in addition, especially, they do not work in high-dimensional space. The VA-File [WSB98] using cell block attacks this problem. The VA-File divides the data space into cells and allocates a bit-string for each cell. As shown in Figure 3.5, vectors inside a cell are approximated by the cell, and the VA-File itself is simply an array of these geometric approximations. For nearest neighbor search, the entire VA-File is scanned to select out candidate vectors, then those candidates are verified by visiting the vector files.



	Approximation	Vector
p1	0.65 , 0.85	10 , 11
p2	0.16 , 0.60	00 , 01
p3	0.92 , 0.10	11 , 00

Figure 3.5. The VA-File.

In concrete, the VA-File has three steps for searching data objects.

(1) Compressing vector data

At first, the code of length l_i is assigned for each dimension i in n-dimensional space. The whole search space is divided into 2^l cells which $l = \sum_{i=1}^{n} l_i$, and all data objects are stored in the corresponding cell. As shown in Figure 3.5, the approximate value is calculated for the vector of each point.

(2) Filtering step

The entire VA-File is scanned in order to select out candidate vectors for nearest neighbor query, and upper and lower bounds on the distance between the query point and each cell are computed as the approximate distance for the distance between the query point and an object in the cell. Objects in the cell of which lower bound is larger than the smallest upper bound found in this step, are excluded from the next step.

(3) Accessing candidate objects

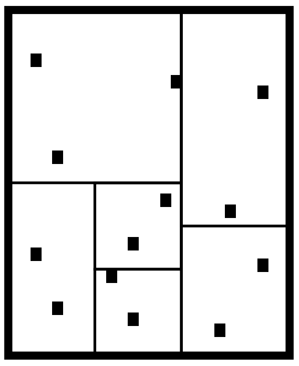


Figure 3.6. The kdb-tree.

Only candidate objects that remain from the filtering step are verified that whether the object is nearest neighbor for the query point by visiting the vector files. In the mechanism of the VA-File, all candidate objects do not have to be visited. The objects are accessed in increasing order of their lower bound on the distance between the query point and the corresponding cell, and also the mechanism stops as the distance of lower bound to the corresponding cell exceeds the nearest distance to accessed vector.

In these steps, k-th nearest neighbor query can be also performed in a similar way.

Tree Structures without Overlapping Regions

The kdb-tree [Rob81] is a hierarchical tree structure using the ideas of the B+-tree [Com79] and the kd-tree [Ben75], and focused on indexing point object and point query. This structure is constructed by the node split policy which current region is divided into two child regions without dead space and overlap of child regions as shown in Figure 3.6. Since the kdb-tree does not allow overlap between regions of nodes, the structure selects only one search path for point

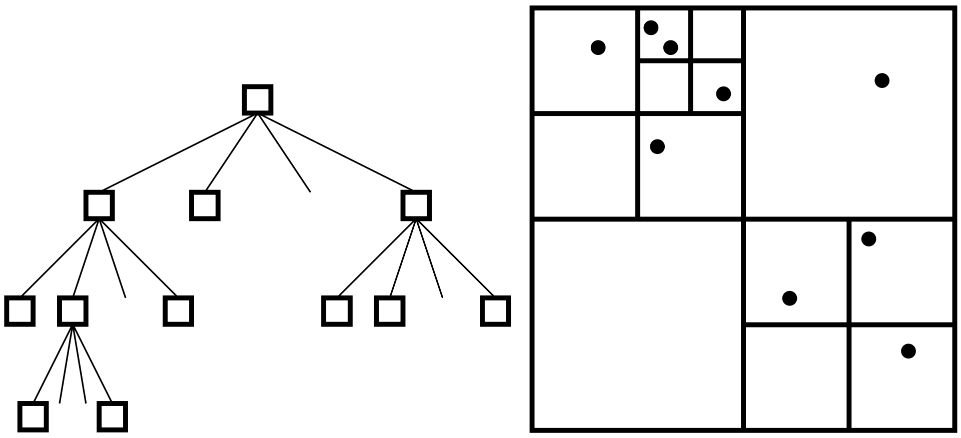


Figure 3.7. Structure of the PMR quadtree.

query, and is suitable for point query or region query which search region is quite small. However, the utilization of this structure is low because of less flexibility relative to node split, thus, the kdb-tree does not work on nearest neighbor query.

The PMR quadtree [NS86] [NS87] [HS92] is one of quadtree variants [Sam84] [SW84] [SW85]. This method is also tree structure such as the kdb-tree, however, the structure is not balanced. Each block of the PMR quadtree is square for two-dimension, cube for three dimension, or hypercube for high dimension. As a node overflows, the block is divided into 2^n child blocks of which the size is equal, where n is dimension. Figure 3.7 shows the structure in two-dimensional space. For this structure, the depth of tree can be large in a dense space. This property causes considerable inferior performance as using non-uniformly distributed data set, thus, the structure is not fit for real data handling system such as an image database.

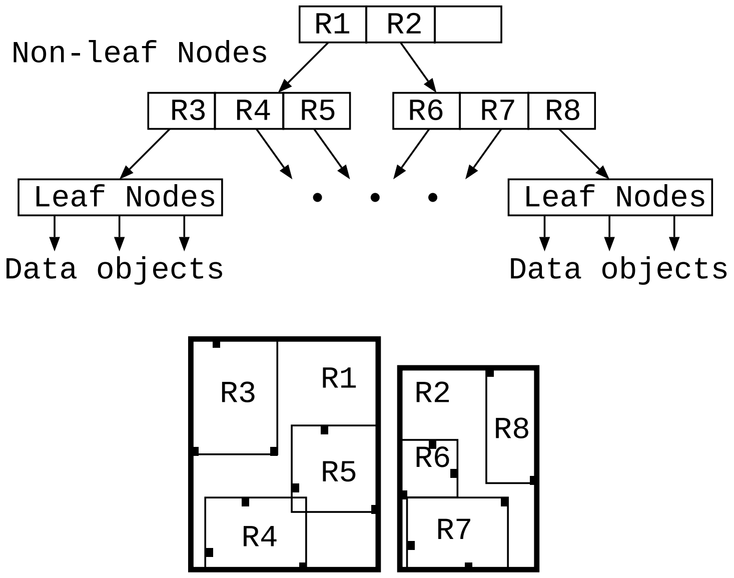


Figure 3.8. Structure of R-tree family using only MBR.

Tree Structures with Overlapping Regions

The R-tree [Gut84] has excellent mechanism for the circumvention of the problem on the kdb-tree or the PMR quadtree. The R-tree is hierarchical structure, and uses MBRs (Minimum Bounding Rectangles) as bounding regions for partitioning data objects. Unlike the kdb-tree or the PMR quadtree, the R-tree allows the overlap between regions of node as shown in Figure 3.8. The utilization is improved owing to overlapping MBR, and also, the flexible node split policy in the R-tree leads to higher search performance for region query.

It is composed of MBRs organized as tree structure in which geometrical objects are packed in leaf nodes. Leaf nodes contain entries of the form

$$\begin{aligned} LeafNode &= (\varepsilon, Record_i) \;\; (i = 1, \ldots, \varepsilon) \\ Record_i &= (C, object - identifier), \end{aligned}$$

where ε is the number of entries in the leaf, and is bounded by minimum m and maximum $M (m \leq \varepsilon \leq M)$. C is the MBR which approximates a geometrical object or a spatial vector which represents a point. $object - identifier$ refers to contents of object. Non-leaf nodes contain entries of the form

$$\begin{aligned} Non-leafNode &= (\varepsilon, Record_i) \;\; (i = 1, \ldots, \varepsilon) \\ Record_i &= (C, child-pointer), \end{aligned}$$

where C is the MBR of a child node, and $child - pointer$ refers to the child node.

The R-tree has had many variants and search algorithms using these structures. Since these variants have relation to the method proposed in the dissertation, Section 3.4.2 presents detail explanation.

3.4.2 R-tree Family

Index Structures

The mechanisms of the R-tree are introduced in many structures, thus, the term, R-tree family is used to generally represent these variants. Many index trees on R-tree family have been proposed so far. They include the R-tree [Gut84], the R*-tree [BKSS90], the Hilbert R-tree [KF94], the X-tree [BKK96], the SS-tree [WJ96b], the SR-tree [KS97] and so on.

In the family, the R*-tree [BKSS90] has higher capability in searching than the original R-tree according to improvement of insertion algorithm, and it is one of the most common and successful structures for multi-dimensional searching, especially, for low dimension. Although the structure of the R*-tree is similar to that of the R-tree, the insertion algorithms are introduced newly for enhanced search performance. The insertion algorithm of the R-tree is designed to minimize area of MBR, whereas that of the R*-tree aims at three targets:

(1) The overlap between MBRs is minimized.

(2) The margin of MBRs is minimized.

(3) The area of MBRs or dead space is minimized.

In particular, small overlap leads to the improvement of pruning for searching.

The forced reinsert, the algorithm which is to delete overflowing node and reinsert the orphaned entries in the corresponding level, invokes local reorganization of the tree structure. The reorganization keeps small overlap and high utilization, and also it supports good search performance. Since the forced reinsert mechanism is reasonable, it is used for most of later search method in R-tree family. Concretely, in the algorithm for overflow handling, if the level of overflowing node is not the root level and this overflow handling is the first call on the level during the insertion of one object, a portion of entries in the node is reinserted. Otherwise, the node is split.

The improvement of utilization is effective in lowering search cost. The Hilbert R-tree [KF94] achieves higher utilization. In the R-tree or the R*-tree, the overflow handling turns one node to two nodes, that is 1-to-2 split. The Hilbert R-tree has s-to-$(s + 1)$ split mechanism which the overflow handling performs split s nodes to $s+1$ nodes, or it moves the entries in overflowing nodes to sibling nodes. The overflow handling accommodates some entries among sibling nodes, it needs the information on positional relations between sibling nodes. The Hilbert R-tree uses the Hilbert curve [FR89] as the information (see Figure 3.9). Owing to the split policy with the Hilbert curve, the structure keeps almost 100 percent utilization.

On the X-tree [BKK96], the insertion algorithms for the circumvention of the overlap among nodes which constitutes the obstacle to pruning nodes in searching relative to the R-tree and the R*-tree are introduced. Above all, the notion of

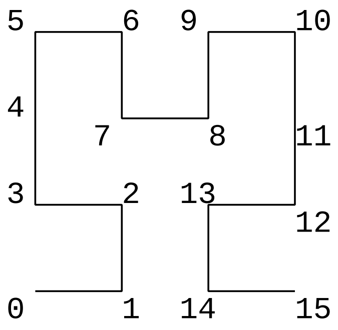


Figure 3.9. An example of Hilbert curve.

supernode which occupies one or more disk pages is progressive and important relative to the circumvention of bad node split. The mechanism of the X-tree leads to higher performance than the R*-tree for high-dimensional data.

In [BKK96], the problem that overlap between MBRs of nodes frequently occurs is indicated. The structure and algorithm for the X-tree are motivated by this observation. The structure of the X-tree is an impressive combination of a linear array-like organization and a hierarchical tree-like organization. In low-dimensional spaces, the algorithm prefers a hierarchical tree-like organization as shown in Figure 3.10 (a) due to the low overlap between MBRs of non-leaf nodes. However, the superiority of a linear array-like organization grows as dimensionality increases. In marked contrast to low dimension, a linear organization is more efficient for very high dimension as shown in Figure 3.10 (c). Moreover, for medium dimension, a combination of a linear array-like organization and a hierarchical tree-like organization is required (Figure 3.10 (b)). In [BKK96], the X-tree outperforms the R*-tree and the TV-tree[LJF94].

The SS-tree [WJ96b] is a tree structure using MBSs (Minimum Bounding Spheres) for high-dimensional data objects as shown in Figure 3.11. MBSs consist

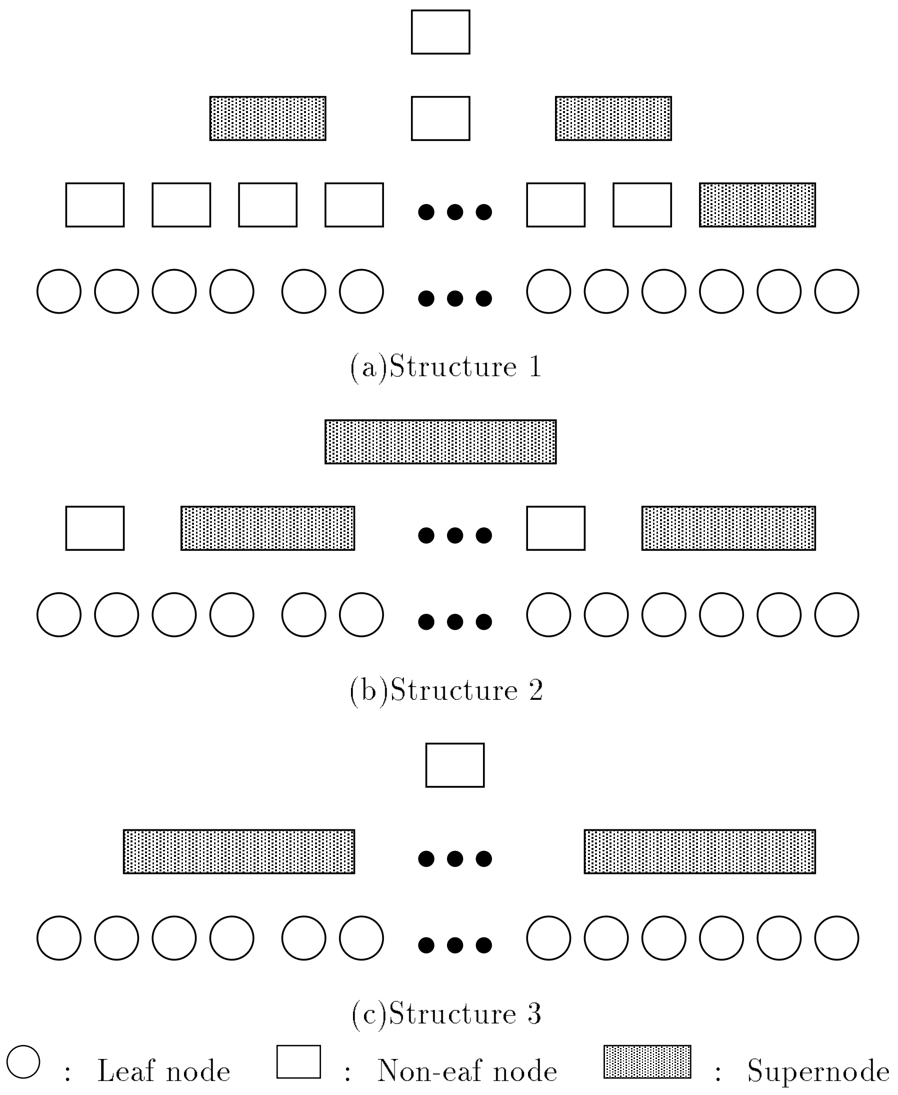


Figure 3.10. The X-tree.

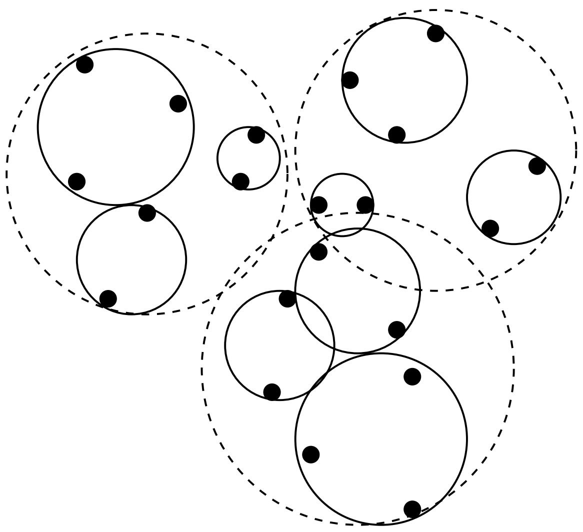


Figure 3.11. Data partitioning by MBS.

of the radius and the center point which is the average of all data objects enclosed by the sphere. In the SS-tree, an insertion of objects and a split of overflowing node are performed as the variance of each node is minimized. For insertion, the node of which its centroid is closest to the insert object, is chosen. If the number of entries in a node exceeds threshold, i.e. fanout, forced reinsert or node split is invoked.

In the node split of the SS-tree, first, the variance of the center point of MBS for each dimension is calculated, and then, the dimension with the highest variance is determined as the split dimension. Secondly, for the distribution of the entries to two new created nodes, the split location on the split dimension is chosen as the sum of the variances on each side of the split is minimized.

In addition, although the SS-tree uses the mechanism of forced reinsert, the opportunity of the mechanism for the SS-tree is not the same as one for the R*-tree. In the R*-tree, if the level of overflowing node is not the root level and this overflow handling is the first call on the level, the forced reinsert is invoked. By contrast, the forced reinsert of the SS-tree is performed if the overflowing node is not the root and this overflow handling is the first call on the node. As mentioned above, the forced reinsert of the SS-tree is performed with higher frequency than that of the R*-tree, and this superior mechanism is also adopted by the SR-tree.

In connection with the SS-tree, [KS97] indicates two strong points:

(1) The fanout is low since the representation cost of MBS is smaller than that of MBR.

(2) Closer objects can be clustered since the index structure attaches importance to the variance of entries in a node or the distance from the centroid of a node.

These strong points contributes to improving performance.

In the SR-tree [KS97], not only MBSs such as the SS-tree, but also MBRs are introduced into the tree structure (cf. Figure 3.12). The non-leaf nodes of the SR-tree contain entries of the form

$$\begin{aligned} Non-leafNode &= (\varepsilon, Record_i) \ \ (i = 1, \ldots, \varepsilon) \\ Record_i &= (C_S, C_R, \omega, child-pointer), \end{aligned}$$

where C_S and C_R are the MBS and the MBR of a child node, respectively. ω is the total number of points contained in the subtree whose top is the child pointed by $child-pointer$. Similar to the R-tree, ε is the number of entries, and $child-pointer$ refers to the child node.

The region of a entry for the SR-tree is represented by the intersection of MBS and MBR in the entry. Therefore, for the search algorithm in the SR-tree, the minimum distance from query point to bounding rectangle and the minimum distance from query point to bounding sphere are computed,then larger

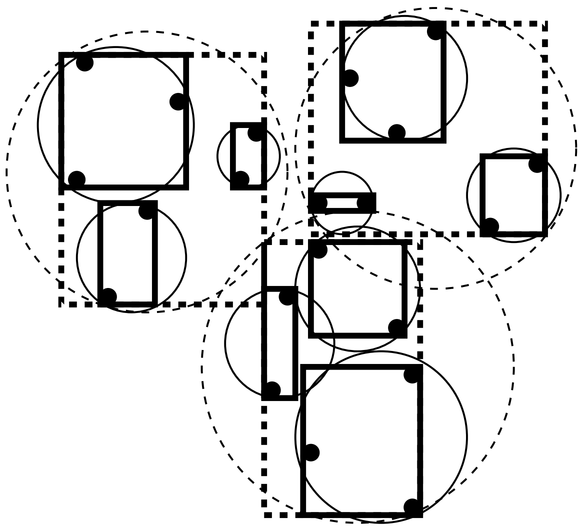


Figure 3.12. The SR-tree.

distance is chosen between these ones as distance of the region, and then, it is used for pruning nodes. As a result of the smaller region, it causes higher search performance. The SR-tree in R-tree family is one of the most excellent methods and it has high capability for node pruning, especially in high-dimensional space.

Nearest Neighbor Search Algorithms using R-tree family

As mentioned in Section 3.2, several kind of queries are useful in the field of spatial search. Relative to these queries, the nearest neighbor search algorithms for R-tree family have been proposed [RKV95] [HS95] [AMN$^+$94] [AMN$^+$98].

One is the branch-and-bound R-tree traversal algorithm [RKV95] proposed by Roussopoulos et al.. This algorithm based on ordering and pruning finds desired objects without missing out. In [RKV95], pruning MBR is based on the following strategies. In these strategies, Q represents a query point, R and R' represent MBRs, O and O' represent objects. In each strategy, if the corresponding condition is satisfied, R or O is pruned away.

$$\begin{aligned}
Strategy1-a &: MINDIST(Q,R) > MINMAXDIST(Q,R') \\
Strategy2-a &: \|Q,O\| > MINMAXDIST(Q,R') \;\; (O \notin R') \\
Strategy3-a &: MINDIST(Q,R) > \|Q,O'\|,
\end{aligned}$$

where $MINDIST(a,A)$ is the minimum distance from a point a to a rectangle A, and $\|a,b\|$ is Euclid distance between points a and b. $MINMAXDIST(a,A)$ is minimum distance which is guaranteed that at least one object enclosed by A exists in the hypersphere of which center point is a and radius is $MINMAXDIST(a,A)$. More concretely, observe that A in n-dimensional space is bounded by n pairs of hyperplanes, e_i and $E_i(i=1,\ldots,n)$, where each e_i is the hyperplane of which the distance from a is smaller than that of corresponding E_i. Let $b_i(i=1,2,\ldots,n)$ be vertexes which is farthest, from the point a, in each of the hyperplane e_i. Then,

$$MINMAXDIST(a,A) = \min_i \|a,b_i\|.$$

Figure 3.13 shows an example as $n=2$. In this example,

$$\begin{aligned}
MINDIST(a,A) &= \|a,b_0\|, \\
MINMAXDIST(a,A) &= \|a,b_2\|.
\end{aligned}$$

On [RKV95], the combination of three strategies leads to fewer disk accesses.

The description of the strategies presented above are mainly focused on nearest neighbor search. Since the goal of this dissertation is to find the k-th nearest neighbor objects as ranking query, $Strategy1-a$ and $Strategy2-a$ are extended as follows:

$$\begin{aligned}
Strategy1-b &: MINDIST(Q,R) > D \\
Strategy2-b &: \|Q,O\| > D \;\; (O \notin R')
\end{aligned}$$

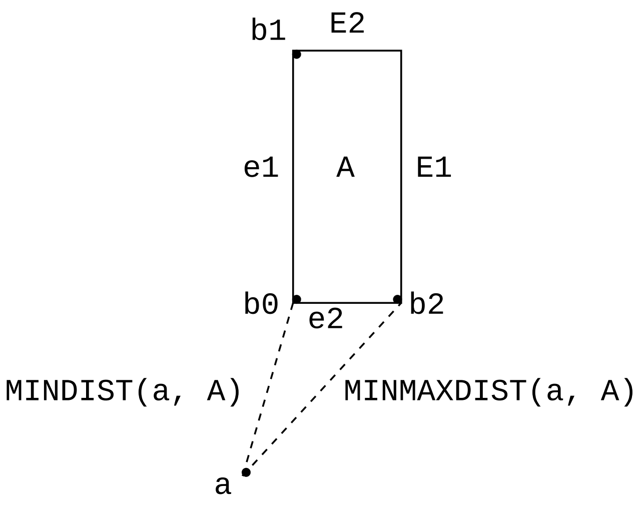


Figure 3.13. MINDIST and MINMAXDIST in two-dimensional space.

where

$$D = \begin{cases} NN.dist[k] & if\ NN.dist[k] \leq MINMAXDIST(Q, R') \\ & \quad \wedge NN.number = k \\ MINMAXDIST(Q, R') & otherwise. \end{cases}$$

$NN.number$ is the number of candidate objects collected during search processing, and $NN.dist[i]$ $(1 \leq i \leq NN.number)$ is the i-th nearest neighbor distance in the collections. If the number of collected objects is equal to k and $NN.dist[k]$ is not longer than $MINMAXDIST(Q, R')$, distance from the k-th object to Q is adopted for pruning strategy. Otherwise, the strategy of Roussopoulos et al. [RKV95] is adopted. $Strategy1 - b$ (and $Strategy2 - b$, respectively) which subsumes $Strategy1 - a$ (and $Strategy2 - b$, respectively) is used in the performance tests described below.

In [HS95], another method is proposed and applied to the PMR-quadtree [NS87]. The differences between the nearest neighbor algorithms are as follows:

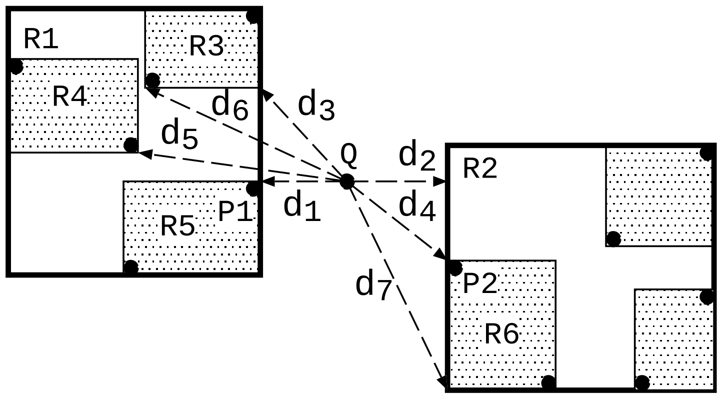


Figure 3.14. Two nearest neighbor search algorithms, [RKV95] and [HS95].

(1) The branch list is used as local variable in [RKV95], whereas in [HS95], the list is used as global variable. According as this change, the algorithm of [HS95] includes no recursive procedure.

(2) The algorithm of [HS95] uses only $Strategy3-a$. For $Strategy3-a$, the rectangle or the region R which the distance to query point Q is larger than the distance from the object O to Q, is pruned and removed. The algorithm in [HS95] does not use $Strategy1-b$ or $Strategy2-b$ since these strategies make no sense for pruning on this algorithm.

Here, the difference between these algorithms is mentioned using Figure 3.14. In this figure, the query point Q, two nearest neighbor point from Q, P_1 and P_2, several MBRs are presented. d_i is the distance value which is larger as i $(1 \leq i \leq 7)$ grows. On two-nearest neighbor query by [RKV95], the MBR R_1 of which $MINDIST$ is smaller than that of R_2, is accessed at first. And also, R_5 and R_3 are visited. Then, R_4 is accessed since the $MINDIST$ of R_4 (i.e. the distance is d_5) is smaller than the $MINMAXDIST$ of R_3 (i.e. the distance is d_6). Finally, R_2 and R_6 are accessed. Moreover, the other MBRs are pruned away.

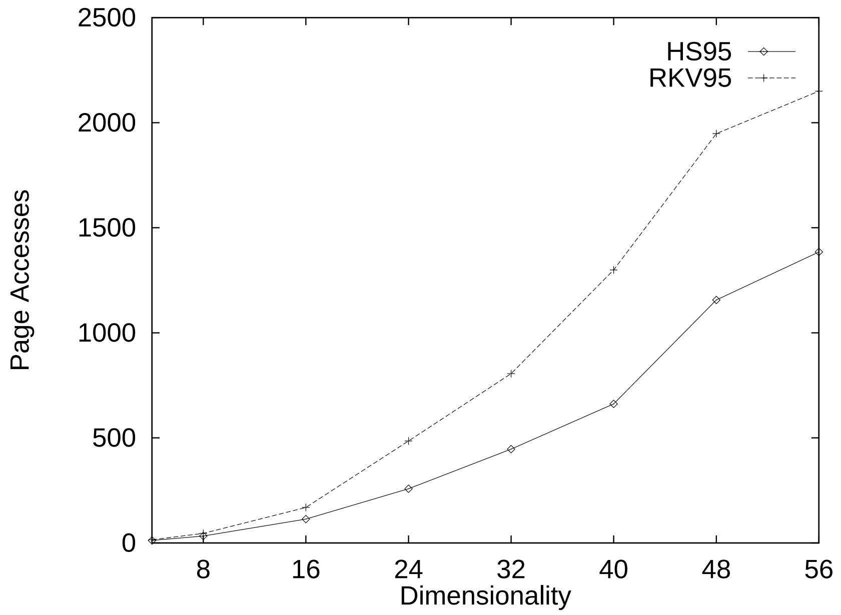


Figure 3.15. Number of page accesses for [RKV95] and [HS95].

However, the algorithm in [HS95] shows different behavior. In [HS95], R_1 and R_5 that have the smallest distance for $MINDIST$, are accessed at first. Then, R_2, R_3 and R_6 are accessed, however, R_4 is not visited since the $MINDIST$ of R_4 is larger than the distance between P_2 and Q. In this case, the algorithm of [HS95] achieves fewer page accesses than that of [RKV95].

Berchtold et. al. prove that the search algorithm of [HS95] is superior to that of [RKV95] theoretically for page accesses in [BBKK97], however, no experimental comparison between both algorithms has been reported so far. Thus, experiments for investigating the difference of the algorithms on page accesses and CPU-time as search performance, are performed in this research. Figure 3.15, Figure 3.16 and Figure 3.17 show the results for page accesses, CPU-time and total search time, respectively. This experiment is performed under the conditions described in Chapter 2. The size of data set is 100,000. The dimensionality varies from 4

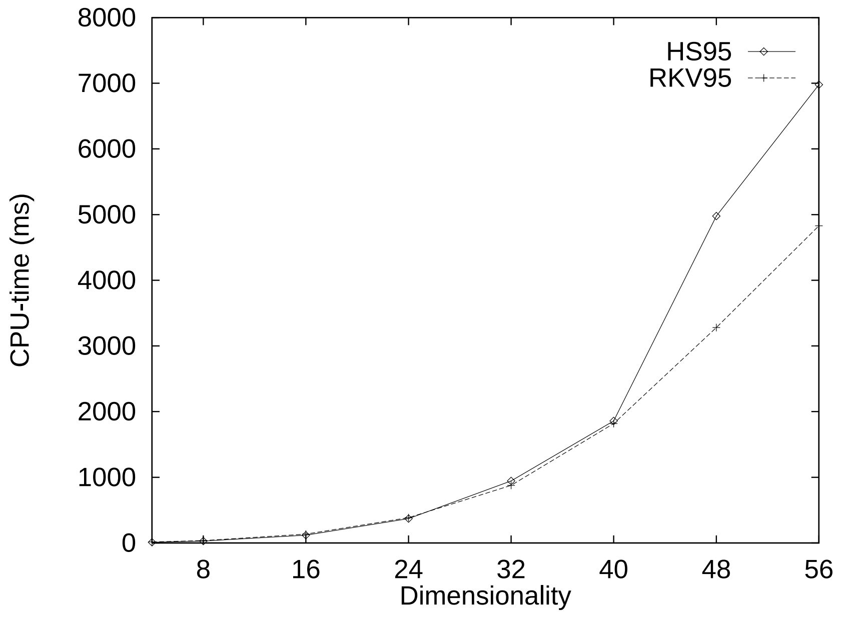


Figure 3.16. CPU-time on search processing for [RKV95] and [HS95].

to 56, and also this evaluation is based on the SR-tree with the real data sets.

Figure 3.15 shows that [HS95] is superior in all data sets ranging in dimensionality from 4 to 56, especially, in high-dimensional data sets. On the other hand, in connection with CPU-time, the superiority of [RKV95] is confirmed for 48 and 56 dimensions as shown in Figure 3.16. As total search time that is calculated by the page access time, 25 ms in [BKK96], [HS95] outperforms [RKV95] (cf. Figure 3.17).

The following is this reason. Since nearest neighbor queries occur many access paths to find required objects, the difference of the order of accessing nodes may cause the difference of search performance. Since it is possible to distribute k nearest neighbor objects into various leaf nodes or non-leaf nodes, the algorithm [RKV95] which is a depth-first search according as the metric $MINDIST$, causes not only necessary node accesses but also unnecessary accesses for sibling nodes.

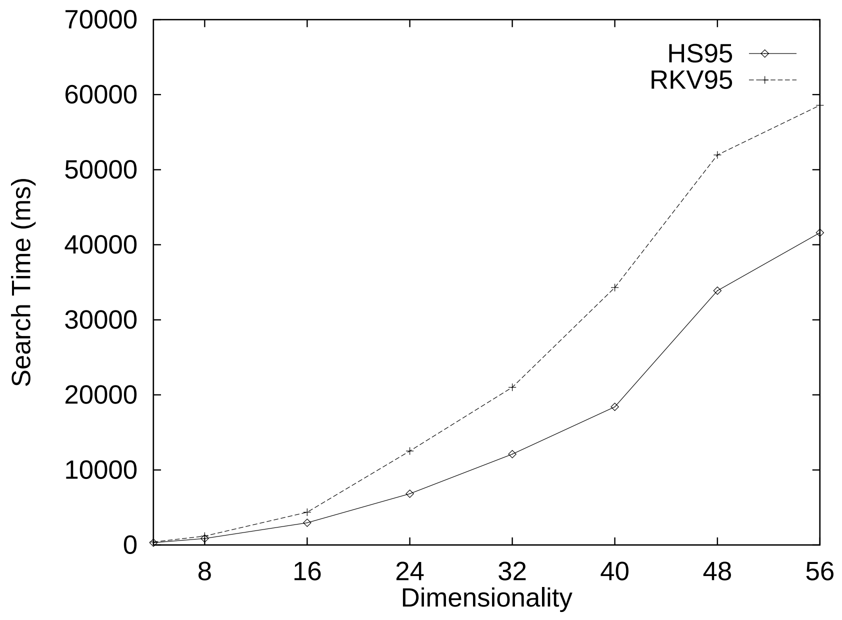


Figure 3.17. Total search time for [RKV95] and [HS95].

It is considerable factor that increase search cost.

In contrast with this depth-first search algorithm, the algorithm [HS95] that uses priority queue visits nodes in ascending order of $MINDIST$ from query point, irrespective of tree level for stratified structure. The way of node access which the node with minimum distance has priority over the other nodes in queue, finds objects with small distance faster than that of depth-first search. Thus, the pruning strategy of the algorithm [HS95] excludes unnecessary node accesses using smaller distance based on collected objects, it leads to lower search costs. Based on these descriptions, the algorithm of [HS95] is adopted into the evaluations of this dissertation.

3.4.3 Search Methods for High-Dimensional Data

Content-based retrieval uses many kind of features for excellent user interface. However, search cost grows dramatically as dimensionality increases. Unlike the methods which are workable only in low-dimensional space, the methods which are focused on this problem, have been proposed. The conventional approach to supporting similarity search in high-dimensional vector space can be broadly classified into two categories. As mentioned above, the first approach is using tree structure such as R-tree family. For R-tree family, neighbor vectors are covered by MBRs (Minimum Bounding Rectangles) or MBSs (Minimum Bounding Spheres), which are organized in a hierarchical tree structure. Two recently proposed indices, the X-tree [BKK96] and the SR-tree [KS97], are reported to offer good performance. The X-tree [BKK96] introduces the notion of supernode, and outperforms the R*-tree. The SR-tree [KS97] has a unique feature in that it uses both MBRs and MBSs, and reported to outperform both the R*-tree and the SS-tree. The second approach is the use of approximation files. Among others, the VA-File (Vector Approximation File) [WSB98] contained in the category of cell-based files, is a simple yet powerful scheme. In [WSB98], Weber et. al. have reported that the VA-File outperforms both the R*-tree and X-tree when the dimension becomes high ($\geq$ around 6.) To sum up, among access methods for high-dimensional vector space search, the SR-tree and the VA-File are two methods which are not reported to be outperformed by other methods.

Chapter 4

Analysis of Index Structures for High-Dimensional Search

In [WSB98], the limitations of R-tree family for high dimensional space is presented. The SR-tree as one of R-tree family is no exception to this assertion. In this chapter, performance evaluation of R-tree family, especially the SR-tree, from different point of view, and analysis of the properties and problems of them, are presented.

4.1 R-tree Family v.s. the VA-File

This section shows evaluation for the performance of indices, R-tree family and the VA-File, using synthesis and real data. The SR-tree which has a beneficial effect on high-dimensional search and the R*-tree as traditional methods are used on behalf of R-tree family. The experimental conditions are shown in Chapter 2. Fanout of nodes in the SR-tree and the R*-tree is shown in Table 4.1 and Table 4.2, and also height of these tree structures which are constructed using 100,000 real objects under the conditions is shown in Table 4.3. The data structure of leaf

Table 4.1. Fanout of tree structures for leaf node.

Dim.	4	8	16	24	32	40	48	56
Fanout	113	60	30	20	15	12	10	9

Table 4.2. Fanout of tree structures for non-leaf node.

Structure	Dimensionality							
	4	8	16	24	32	40	48	56
R*-tree	60	30	15	10	7	6	5	4
SR-tree	36	19	10	6	5	4	3	3

Table 4.3. Height of tree structures.

Structure	Dimensionality							
	4	8	16	24	32	40	48	56
R*-tree	3	4	5	6	7	7	8	9
SR-tree	4	4	5	7	8	9	15	15

nodes in the SR-tree and the R*-tree are the same, and the fanout is set as shown in Table 4.1. Also, the data structure of vector file in the VA-File set exactly same as that of leaf node in R-tree family. The search performances of R-tree family are measured based on the algorithm presented in [HS95] (cf. Section 3.4.2).

Figure 4.1, Figure 4.2 and Figure 4.3 show the number of page accesses of the VA-File, the SR-tree and the R*-tree, for uniformly distributed data, real data and cluster data, respectively. As shown in Figure 4.1, for uniformly-distributed synthetic data set, R-tree family is superior to the VA-File only in 4 and 8 di-

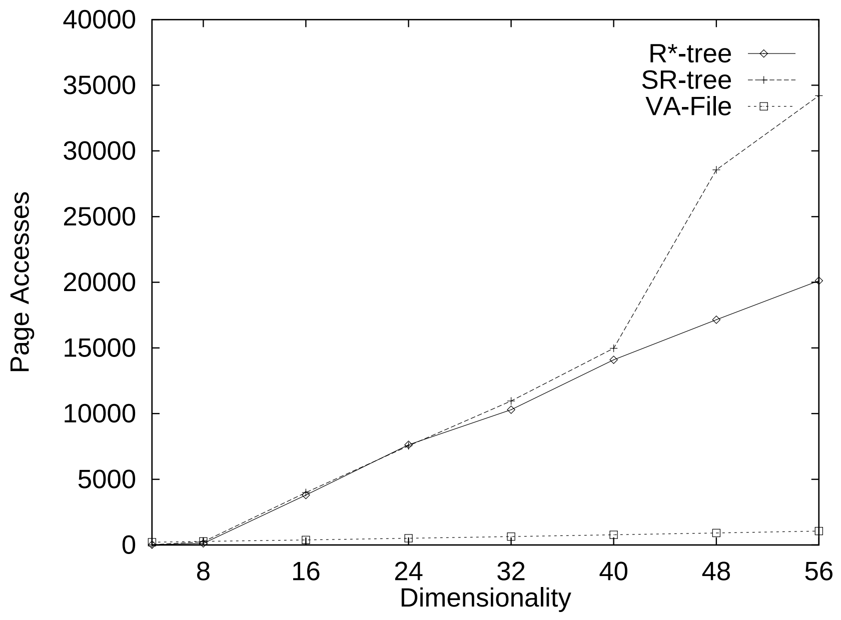


Figure 4.1. Number of page accesses for the VA-File, the SR-tree and the R*-tree (Uniform random data).

mensions. R-tree family deteriorates rapidly in higher than 8 dimension and this result is in accordance with the indication of [WSB98]. In R-tree family, the SR-tree has no superiority over the R*-tree for uniformly synthesis data sets.

On the other hand, for the results using real data in Figure 4.2 and cluster data in Figure 4.3, the SR-tree overwhelms the R*-tree, and also provides better performance than the VA-File up to 40 dimension. For realistic data set of which distribution is not uniform, the volume of MBSs and MBRs in the SR-tree becomes small flexibly in dense local space, and then this flexibility causes considerable effect in search. For the VA-File, filtering by approximation loses its effect for non-uniformly distributed data, and the number of visiting the vector files increases. Since most practical applications use non-uniformly distributed data set, the following discussion is focused on the SR-tree which shows desirable

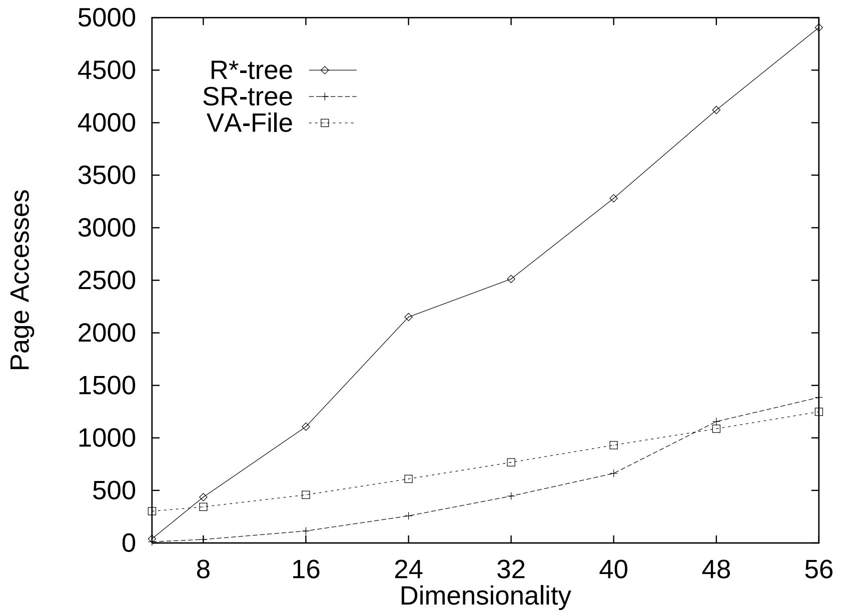


Figure 4.2. Number of page accesses for the VA-File, the SR-tree and the R*-tree (Real data).

performance.

Despite the fact that the SR-tree is good at non-uniformly distributed data, it degrades as dimensionality grows. In 48 dimension or higher, the SR-tree is inferior to the VA-File for real data sets. To probe into the cause, the next section shows a more detailed analysis.

4.2 Property of the SR-tree

Figure 4.4 shows the number of non-leaf and leaf accesses of the R*-tree and the SR-tree using the real data set under the same conditions as Section 4.1. The number of non-leaf accesses considerably increases as dimensionality becomes large for both the SR-tree and the R*-tree. In particular, the SR-tree exhibits

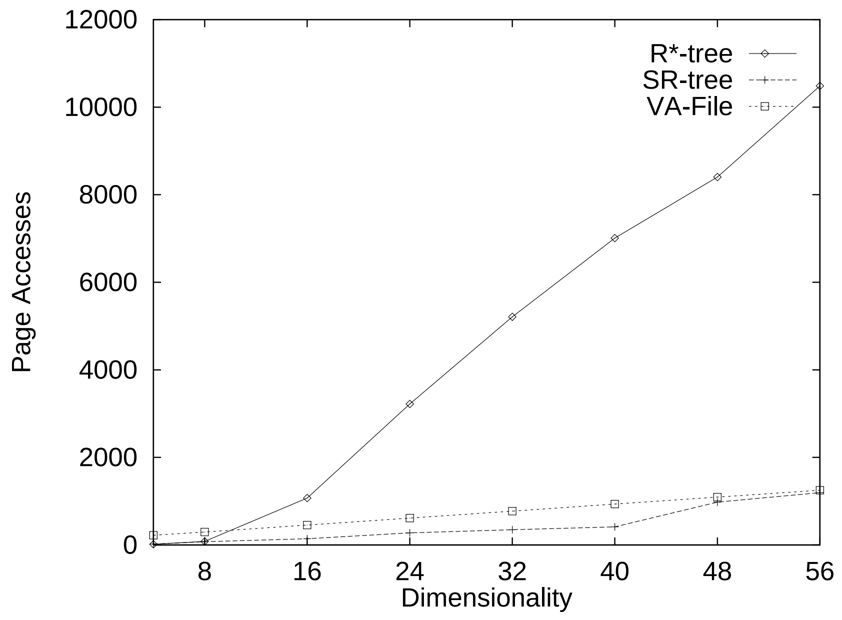


Figure 4.3. Number of page accesses for the VA-File, the SR-tree and the R*-tree (Cluster data).

marked tendency. Increase of non-leaf accesses leads to higher page accesses especially in over 40 dimension.

Non-leaf of both structures includes position coordinates of minimum bounding regions. The volume of coordinates are directly proportional to dimensionality, namely, fanout gets small with increase in dimensionality as shown in Table 4.2. Since the non-leaf nodes in the SR-tree include both MBRs and MBSs, the fanout of them are especially small. This causes increase of non-leaf accesses.

To point out another property, the relation between fanout and subtree-pruning capability is described. First, two SR-tree structures are constructed: the tree structure of which fanout accords Table 4.2, and one whose non-leaf nodes have ten times larger fanout than that shown in Table 4.2. In Figure 4.5, the number of leaf accesses versus the number of dimension is shown. In this

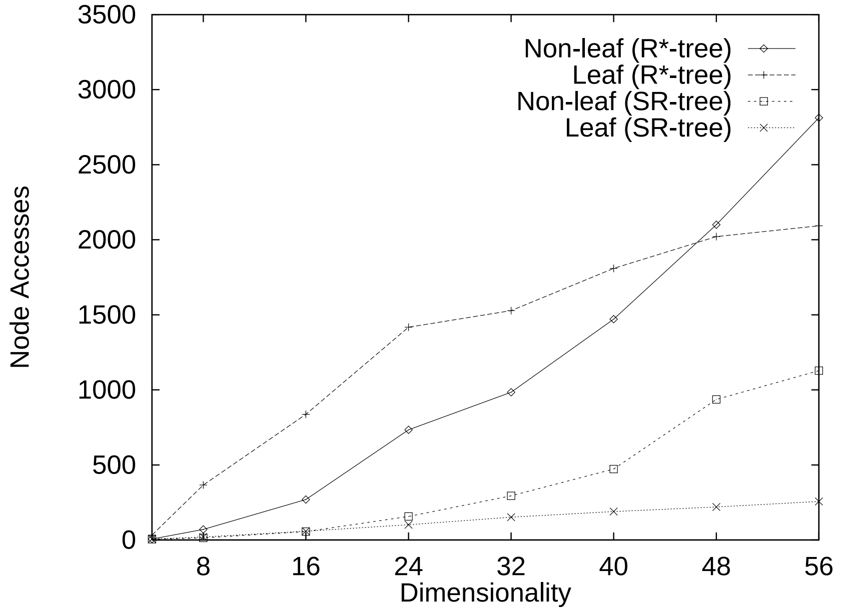


Figure 4.4. Number of node accesses for the SR-tree and the R*-tree.

figure, SR-tree(1) indicates the normal structure of the SR-tree, and SR-tree(10) is the SR-tree with non-leaf nodes of ten times larger fanout. SR-tree(10) exhibits smaller number leaf accesses in this figure. The reason is explained as follows. In the SR-tree, an object is inserted so as to minimize the variance of objects in each node. However, owing to update of the tree structure after this insertion, previously inserted objects are forced to be located undesirable position[1]. Increase of fanout leads to higher possibility that objects are located desirable position, and then it causes to reduce number of leaf accesses. Therefore, the SR-tree has the property that this method is liable to place objects on unsuitable position because of its small fanout.

[1]Not all objects are located desirable position in tree structure by the forced reinsert.

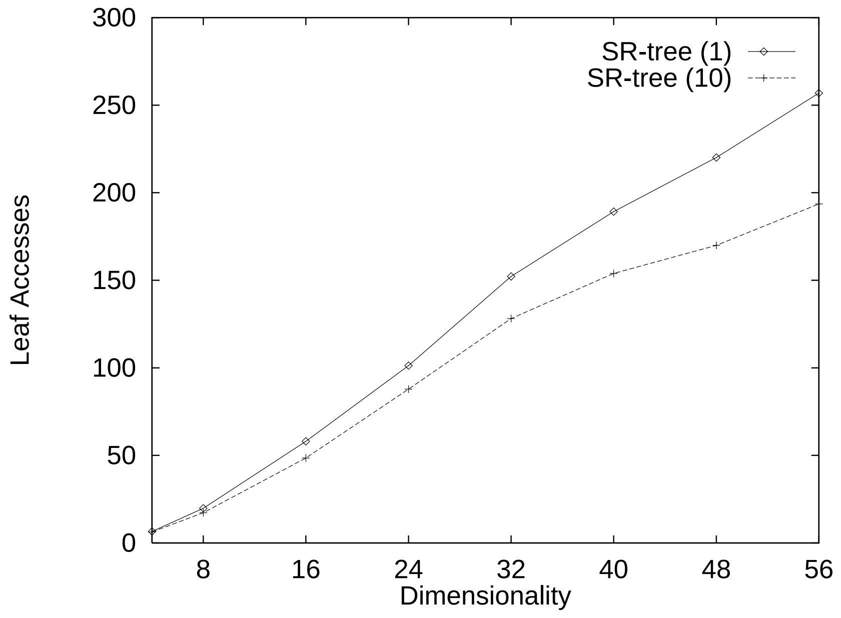


Figure 4.5. Relation between fanout and leaf accesses.

4.3 Conclusions

The discussions in this chapter are summarized as follows:

(1) The SR-tree in R-tree family is superior in search performance. For non-uniformly distributed real data, the SR-tree overwhelms the R*-tree and offers better performance than the VA-File up to 40 dimension.

(2) The SR-tree has high pruning capability using its non-leaf nodes. However, large volume of non-leaf nodes causes to increase page accesses and degrades the search performance. In addition, this tendency of performance degradation becomes worse as dimensionality grows.

(3) Smaller fanout leads to lower pruning capability and increase of leaf accesses. As dimensionality grows, the influence of this problem increases

also.

Although the R*-tree is in a similar for the second and the third indication, the SR-tree is more sensitive to the influence.

Based on the analysis of the experimental result, the Subspace Coding Method (SCM), a new indexing scheme which is applicable to many tree indices, is introduced and presented in Chapter 5 for reducing number of non-leaf accesses and improvement of pruning capability.

Chapter 5

The Subspace Coding Method

5.1 Introduction

The experimental results in Chapter 4 suggest that the R-tree family, especially the SR-tree, has the possibility to outperform the VA-File in high dimensionality by increasing the fanout of non-leaf nodes. This observation is the motivation for the introduction of the Subspace Coding Method (SCM). The basic idea of the SCM is the introduction of *VBR (Virtual Bounding Rectangle)* and *VBS (Virtual Bounding quasiSphere)*, which covers and approximates MBR and MBS, respectively. Since the required storage size for VBRs are[1] much smaller than that for MBRs, storing VBRs in non-leaf nodes increases the fanout by an order of magnitude. This idea is analogous to the prefix B-tree [BU77]; both the prefix B-tree and the SCM reduce the size of keys in non-leaf nodes thus making the fanout larger and the size of entire tree smaller. However, unlike the prefix B-tree, VBRs have negative effects as well. Since VBRs cover and hence are

[1]The following discussion is focused on VBRs and MBRs for simplicity. Analogous discussion applies to VBSs and MBSs as well. The formal definition of VBRs and VBSs is given in Chapter 5.2.

slightly larger than MBRs, the intersection of children's' VBRs are larger than the intersection of corresponding MBRs. This factor possibly causes the degradation of the power of pruning subtrees in searching. However, as presented in Chapter 6, the performance evaluation shows the positive effect of VBRs overwhelms this negative factor in practical situation.

One important feature of the SCM is the relative approximation of rectangles. Since the overlap of MBRs are allowed in tree indices of the R-tree family, the approximation error in non-leaf nodes near leaves significantly affects the search performance. Therefore, the coding scheme used for representing VBRs should be compact, and approximation error should be independent to the size of MBRs. The subspace code introduced in this chapter meets these requirements. Unlike the approximation of absolute vector positions used in the VA-File, the subspace code represents the relative position of VBRs/VBSs in terms of parent's VBR/VBS. Since the approximation of absolute vector positions used in the VA-File are independent of the data distribution, many dense data tend to be approximated by same value, hence is not effective to non-uniformly distributed data which are commonly found in real applications. By contrast, the relative representation in the SCM is effective for non-uniformly distributed vectors.

In connection with the above discussion, Figure 5.1 shows the classification of spatial access methods from two viewpoints: representation of spatial objects and index structure. Conventional spatial access methods can be roughly classified into the following three categories: (1) linear scan; (2) the VA-File, a sequential file of absolute approximation of feature vectors; and (3) R-tree family which has tree structures. R-tree family can be further classified into "pure" tree-structured indices such as the R*-tree and the SR-tree, and "hybrid" of tree and sequential structure such as the X-tree, which, with the notion of supernode, shows higher property of sequential scan as dimensionality increases. The SCM does not belong

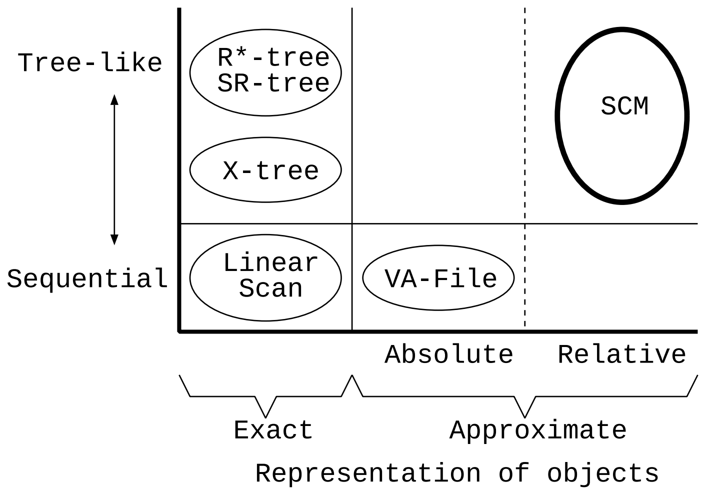


Figure 5.1. Classification of spatial access methods.

to any of these categories and is unique in that i) the SCM, by itself, is not an index but an index scheme applicable to any tree-structured index; and ii) the representation of bounding regions is based on approximation relative to their parent nodes. The SCM gives higher search performance from these properties.

In this chapter, the notion of VBR and VBS is introduced, and then the SCM, the index structure using it, and the algorithms for searching, insertion and deletion are described.

5.2 Preliminaries

A rectangle A in n-dimensional space is represented by the two endpoints $\boldsymbol{a}$ and $\boldsymbol{a}'$ of its major diagonal: $A = (\boldsymbol{a},\ \boldsymbol{a}')$, where $\boldsymbol{a} = [\phi_1, \phi_2, \ldots, \phi_n]$, $\boldsymbol{a}' = [\phi'_1, \phi'_2, \ldots, \phi'_n]$ and $\phi_i \leq \phi'_i$ for $i \in \{1, 2, \ldots, n\}$.

Let B be a geometrical object in n-dimensional space. Here, let assume two cases: one is the case B is a rectangle, the other is the case B is a point object. $B \subseteq A$ is used to represent that B is contained in A. More formally, let $A = (\boldsymbol{a}, \boldsymbol{a}')$ and coordinate values of $\boldsymbol{a}$ and $\boldsymbol{a}'$ be those given above. Also, if B is a rectangle, let $B = (\boldsymbol{b}, \boldsymbol{b}')$ and coordinate values of $\boldsymbol{b}$ and $\boldsymbol{b}'$ be: $\boldsymbol{b} = [\psi_1, \psi_2, \ldots, \psi_n]$, $\boldsymbol{b}' = [\psi'_1, \psi'_2, \ldots, \psi'_n]$. Then, $B \subseteq A$ holds if and only if

$$\phi_i \leq \psi_i \leq \psi'_i \leq \phi'_i \quad (i = 1, 2, \ldots, n). \tag{5.1}$$

In addition, the case B is a point object satisfies $\boldsymbol{b} = \boldsymbol{b}'$. Similarly, relationship $B \subseteq A$ holds in this case.

5.3 Virtual Bounding Region and Subspace Code

In R-tree family, the coordinate values of endpoints of MBRs or the coordinate values of center points of MBSs are represented by absolute values in coordinate system. Although absolute values represent exact positions, the required storage size of absolute values grows in proportional to the number of dimension.

Bounding regions of child nodes are contained in a bounding region of the parent node in R-tree family. Hence, a bounding region B of a child node can be represented by the relative position in terms of a rectangle A in the parent node. The basic idea of SCM is based on this observation. Moreover, by using approximate relative positions, the length of bits required to represent positions is substantially reduced.

Definition 1 *Let A and B be rectangles in n-dimensional space such that $B \subseteq A$. Also, let q (≥ 1) be an integer. The* **virtual bounding rectangle** (**VBR** *for short) of B (in A with radix q) is the rectangle*

$$V_b = (\boldsymbol{v}, \boldsymbol{v}')$$

where

$$\boldsymbol{v} = [\tau_1, \tau_2, \ldots, \tau_n], \ \ \boldsymbol{v}' = [\tau'_1, \tau'_2, \ldots, \tau'_n], \ \ \tau_i \leq \tau'_i$$

such that

$$\tau_i = \phi_i + \left\lfloor \frac{\psi_i - \phi_i}{(\phi'_i - \phi_i)/q} \right\rfloor \cdot (\phi'_i - \phi_i)/q \tag{5.2}$$

$$\tau'_i = \phi_i + \left\lceil \frac{\psi'_i - \phi_i}{(\phi'_i - \phi_i)/q} \right\rceil \cdot (\phi'_i - \phi_i)/q \tag{5.3}$$

$$(i = 1, 2, \ldots, n).$$

In particular, if B is a point object, the virtual bounding region V_b is defined under the condition $\psi_i = \psi'_i$. □

Example 1 *In Figure 5.2(a), rectangles A and B that satisfy $B \subseteq A$ are given. The rectangle V_b shown by the broken line is the VBR of B in A with radix 8.* □

Lemma 1 *For a VBR V of B in A, the containment relationships $B \subseteq V$ and $V \subseteq A$ hold.*

Proof: Since $B \subseteq A$, the inequality (5.1) holds. Hence,

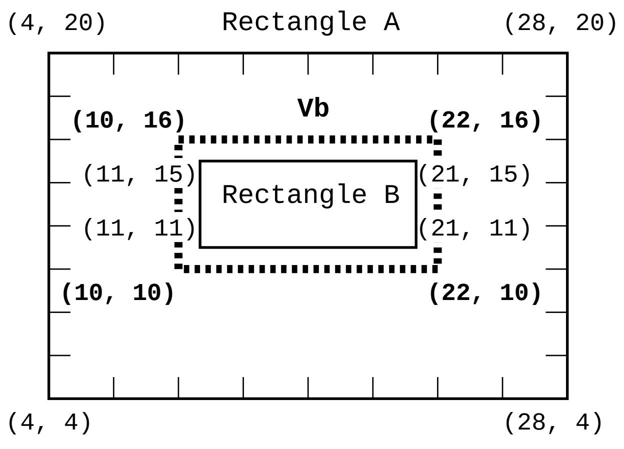
$$\left\lfloor \frac{\psi_i - \phi_i}{(\phi'_i - \phi_i)/q} \right\rfloor \cdot (\phi'_i - \phi_i)/q \ \geq \ 0$$

holds in (5.2), which implies $\tau_i \geq \phi_i$.

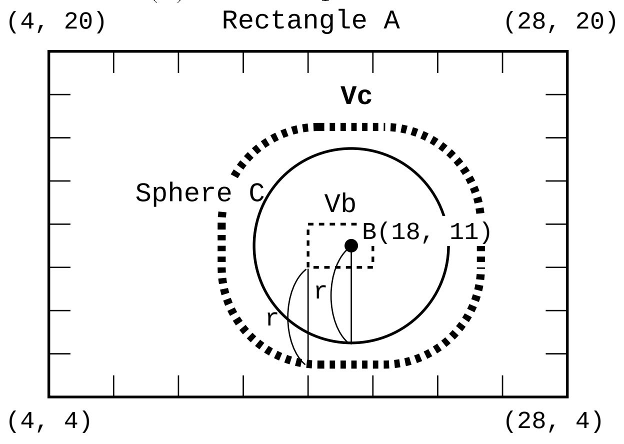
From (5.2),

$$\begin{aligned}\tau_i &= \phi_i + \left\lfloor \frac{\psi_i - \phi_i}{(\phi'_i - \phi_i)/q} \right\rfloor \cdot (\phi'_i - \phi_i)/q \\ &\leq \phi_i + \frac{\psi_i - \phi_i}{(\phi'_i - \phi_i)/q} \cdot (\phi'_i - \phi_i)/q \\ &= \psi_i.\end{aligned}$$

Hence, $\phi_i \leq \tau_i \leq \psi_i$. Similarly $\psi'_i \leq \tau'_i \leq \phi'_i$ holds. Therefore, $B \subseteq V$ and $V \subseteq A$. □



(a) An example of VBR



(b) An example of VBS

Figure 5.2. Examples of spatial representation using virtual bounding regions.

Definition 2 *Let A be a rectangle and C be the sphere of which radius is r and center point is B such that $B \subseteq A$. And then, let V_b be the VBR of B in A. Here, the* **virtual bounding quasisphere** (**VBS** *for short*) *of C(in A with radix q) is the geometrical object which represented by the Minkowski sum [BBKK97] of C and V_b.* □

The term *virtual bounding region* will be used to generally represent both VBR and VBS.

Example 2 *In Figure 5.2(b), a rectangle A and a sphere C are given. Also, r is the radius of C and B is the center point of C that satisfy $B \subseteq A$. The rectangle V_b shown by the broken line is the VBR of B in A with radix 8 and V_c is the VBS of C in A.* $\square$

Theorem 1 *The nearest-neighbor search algorithms of [HS95] and [RKV95], applied to virtual bounding regions, obtain the same results as the case when they are applied to minimum bounding regions.*

Proof: Let B_r be an MBR, and C be an MBS of which center point is B_p. Also, let V_r, V_c and V_p be a VBR of B_r, a VBS of C and a VBR of B_p, respectively. From Lemma 1, $B_r \subseteq V_r$ holds. Similarly, since $B_p \subseteq V_p$, $C \subseteq V_c$ holds. Hence, the following relationship holds between minimum bounding region B and virtual bounding region V of B at any query point Q in terms of the metric $MINDIST$. (Given a point P and a geometrical region R, $MINDIST(P, R)$ is the minimum distance between P and R if P is not contained in R, and is 0 otherwise.)

$$MINDIST(Q, V) \leq MINDIST(Q, B)$$

$\square$

Theorem 1 guarantees that search algorithms based on virtual bounding regions successfully find desired objects without missing out.

Next, the definition of subspace code is described. First, a simple example is given, then a formal definition follows.

Example 3 *Figure 5.3 gives a simple example to demonstrate how subspace code is calculated. The line A occupies from 3 to 19 on a coordinate, and the line B*

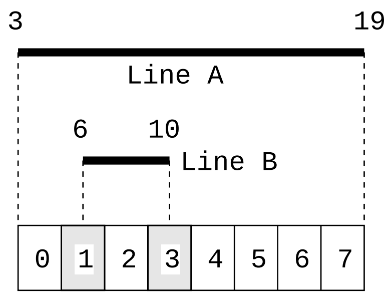


Figure 5.3. An example of subspace code.

occupies from 6 to 10 on the same coordinate. In general, a binary code of length l *can represent one of* 2^l *subintervals of the interval of A. Subspace code is used to represent the subintervals to which start and end points of the interval of B belong. In this way, the subspace code approximately represent B in terms of A. Figure 5.3 shows the case when* $l = 3$*, namely the interval of A is divided into* 8 *sub-intervals. Since B occupies the second and fourth sub-intervals, the interval of B is approximated as a pair of 8-ary code (1, 3) or binary code (001, 011). If B is a point object, the code of half length is calculated similarly.* □

Definition 3 *Let* $A = (\boldsymbol{a}, \boldsymbol{a}')$ *and* $B = (\boldsymbol{b}, \boldsymbol{b}')$ *be two rectangles in n-dimensional space and* $V = (\boldsymbol{v}, \boldsymbol{v}')$ *be the VBR of B in A with radix q such that* $B \subseteq V \subseteq A$*, hence the following relationship holds.*

$$\phi_i \le \tau_i \le \psi_i \le \psi_i' \le \tau_i' \le \phi_i' \qquad (i = 1, 2, \ldots, n).$$

Let η_i *and* η_i' *be q-ary code with* $\eta_i \le \eta_i'$ *that represents a sub-intervals on the i-th dimensional coordinate, such that*

$$\begin{aligned} \eta_i &= \frac{(\tau_i - \phi_i) \cdot q}{\phi_i' - \phi_i} \\ \eta_i' &= \frac{(\tau_i' - \phi_i) \cdot q}{\phi_i' - \phi_i} - 1. \end{aligned}$$

Also, let $F_2(\eta_i, l)$ be the binary representation of η_i by code of length l where $l = \lceil \log_2 q \rceil$. The **subspace code of V in** *A is defined to be the binary code*

$$S = (\boldsymbol{s}, \boldsymbol{s}')$$

where

$$\boldsymbol{s} = [F_2(\eta_1, l), F_2(\eta_2, l), \ldots, F_2(\eta_n, l)],$$
$$\boldsymbol{s}' = [F_2(\eta'_1, l), F_2(\eta'_2, l), \ldots, F_2(\eta'_n, l)].$$

And also, if B is a point object (i.e. $B = (\boldsymbol{b})$), the subspace code is defined to be $S = (\boldsymbol{s})$ □

In spite of fewer amount of information, subspace code has the capability to reconstruct bounding rectangle.

In contrast to original rectangle based on absolute position, virtual bounding region based on relative location with little error is restored by subspace code.

Example 4 *Figure 5.2 is an example of subspace code in two-dimensional space. In Figure 5.2(a), rectangles A, B and the VBR V_b of B in A that satisfy $B \subseteq V_b \subseteq A$ are given. The subspace code of V_b in A with radix 8 is*

$$S = (010, 011, 101, 101).$$

And also, B is the center point of the sphere C in Figure 5.2(b). Here, the subspace code of V_b is

$$S = (100, 011).$$

The VBS V_c of C is represented by S and the radius r of C. □

5.4 Tree Structure by Subspace Code

The differences between structure by the SCM and usual tree structure (i.e. R-tree family) are the following.

(1) Not only the tree structure organized by subspace code but also traditional tree is generated. Two structures that one called virtual part is organized by subspace code and the other called real part is traditional structure in order to distinguish these structures, are defined.

(2) Each non-leaf node in virtual part is created corresponding to non-leaf node in real part of tree structure, however no leaf node is created in virtual part. If the height of real part structure is s and the level of root node is s, the node on second level of virtual part indicates the node on first level of real part (i.e. leaf node) as child pointer.

(3) In structures of R-tree family except for X-tree [BKK96] and Hilbert R-tree [KF94], one node mainly corresponds to one page. For the SCM, since structure of virtual part is designed that one node corresponds to one page, one node occupies more than one page in real part. This structure leads to larger fanout.

(4) Index keeps size and position of root node that can show MBR.

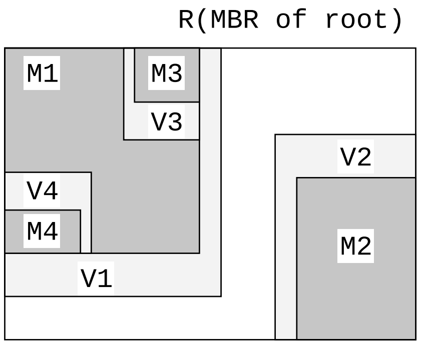
(5) Subspace code representation of virtual bounding region of child node is calculated by virtual bounding region of current node in virtual part and minimum bounding region of child node in the other part. In addition, subspace code which represents virtual bounding region of grandchild is calculated by virtual bounding region of child node and minimum bounding region of grandchild node, recursively. In more detail, on the tree structures

of which node contains MBRs (e.g. the R*-tree, the X-tree), subspace code which approximates MBR is calculated by VBR of current node and MBR of child node. On the tree structure of which node contains MBRs and MBSs (i.e. the SR-tree), the way of making subspace code which approximates MBR is similar to that of the R*-tree or the X-tree, and also subspace code which approximates center point of MBS is calculated by VBR of current node and center point of MBS of child node. For the tree structure of which node does not include MBR (e.g. the SS-tree), minimum bounding rectangle which contains VBS of current node is calculated at first, then subspace code is computed by the calculated rectangle of current node and center point of MBS of child node.

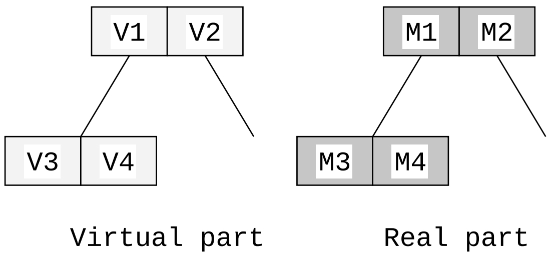
(6) Exceptionally, in the case that current node is root, subspace code which represents virtual bounding region of child node is calculated by MBR of root and minimum bounding region of child node.

(7) In search processing, virtual bounding region of child node is calculated by virtual bounding region of current node and subspace code of child node at every node of virtual part, and then this virtual bounding region is used for pruning.

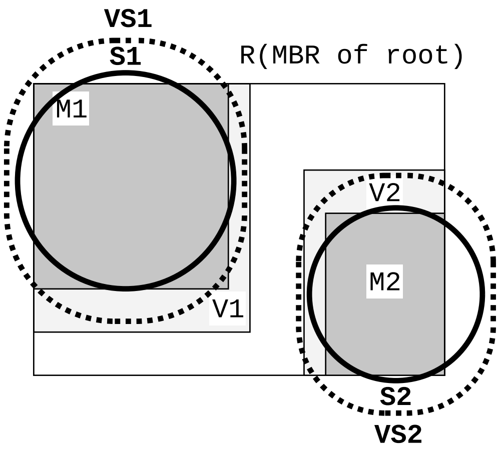
For the SCM, a node in virtual part has the same structure as that of real part except for that position coordinate is coded. Figure 5.4 explains the relationship between real part and virtual part. In Figure 5.4(a), R means MBR of root node. Also, $M1$ and $M2$ mean MBRs in R, and MBRs $M3$ and $M4$ are contained in $M1$. In this structure, $V1$, $V2$, $V3$ and $V4$ are the VBR of $M1$ in R, the VBR of $M2$ in R, the VBR of $M3$ in $V1$ and the VBR of $M4$ in $V1$, respectively. Structure of virtual part are similar to that of real part as shown in Figure 5.4(b). Figure 5.4(c) is an example of the structure of the SCM applied to the SR-tree.



(a) VBR and MBR



(b) Tree structures



(c) VBS and MBS

Figure 5.4. Real and virtual part in tree structure.

In this figure, $VS1$ is the VBS of $S1$ in R, and $V1$ is the VBR of $M1$ in R. In the tree index node of R, that is root node, $VS1$ and $V1$ are packed in the same entry.

5.5 Searching

For search processing, nodes in virtual part and leaf nodes in real part are used in order to find objects and prune nodes. Virtual bounding regions are applicable to the algorithm presented in [HS95] or [RKV95] as shown in Theorem 1.

Figure 5.5 shows nearest neighbor search algorithm using VBRs and VBSs which is improved on the algorithm in [HS95]. In the program codes, the definition of variables is omitted for brevity. In steps 11 and 12, *reconstruct* is the function which calculates the virtual bounding region R for a given rectangle A and a subspace code S, i.e.

$$R = reconstruct(A, S).$$

This algorithm has the capability to collect k nearest neighbor objects for a given query point.

In the procedure *search*, root node, zero distance and the MBR of root node are inserted into priority queue as initial values (step 1). In step 3, the node which is closer to a query point, *query*, is picked out from the queue. At every visited non-leaf node, virtual bounding regions of all child nodes are computed with bounding rectangles of current node taken from the priority queue and subspace code of child node by the function *reconstruct* (steps 11 and 12). As the distance from *query* to calculated virtual bounding region, the longer distance between the minimum distance to VBR and the minimum distance to VBS is selected (step 13). The child pointer, the distance from *query* to virtual bounding region and calculated virtual bounding rectangle are inserted into the queue if this distance is less than or equal to the distance from *query* to the k-th nearest neighbor (steps 14 and 15). If visited node is a leaf node, the distance between *query* and data object is computed, then object and the distance are stored into *nnlist* as candidate for nearest neighbor (step 8).

```
Procedure search(Point query, Integer k)
    // input data(node, distance, region) into priority queue
1.    queueInput(root, 0, rootrectangle);
2.    while emptyQueue() = false do
3.      p = queueOutput();
4.      if p.distance > nnlist[k].distance then break;
5.      if p.node is a leaf node then
6.        for each object $\in$ p.node do
7.          if MINDIST(query, object) $\leq$ nnlist[k].distance then
8.            nnlistInput(object, MINDIST(query, object));
9.      else   // p.node is a non-leaf node
10.       for each entry $\in$ p.node do
11.         $R_{VBR}$ := reconstruct(p.rectangle, entry.$code_{VBR}$);
12.         $R_{VBS}$ := reconstruct(p.rectangle, entry.$code_{VBS}$);
13.         dist = max(MINDIST(query, $R_{VBR}$), MINDIST(query, $R_{VBS}$));
14.         if dist $\leq$ nnlist[k].distance then
15.           queueInput(entry.node, dist, $R_{VBR}$);
16.       end;
17.   end;
18.   report(nnlist);   // output answer
end.
```

Figure 5.5. Nearest neighbor search algorithm.

Here, search algorithm is explained using Figure 5.4(a) as an example. First, VBRs $V1$ and $V2$ are calculated by the position coordinates of R and the subspace codes of $V1$ and $V2$. If the distance from a query point to $V1$ is less or equal to the distance to the k-th nearest neighbor, VBRs $V3$ and $V4$ are calculated by the position coordinates of $V1$ and the subspace codes of $V3$ and $V4$. Similarly, the calculated VBRs are compared with the k-th nearest neighbor. In addition, search using bounding sphere is performed with the same algorithm.

5.6 Insertion

In the SCM, it is necessary to update both of the real and virtual parts in a structure. Insertion for real part is similar to one for structure of R-tree family. Since overflow handling for real part and one for virtual part are the same, nodes in virtual part are updated as keeping structure which is similar to real part. Process for updating subspace code in virtual part will start after object insertion at real part and overflow handling of real part and virtual part is finished. The following description accounts for update algorithm of subspace code. In this description, update algorithm for the tree structure is explained using the SR-tree which contains MBRs and MBSs.

Subspace codes in nodes of virtual part corresponding to nodes whose MBR or MBS is updated in real part must be checked. Especially, for MBR, not only the code corresponding to updated MBR but also the code in all descendants of the updated node must be checked, and if necessary, be updated. Update of MBR propagates upwards. In contrast, update of subspace code propagating downwards is not necessarily over even the process reaches a node of which subspace code needs no renewal. Therefore, process further propagates to lower nodes. The reason is as follows: since a VBR whose subspace code based on relative position

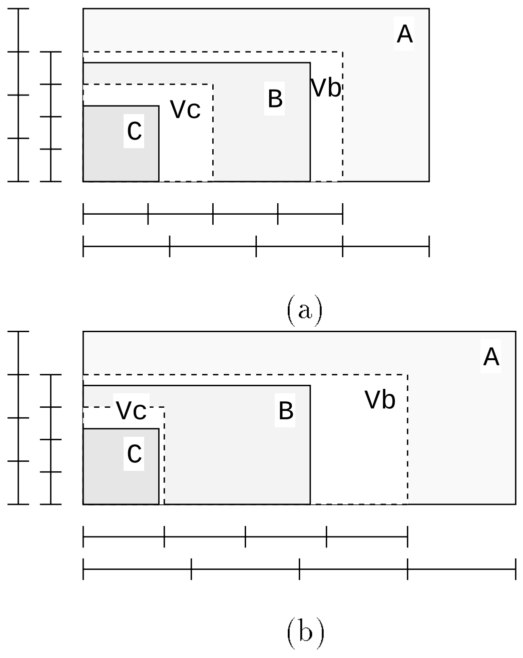


Figure 5.6. Process for making VBR

contains error, regardless of necessity to update current node, lower nodes have possibility that the subspace code of them needs correction. Here, the reason is illustrated using Figure 5.6. In Figure 5.6(a)(b), let rectangles A, B and C be root node, a child node and a descendant, respectively. Also, let V_b be the VBR of B in A with radix four and V_c be the VBR of C in V_b. As compared with Figure 5.6(a), only rectangle A is enlarged for the x-axis in Figure 5.6(b). Although the size of rectangle B is unchanged, enlargement of rectangle A leads to enlargement of V_b. Moreover, although the size of the rectangle C is unchanged, enlargement of rectangle V_b leads to decrease in size of V_c. In this case, the subspace code of V_c needs to be adjusted, despite no necessity of updating the subspace code of V_b.

Figure 5.7 shows the algorithm for updating subspace code in detail. In this description, *calculateRegion* is the function, which calculates virtual bounding region of child node using rectangle of current node and minimum bounding region of child node (steps 5, 11, 13, 18 and 20). In addition, subspace code of child node is computed by the function *calculateCode* (steps 6, 12, 14, 19 and 21). In order to update subspace code, the procedure *update* makes use of the insert path, *path*. In this procedure, *routeOutput* is the function which outputs the route from root to the top node of subtree for each level on insert path where the top node has more than one entry of which MBR is updated.

The linkages between nodes in virtual part are similar to that of real part, and also the access of node in real part is performed together with that of the corresponding node in virtual part. Therefore, it is easy to find the entry in node of virtual part which corresponds to the entry in that of real part, and there is no need for special or complicated operation. There is one-to-one correspondence between $entry_{real}$ and $entry_{virtual}$ in Figure 5.7, the function *correspond* is described in order to show the correspondence.

In *update*, first, subspace codes of VBSs in the entries on the route from root to the top of subtree are updated and the VBR of the top node is calculated (from step 1 to step 8). Secondly, the codes for VBR and VBS in the top node of subtree is updated using the VBR of the top node (from step 9 to step 14). Thirdly, the recursive procedure *adjustCode* with VBR as parameter calculated by *calculateRegion* is invoked for updating code sequentially toward lower nodes. The procedure *adjustCode* using *calculateRegion* and *calculateCode* calculates virtual bounding regions and subspace code by the rectangle of current node generated in upper node and minimum bounding region of child node, and then subspace code is updated. Moreover, *adjustCode* is invoked again with VBR as parameter and code of descendants is updated.

Owing to the reference of MBRs for checking subspace code, insertion causes accesses to both virtual and real parts. Concerning insertion costs, the SCM

```
Procedure update(InsertRoute path)
1.    $R_p$ := rootrectangle;
2.    while routeReachSubtree(path) = false do
3.       $entry_{real}$ := routeOutput(path, realpart);
4.       $entry_{virtual}$ := correspond($entry_{real}$);
5.       $R_s$ := calculateRegion($R_p$, $entry_{real}$.MBS);
6.       $entry_{virtual}.code_{VBS}$ := calculateCode($R_p$, $R_s$);
7.       $R_p$ := reconstruct($R_p$, $entry_{virtual}.code_{VBR}$);
8.    end;
      // the top of subtree is reached
9.    $entry_{real}$ := routeOutput(path, realpart);
10.   $entry_{virtual}$ := correspond($entry_{real}$);
11.   $R_c$ := calculateRegion($R_p$, $entry_{real}$.MBR);
12.   $entry_{virtual}.code_{VBR}$ := calculateCode($R_p$, $R_c$);
13.   $R_s$ := calculateRegion($R_p$, $entry_{real}$.MBS);
14.   $entry_{virtual}.code_{VBS}$ := calculateCode($R_p$, $R_s$);
      // adjust the code of descendant recursively
15.   adjustCode($entry_{virtual}$.node, $entry_{real}$.node, $R_c$);
end.

Procedure adjustCode(VirtualNode $node_{virtual}$, RealNode $node_{real}$, Region $R_p$)
16.   for each $entry_{real} \in node_{real}$ do
17.      $entry_{virtual}$ := correspond($entry_{real}$);
18.      $R_c$ := calculateRegion($R_p$, $entry_{real}$.MBR);
19.      $entry_{virtual}.code_{VBR}$ := calculateCode($R_p$, $R_c$);
20.      $R_s$ := calculateRegion($R_p$, $entry_{real}$.MBS);
21.      $entry_{virtual}.code_{VBS}$ := calculateCode($R_p$, $R_s$);
22.      if $entry_{real}$.node is a non-leaf node then
23.         adjustCode($entry_{virtual}$.node, $entry_{real}$.node, $R_c$);
24.   end;
end.
```

Figure 5.7. Update algorithm.

needs more accesses than the structure which does not use the SCM.

In this case that the SCM applies to the SR-tree which contains MBRs and MBSs, coding not only MBRs but also MBSs is necessary for constructing virtual part. However, if MBR is unchanged and only center point of MBS is changed in an entry of node, only the entry must be updated. That is, because changes of center point of MBS in current node have no effect on calculating the code of lower nodes, descendants of the node do not have to be updated.

5.7 Deletion

Deletion needs update of both parts of structure just like insertion. Process for updating subspace code in virtual part will start after object deletion at real part and underflow handling of real part and virtual part are finished. Deletion uses the algorithm shown in Figure 5.7 as insertion does for updating subspace code.

While insertion and deletion cause accesses to both parts that more costs are required than usual, searching leads to fewer disk accesses with pruning by virtual part. More detailed discussion with experimental results will be presented in the next chapter.

5.8 Conclusions

In this chapter, the subspace coding method (SCM) has been proposed to accelerate search performance. In Chapter 4, a result which small fanout is a significant factor in decline of search performance for R-tree family, especially the SR-tree, was obtained. The SCM is motivated by this observation. Not only the number of non-leaf nodes but also that of leaf nodes can be reduced by increasing the fanout of non-leaf nodes for the SCM. In the concrete, for the SCM, virtual bounding region covers and approximates minimum bounding region and the required storage size for virtual bounding regions is smaller than that for minimum bounding regions. This leads to higher fanout, resulting in a superior search performance.

On the one hand higher fanout causes better performance, however on the other hand, the approximation error of non-leaf nodes leads to decline in the power of pruning subtrees in searching. Therefore, in the SCM, the subspace code represents the relative position of virtual bounding regions in terms of parent's region. Large reductions in approximation error can be achieved by the relative approximation of regions.

Chapter 6

Performance Evaluation

6.1 Introduction

To verify the effectiveness of the SCM, the SCM and competitive algorithms have been implemented and the SCM has been compared with the VA-File, the SR-tree and the R*-tree. The SR-tree and the R*-tree were used for constructing the real part in the SCM.

The tests were conducted under the following conditions:

- Basically, the performance is measured under the conditions described in Chapter 2. These conditions includes: the real data sets and synthesis data sets, all parameters such as page size and fanout of the ordinary SR-tree and the ordinary R*-tree; and the number of nearest neighbors in queries, which is 20.

- The length l for the subspace code varies 4, 6, 8, 10, 12 (i.e. radix q varies 16, 64, 256 1024 and 4096).

- For the evaluations of the VA-File, the most superior approximation file from among the three variations, $l = 4$, $l = 6$ and $l = 8$, is chosen according

Table 6.1. Fanout of non-leaf nodes (the SCM).

Structure		Dimensionality							
		4	8	16	24	32	40	48	56
R*-tree	l = 4	511	340	204	146	113	92	78	68
	l = 6	409	255	146	102	78	63	53	46
	l = 8	340	204	113	78	60	48	40	35
	l = 10	292	170	92	63	48	39	32	28
	l = 12	255	146	78	53	40	32	27	23
SR-tree	l = 4	227	170	113	85	68	56	48	42
	l = 6	194	136	85	61	48	40	34	29
	l = 8	170	113	68	48	37	30	26	22
	l = 10	151	97	56	40	30	25	21	18
	l = 12	136	85	48	34	26	21	17	15

to [WSB98].

- The fanout of non-leaf node, when the SCM applied to the R*-tree and the SR-tree, is shown in Table 6.1. Table 6.2 shows the height of the structures of the SCM for the code length $l = 8$ as an example.

Table 6.2. Height of tree structures (the SCM for $l = 8$).

Structure	Dimensionality							
	4	8	16	24	32	40	48	56
R*-tree	3	3	3	4	4	4	4	4
SR-tree	3	3	4	4	4	4	5	5

6.2 Search Performance

6.2.1 Assignment of Code Length

Code Length for the SCM

Figure 6.1, Figure 6.2 and Figure 6.3 show the comparisons of the SCM with the VA-File, the SR-tree. In the comparisons , the code length $l(\in \{4, 6, 8, 10, 12\})$ of the SCM which gives the best search performance for each dimension, have been examined. For the code of length l in Figure 6.2 and Figure 6.3, the optimum code length is $l = 12$ for 4 and 8 dimensions, $l = 6$ for dimensionality from 16 to 56.

In Figure 6.1, the code of length $l = 8$ for 4, 8 and 16 dimensions, $l = 4$ for dimensionality from 24 to 56, have been chosen. As described in Chapter 4, uniformly distributed data sets degrade search performance of tree structure indices, especially, for high dimension. The structure using the code of smaller length can be more profitable on high-dimensional space since most of non-leaf nodes in tree structure are visited. According to this observation, in contrast to real data set and cluster data set of which data objects are distributed non-uniformly, the SCM has the disposition that the structure using the code of smaller length saves more page accesses in uniformly data set.

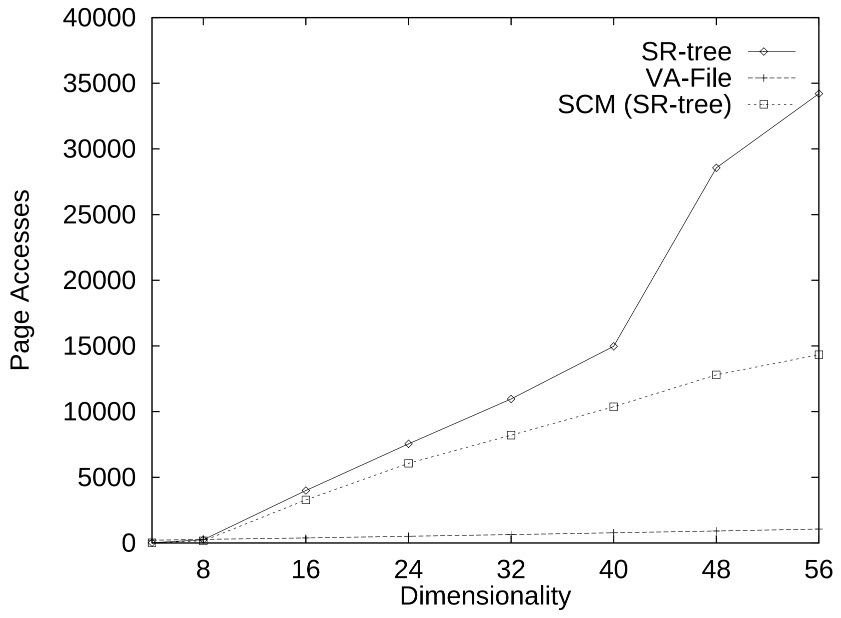


Figure 6.1. Number of page accesses with varying dimensionality for search (Uniform random data).

Code Length for the VA-File

For the VA-File, the most superior approximation file from among the three variations, $l = 4$, $l = 6$ and $l = 8$, is chosen. In the experience with real data set, $l = 8$ is chosen for dimensionality from 4 to 8 and $l = 6$ is the best for dimensionality from 16 to 56. In connection with cluster data set, the optimum code length is $l = 6$ for all dimensions. And also, in Figure 6.1 by uniformly distributed data set, the code of length $l = 6$ for 4 and 8 dimensions, $l = 4$ for dimensionality from 16 to 56, have been chosen.

For uniformly distributed data set, each object which consists the data set is stored in the corresponding cell and distributed uniformly. Therefore, the mechanism of the VA-File has the suitability for uniformly distributed data and

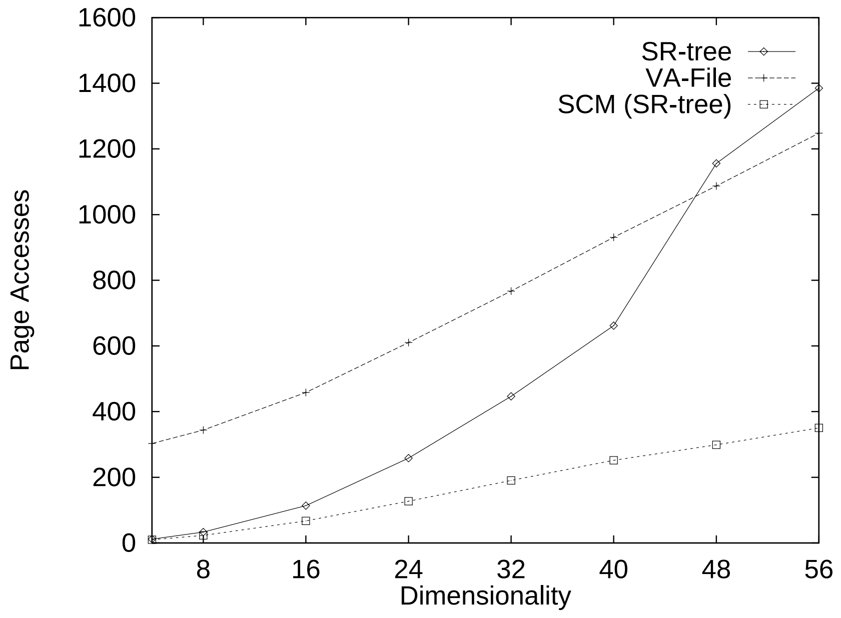


Figure 6.2. Number of page accesses with varying dimensionality for search (Real data).

it shows high performance on the data as presented in these figures. The code of small length is preferred by the VA-File since uniformly distributed data objects are pruned efficiently by not only close approximation but also crude one.

On the other hand, for real data or cluster data, many objects can be assigned to one cell since the distribution is non-uniform. The filter step with crude approximation in the VA-File can not prune objects effectively, thus, non-uniform data sets make the mechanism of the VA-File choose longer code than uniform data sets do. Non-uniform distribution leads to lowering performance in the VA-File as a result. For instance, Figure 6.1 shows almost 1060 page accesses for uniform data, however, Figure 6.2 and Figure 6.3 show almost 1250 page accesses for non-uniform data in 56 dimension.

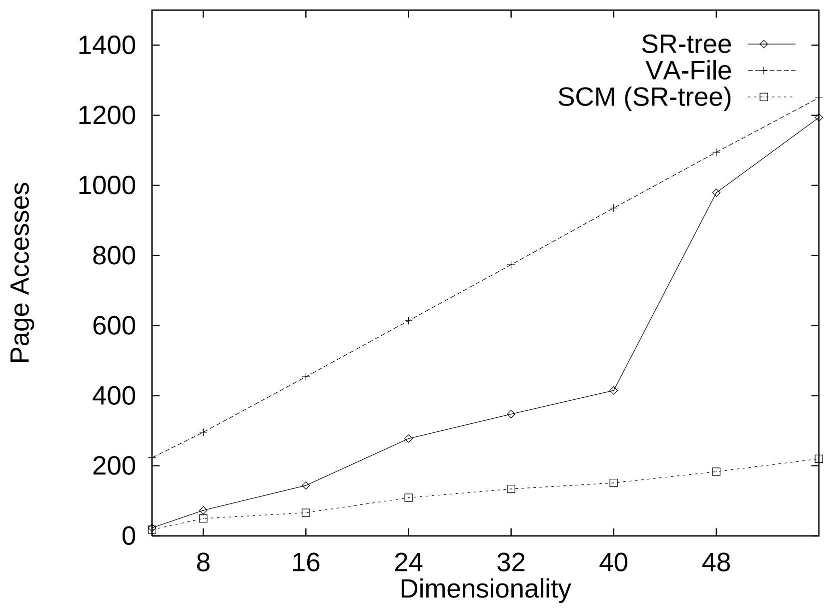


Figure 6.3. Number of page accesses with varying dimensionality for search (Cluster data).

6.2.2 Experimental Results

As Figure 6.1 presents, the SCM outperforms the SR-tree, however, it is inferior to the VA-File for uniformly distributed data sets. In contrast to the data sets, on Figure 6.2 by real data set and Figure 6.3 by cluster data set, the superiority of the SCM can be confirmed in all data sets ranging in dimensionality from 4 to 56. Although the performance of SR-tree degenerates in high dimensionality, the SCM has superiority over the VA-File. For example, the SCM achieves about 62.0 % (74.7 %, resp.) saving over the SR-tree in 40 (56, resp.) dimension and 71.9 % saving over the VA-File in 56 dimension on real data set. The SCM is highly effective for non-uniformly distributed data as real data in particular, and it can support content-based retrieval powerfully.

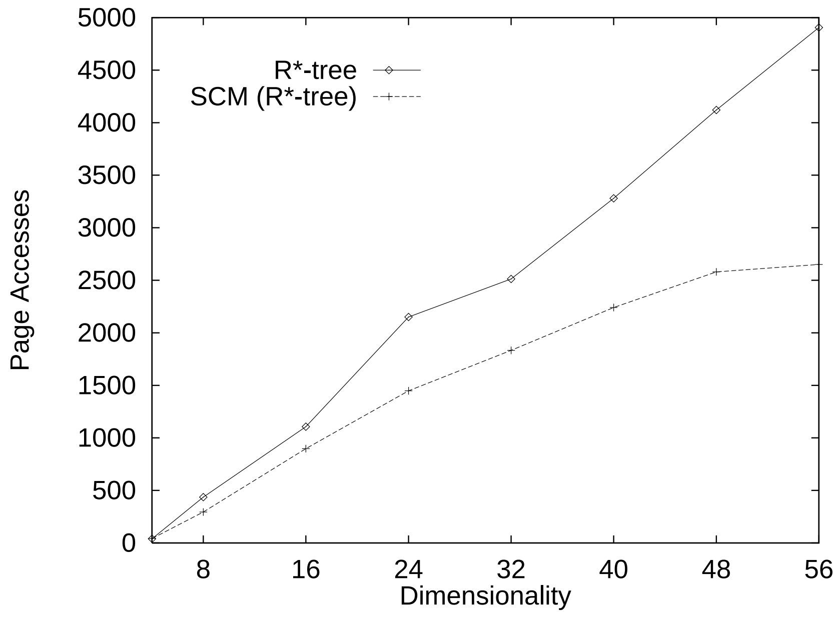


Figure 6.4. Number of page accesses with varying dimensionality for the R*-tree and the SCM (Real data).

In addition, the SCM is also effective when applied to the R*-tree. In this case, the SCM achieves about 32.4 % saving in 8 dimension and 46.0 % saving in 56 dimension over the ordinary R*-tree as shown in Figure 6.4. Next section is described the reason for the superiority of the SCM in detail.

6.3 Superiority of the SCM

6.3.1 Superiority of the SCM over the VA-File

First, the reason of the superiority of the SCM over the VA-File is explained. Approximation of position coordinate is the common idea of the VA-File and the SCM. However, the data structures and the algorithms of the VA-File and

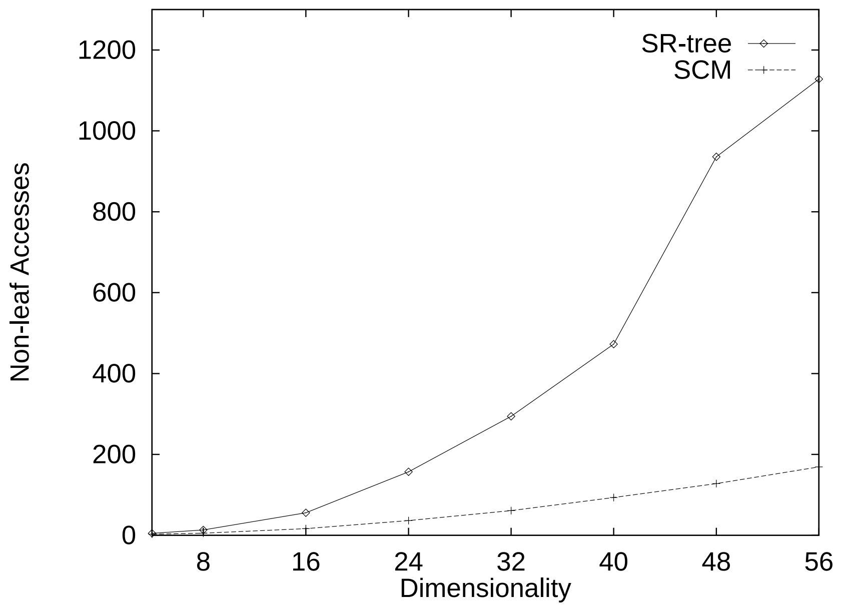


Figure 6.5. Number of non-leaf accesses with varying dimensionality for search.

the SCM are completely different, which results in much difference in search performance. The approximation calculated in terms of bounding region of parent nodes in the SCM has stratified architecture, which is in clear contrast to the approximation computed by the whole search space in the VA-File. That is, in the SCM, the approximation error becomes smaller as nodes become closer to leaves; consequently, the number of leaf accesses becomes smaller and the search cost is reduced.

6.3.2 Superiority of the SCM over the SR-tree

The comparison of the SCM with the original SR-tree is described as follows. Figure 6.5 and Figure 6.6 show the node accesses of the SCM and the SR-tree. As shown in Figure 6.5, the accesses of non-leaf nodes in the SCM are remarkably saved when compared with the SR-tree. Moreover, the difference between two

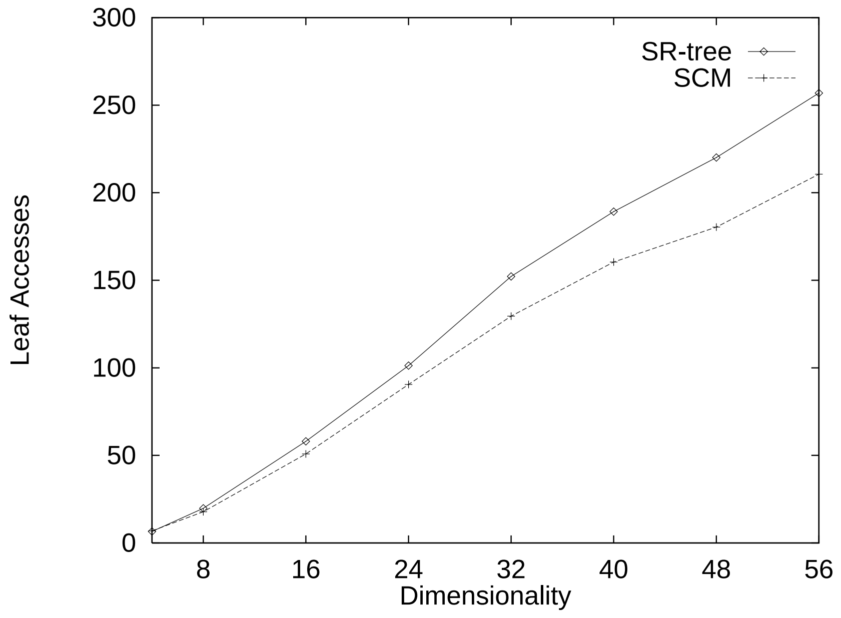


Figure 6.6. Number of leaf accesses with varying dimensionality for search.

curves grows as dimensionality increases. The reason is obvious from the second and third indications in Chapter 4. Storage cost of virtual bounding regions is small in connection with virtual part for search processing and this property for virtual bounding regions carries two advantages. One is the decrease in volume of non-leaf nodes and it leads to lower number of page accesses. The other is the increase in fanout. As a result, pruning capability is improved and accesses of non-leaf node are saved in tree structure as described in Chapter 4.

Moreover, Figure 6.6 shows that the SCM is superior to the SR-tree in leaf accesses. Although virtual bounding regions calculated by subspace code include error in size which causes lowering performance, increase in fanout of non-leaf nodes as shown in Table 6.1 leads to higher pruning capability. The effect of increase in fanout is larger than that of error for virtual bounding regions, and as a consequence not only non-leaf accesses but also leaf accesses are saved.

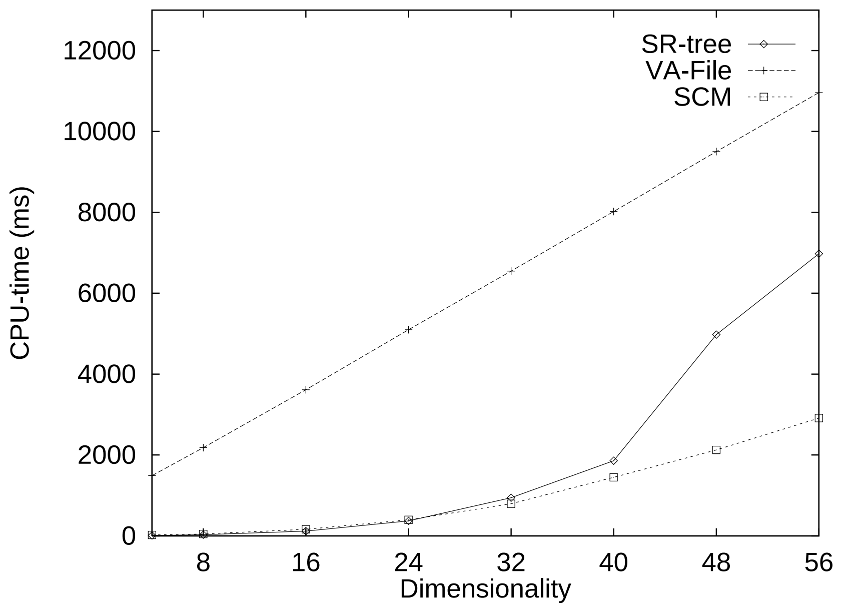


Figure 6.7. CPU-time for search with varying dimensionality.

6.3.3 Evaluation for CPU-time

Figure 6.7 shows measured the CPU-time for the VA-File, the SR-tree and proposed method, the SCM. For the SCM and the VA-File, CPU-time is measured under the same conditions as Figure 6.2. Since the VA-File requires calculating approximated position coordinates for all objects, the CPU-time is much higher as shown in Figure 6.7. On the other hand, although the SCM needs to calculation of virtual bounding regions by subspace codes, the CPU-time of the SCM is lower than the VA-File. The lower CPU-time is achieved by high pruning capability for the SCM. For the comparison of the SCM with the SR-tree, although the performance of the SCM is almost equal to that of the SR-tree up to 40, the SCM is highly superior to the SR-tree in 48 and 56 dimensions. In each node access, the distance of all entries contained in the node from a query

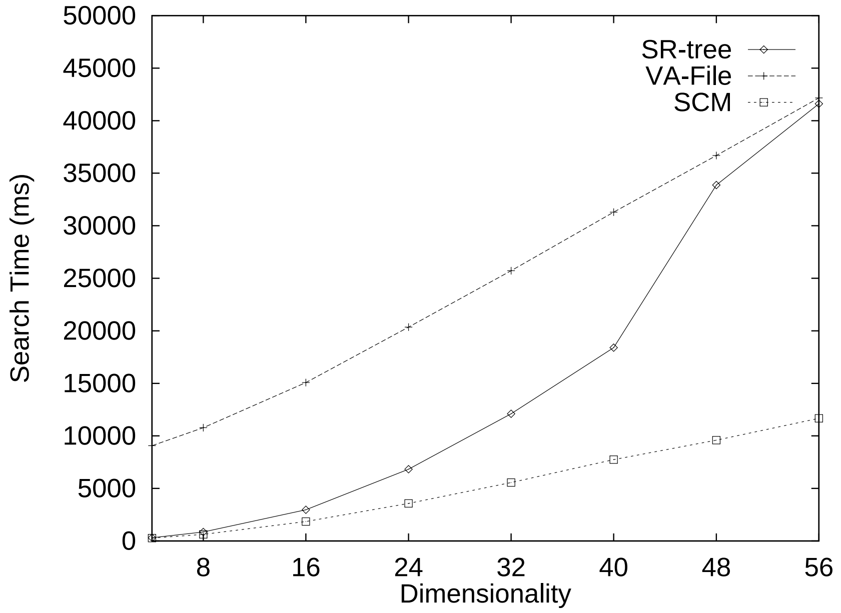


Figure 6.8. Total search time with varying dimensionality.

point, is calculated and inserted in priority queue, then the queue is sorted for the SCM and the original SR-tree. And also, unnecessary nodes are pruned using the queue. Since the number of node accesses is extremely low, calculations and comparisons of the distance from a query point are saved. Therefore, the SCM is fast for CPU-time although the SCM needs the calculation that bounding region is created from subspace code.

In addition, the SCM is more superior than the other methods, the VA-File and the SR-tree, relative to page accesses as shown in Figure 6.2. To discuss the total search time, Figure 6.8 is presented as the measurement based on the page access time which realistic system values is 25 ms in [BKK96]. Figure 6.8 shows prominent superiority of the SCM which achieves about 57.9 % saving in 40 dimension and 71.9 % saving in 56 dimension over the SR-tree, and also 72.3 % saving in 56 dimension over the VA-File.

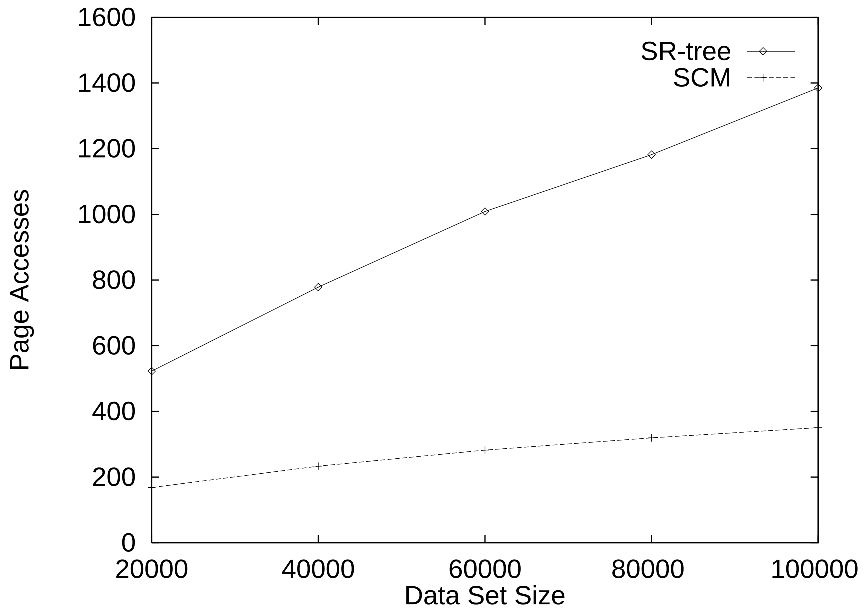


Figure 6.9. Number of page accesses with varying data set size for search.

6.3.4 Comparing with Varying Data Set Size

Figure 6.9 and Figure 6.10 show the comparisons of the SCM and the SR-tree for 56-dimensional data sets with varying the size from 20,000 to 100,000. Figure 6.9 shows the improvement of the page accesses which the SCM reaches 67.8 % saving for size 20,000 and 74.7 % saving for size 100,000. And also, on CPU-time, Figure 6.10 shows highly superiority of the SCM, especially, for larger size of data set. That is, since the performance of the SCM is more desirable as data size grows, this property is appropriate for large multimedia databases.

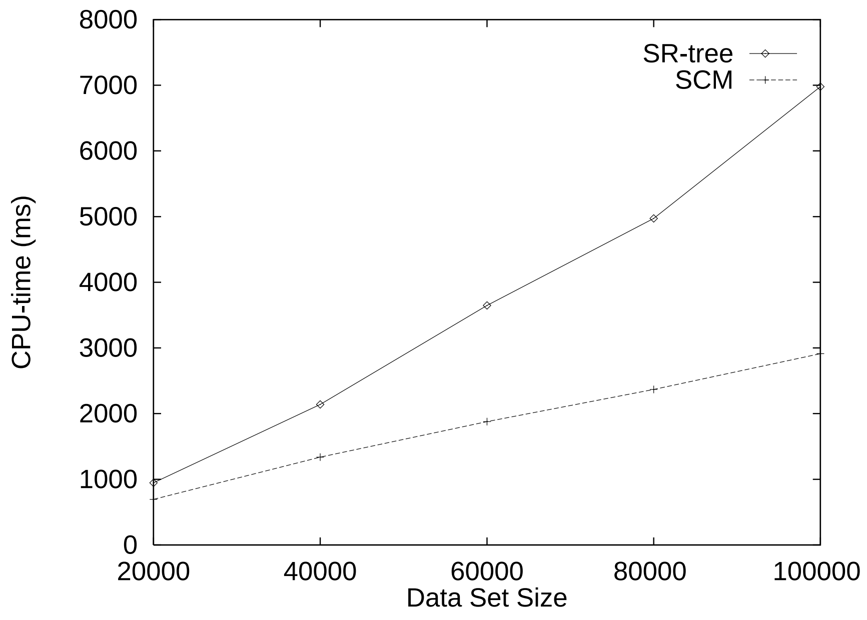


Figure 6.10. CPU-time with varying data set size for search.

6.4 Influence of Code of Variable Length on Search Costs

In this section, new observation from measurement with code of variable length is presented.

6.4.1 Approximation Error for Code of Variable Length

Figure 6.11 shows approximation error for VBRs against the number of dimensionality, in the SCM whose code of length $l = 4$, $l = 8$ and $l = 12$, where dimensionality is varied from 4 to 56 and data size is 100,000. Note that a logarithmic scale of the y-axis in this figure. The approximation error ϵ of tree

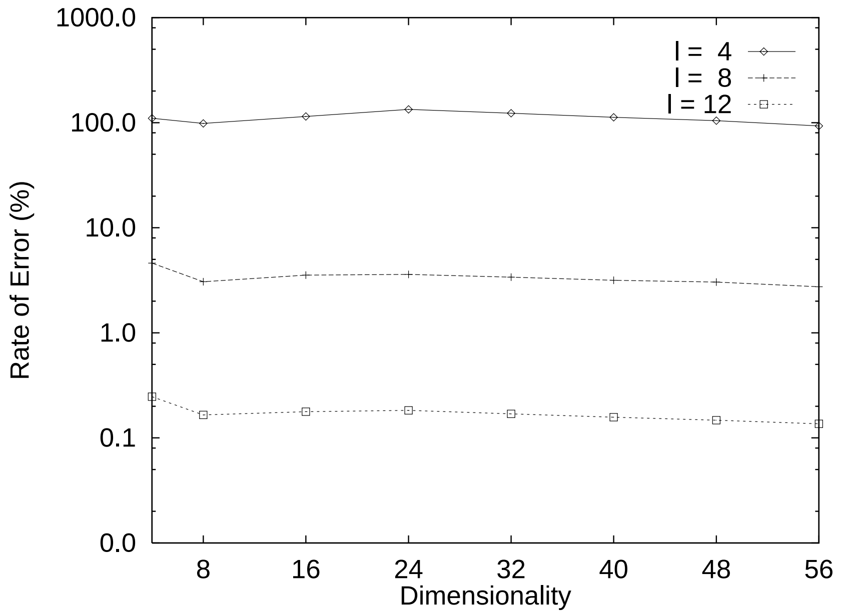


Figure 6.11. Approximation error with varying code length.

structures is calculated as follows:

$$\begin{aligned} \epsilon &= (r-1)\cdot 100 \\ r &= \frac{1}{P\cdot n}\sum_{i=1}^{P}\sum_{j=1}^{n}\frac{V_{i_j}}{R_{i_j}}, \end{aligned}$$

where P is the number of non-leaf nodes and n is dimensionality. V_{i_j} and R_{i_j} are the length of edge on dimension j for the i-th VBR and the i-th MBR, respectively.

In Figure 6.11, differences of approximation error arise between the structures for $l = 4$, $l = 8$ and $l = 12$. In these structures, the edge length of VBRs for the virtual part of $l = 4$ is two times as large as that of MBRs for the real part. The error lowers pruning capability for search.

On the other hand, in spite of the code of small length, the structure of $l = 8$ has only about 3 % error. Since the approximation error is small, the structure

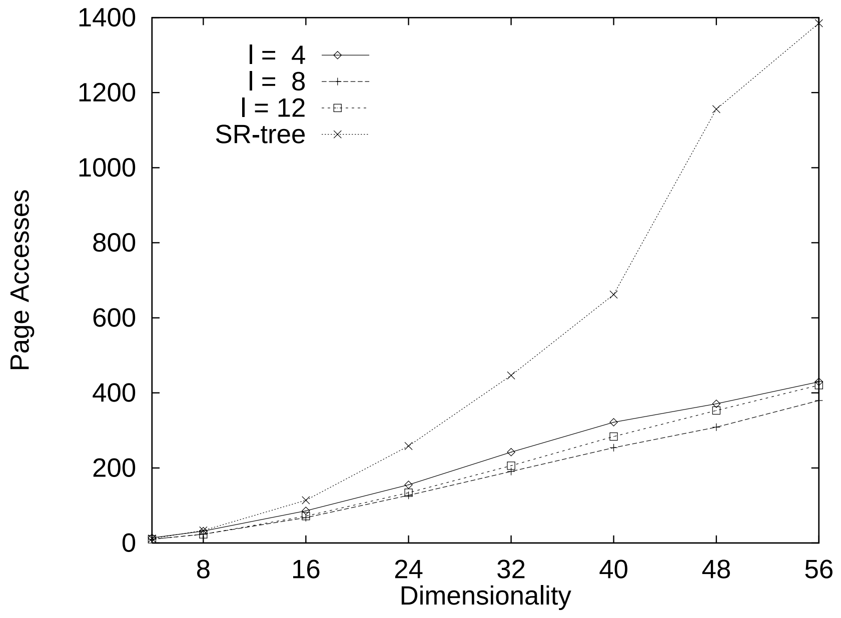


Figure 6.12. Number of page accesses with varying code length for search.

of $l = 8$ can reduce the influence of error on calculating the distance between a query point and regions, and then node pruning in the search algorithm does not reflect the influence of approximation error in this case.

6.4.2 Search Cost for Code of Variable Length

Figure 6.12 shows the number of page accesses against the number of dimensionality, for the SR-tree and each of 4, 8 and 12 bits as length of subspace code l in the SCM, where dimensionality is varied from 4 to 56 and data size is 100,000. Figure 6.13 and Figure 6.14 show the number of accessed non-leaf nodes and leaf nodes under the same condition.

The structures $l = 8$ and $l = 12$ outperform the original SR-tree for non-leaf nodes, leaf nodes, and page accesses. In contrast to these structures, $l = 4$

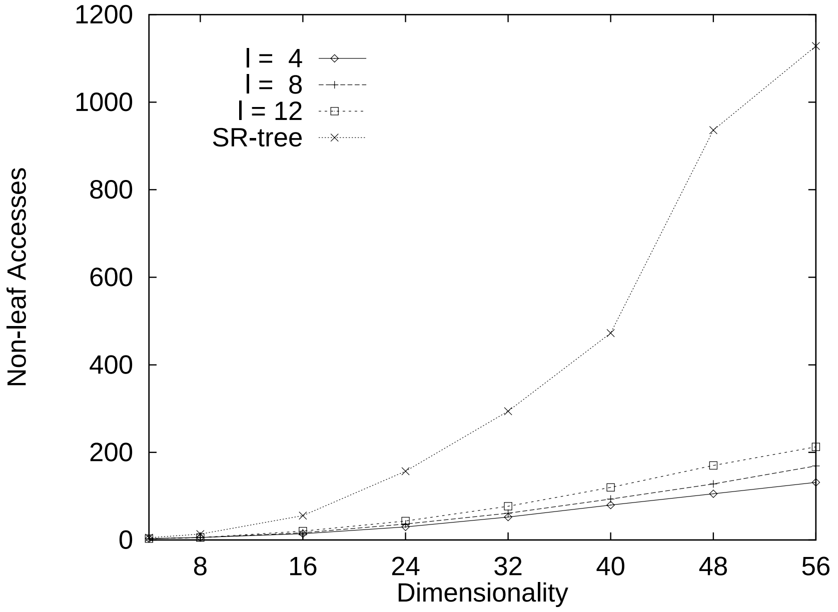


Figure 6.13. Number of non-leaf accesses with varying code length for search.

requires higher cost than the SR-tree only for leaf accesses shown in Figure 6.14. For non-leaf accesses shown in Figure 6.13 , the structure $l = 4$ achieves low cost similar to the structures, $l = 8$ and $l = 12$, although $l = 4$ needs high cost for leaf accesses. As a result of considerable low non-leaf accesses, $l = 4$ offers low cost for page accesses and desirable search performance besides $l = 8$ and $l = 12$.

There are the following two competing factors which influence search performance relative to length of subspace code. First, constructing rectangles by shorter code decreases the representation cost on position and size, and leads to larger saving effect and higher performance. Secondly, VBRs and VBSs generated by subspace code include error in size that is larger as code length decreases and causes lower performance. These factors cause to unusual appearance in two figures, Figure 6.12 and Figure 6.14. That is, although the number of visited leaf nodes in length $l = 4$ is remarkably high in comparison with that of the SR-

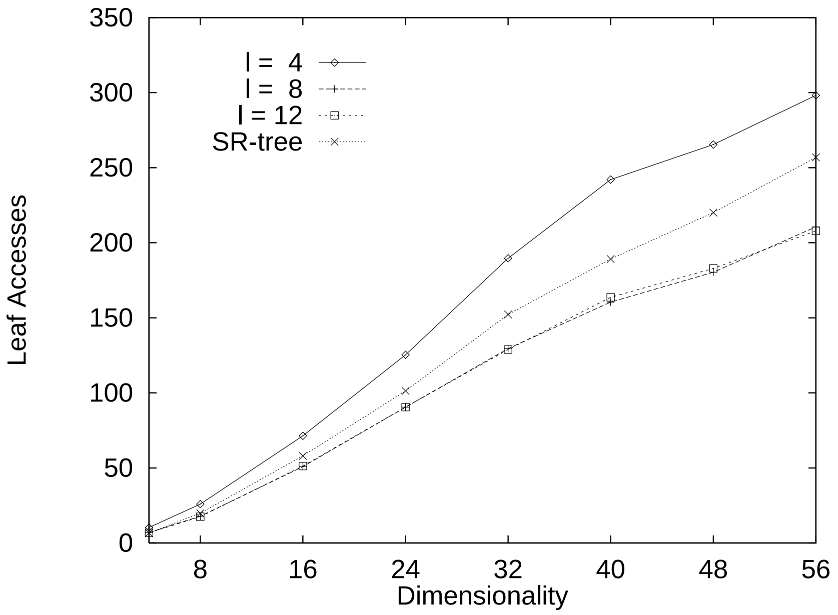


Figure 6.14. Leaf node accesses with varying code length for search.

tree on account of error of VBRs and VBSs, the structure of $l = 4$ outperforms the original SR-tree in page accesses. This reason is that representation cost of VBRs and VBSs is low; consequently, structure of virtual part takes small size and larger fanout. In spite of the result that the curves of $l = 8$ and $l = 12$ are relatively similar to that of the SR-tree in Figure 6.14, the searching costs are lower in Figure 6.12. These things make it clear that virtual bounding regions of $l = 8$ and $l = 12$ contain few errors; consequently, lower representation cost leads to high performance.

It follows from what has been said that the effect of lower representation cost for subspace code and the mechanisms accompanied by subspace code overcomes the influence of code error, and it leads to saving of page accesses and larger profit in the SCM.

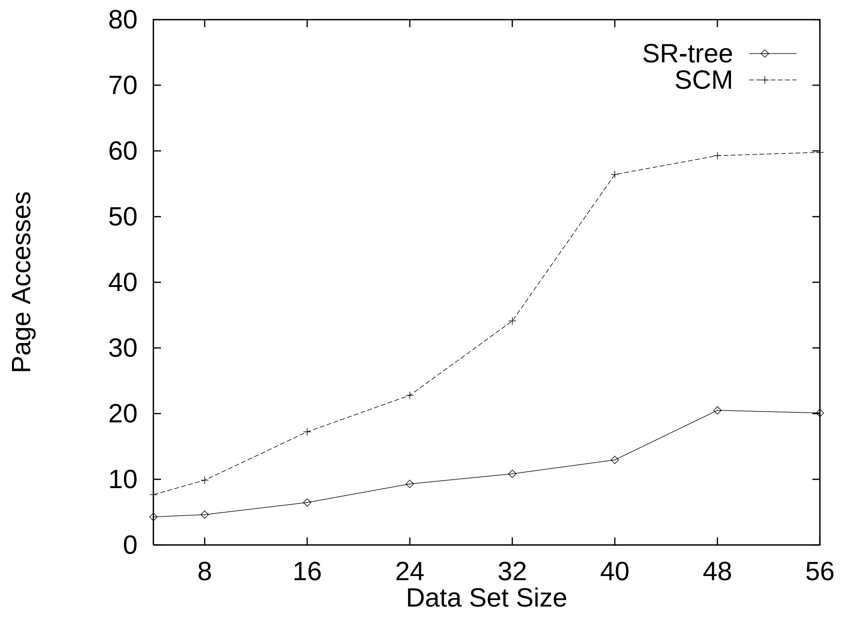


Figure 6.15. Number of page accesses with varying dimensionality for insertion.

6.5 Insertion Cost

Figure 6.15 shows a comparison of the SCM with ordinary SR-tree regarding insertion cost. The code length of the SCM in Figure 6.15 is the same as that of Figure 6.2. Insertion cost is measured as the average cost of inserting 1,000 randomly-selected objects. In this experiment, random objects have been used, since inserted objects are usually unpredictable in practical situation.

The SCM requires larger insertion cost than the SR-tree. In concrete, the increment of insertion cost for size 100,000, in average, is about 3 times in terms of the SR-tree for dimensionality from 16 to 56. For page accesses, if node split or forced reinsert does not occur, insertion cost of the SR-tree is equal to the height of tree, which implies only nodes on one insertion path are visited. In the SCM, it is necessary to access both real part and virtual part for maintenance of

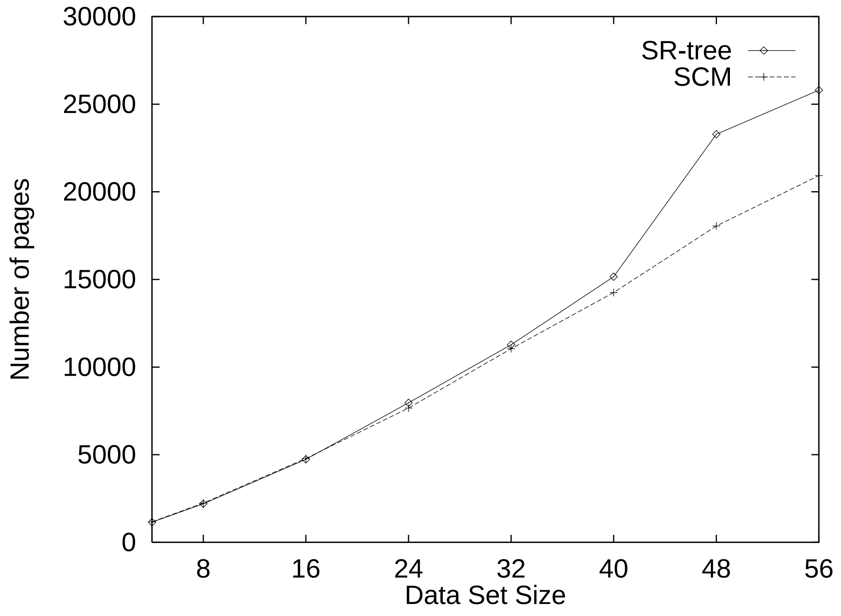


Figure 6.16. Storage cost.

structure and access real part for checking subspace code in virtual part. Hence the SCM incurs higher cost than the original tree structure.

However, in many cases, reducing search cost in multimedia databases has higher priority than reducing insertion cost. Thus, this method that can reduce search cost fits to the use in many practical situations.

6.6 Storage Cost

One of the strength of the SCM is low storage cost. Figure 6.16 shows a comparison of the storage cost of the SCM and the SR-tree under the same condition as Figure 6.2. In this figure, the storage cost of the SCM includes both the real and virtual parts. When compared with the SR-tree, the SCM incurs similar storage cost for dimensionality from 4 to 40, and achieves 18.9 % saving for 56 dimension. The storage cost of the SCM is low in spite of the structure that contains real and virtual part.

There are three reasons. First, the structure of virtual part does not include nodes corresponding to leaf nodes in real part. Secondary, the volume of virtual part is small. Thirdly, the number of non-leaf nodes in the SCM is extremely small on account of larger fanout. Since the storage cost and the search cost are low, the SCM is useful for various applications with content-based retrieval.

6.7 Conclusions

In this chapter, the performance of the SCM is compared with that of the competitive spatial search methods, the VA-File, the original SR-tree and the original R*-tree. For the comparisons, uniformly distributed data sets and real data sets and cluster data sets are used.

For the SCM and the VA-File, approximation of position coordinate is the common idea of these methods. However, the approximation calculated in terms of bounding region of parent nodes in the SCM has stratified architecture, which is in clear contrast to the approximation computed by the whole search space in the VA-File. As shown in the experiment, the approximation error becomes smaller for the SCM, thus, the search cost is markedly reduced.

Storage cost of subspace code for creating virtual bounding region is small. Since the volume of non-leaf nodes decreases, the number of page accesses for non-leaf nodes also declines. In addition, pruning capability for search grows on account of increase in the fanout for non-leaf nodes. The SCM achieves higher search performance than the original SR-tree for the advantages.

The SCM is an index scheme that applying to R-tree family using minimum bounding regions. Various dynamic index structures are discussed in this dissertation. By contrast dynamic index structure whose insertion or deletion cost is low, static structures are unfit for cases not only search cost but insertion cost is considered important. However, static structures can cut down search cost exceedingly.

The VAMSplit R-tree [WJ96a] is one of static structures. Since inspection of whole information for the position of each data object is required by this method, the use of this method is restricted within narrow limits that are cases where the volume of data set is small and the frequency of the update of database is low, however, its search performance is high. In the construction algorithm of the VAMSplit R-tree, a data set is partitioned into two child sets recursively by information on the position of each data object. The algorithm for constructing tree structure is similar to that of the k-d tree [Ben75]. The hyperplane which is orthogonal to the dimension with the higher variance, is a split plane. Unlike dynamic index structures, minimum bounding rectangles corresponding to entries in a node are calculated from the top node to the leaf nodes in the structure of the VAMSplit R-tree. [WJ96a] shows that the VAMSplit R-tree outperforms the R*-tree and the SS-tree for search performance. In [KS98], the static structure for the SR-tree is constructed by the algorithms, [Ben75] and [WJ96a], and it achieves high search performance. The SCM can use the algorithms, and it may be that the static structure for the SCM overcomes the dynamic one.

In addition, the evaluations of search performance in this dissertation are based on Euclidean distance. On the other hand, since spatial search based on general quadratic distance is useful [HSE+95] [ISF98], search algorithms for efficient processing of ellipsoid queries using tree structures have been proposed [SK97] [ABKS98] [Ish99]. These algorithms based on general quadratic distance can be also applied to the SCM and they may function effectively on the SCM. The evaluation of search performance of the SCM on general quadratic distance is important issue.

Chapter 7

Conclusions

7.1 Discussion and Conclusions

In this dissertation, spatial search methods for high-dimensional data have been discussed. And furthermore, the SCM, the subspace coding method has been proposed for high-dimensional spatial search, then the superiority of the SCM is presented through performance evaluations with some different conditions.

In Chapter 4, R-tree family and the VA-File that are useful spatial search methods has been analyzed. Although the VA-File shows superior performance for uniformly distributed data sets, the search cost slightly increases for non-uniformly distributed data sets that include real data sets and cluster data sets. By contrast, R-tree family degrades search performance for high-dimensional and uniformly data sets, however, good performance is achieved as non-uniformly data sets are used. For non-uniformly distributed data sets, especially, SR-tree overwhelms the R*-tree and offers better performance than the VA-File up to 40 dimension on account of high pruning capability using its non-leaf nodes. Despite the fact that the SR-tree is good at non-uniformly data, it degrades as dimensionality grows. In 48 dimension or higher, the SR-tree is inferior to the

VA-File. Through detailed analysis, it makes clear that large volume of non-leaf nodes in the SR-tree causes to increase page accesses and degrades the search performance.

In Chapter 5, the subspace coding method (SCM) is proposed. The SCM is able to solve the problem discussed in Chapter 4 and achieves highly search performance. Subspace code in the SCM is represented by relative positions in terms of parent node. Virtual bounding regions are constructed based on the subspace codes. One important feature of the SCM is the relative approximation of regions. Unlike the approximation of absolute vector positions used in the VA-File, the subspace code represents the relative position of regions in terms of parent's region. This leads to extremely small approximation error, resulting in a superior search performance. Moreover, the relative representation in the SCM is effective for non-uniformly distributed vectors.

Chapter 6 shows that the algorithms for search and update using virtual bounding regions achieve higher searching performance than ordinary methods, the VA-File and the SR-tree, through the performance test. Experimental results with real data set of which size is 100,000 show that the SCM outperforms the SR-tree with reductions of 74.7 % and the VA-File with reductions of 71.9 % for dimensionality 56.

The SCM is orthogonal to any other method using minimum bounding region like as R-tree family in the field of spatial search. That is, the SCM is applicable to all structures and algorithms that use minimum bounding region (i.e. the R-tree, the R*-tree, the X-tree, the SR-tree and so on), and it causes improvement of search performance.

Subspace code approximates a minimum bounding region, however, results as searching is not approximate solution, that is, the SCM finds desired objects without missing out. The search performance is greatly improved by the SCM, on top of that, the storage cost is low. Thus, this method fits to the use in practical situation.

7.2 Future Work

In this dissertation, the SCM is proposed and the superiority of the SCM is presented, and then various spatial search methods are discussed. However, there still remain some issues. In the following, such issues are discussed.

7.2.1 Cost Model for the SCM

First issue is to provide cost model for accurate estimates of search cost on the SCM. For nearest neighbor query, various cost models have been proposed so far [FBF77] [Cle79] [BBKK97] [LCC99]. Cost models of index structure are serviceable for estimates of execution time on search, optimizing the parameters of structures and query optimization. Although the models proposed in [FBF77] and [Cle79] are unfit for the estimate on high-dimensional spatial search, the model of [BBKK97] which takes *boundary effects* into account and therefore also works in high dimensions. Moreover, [LCC99] presents successful model for cost estimation of k-nearest neighbor queries.

In connection with the SCM, a new model is required for the estimation of search execution time and optimizing code length as the parameters of index structure. In order to elaborate accurate model, two factors, approximation error in virtual bounding region and effect of increasing fanout by coding, should be considered.

7.2.2 Spatial Queries Based on Elliptical Distance

The evaluations of search performance in this dissertation are based on Euclidean distance. On the other hand, since spatial queries based on general quadratic distance are useful [HSE$^+$95] [ISF98], search algorithms for efficient processing of ellipsoid queries using tree structures have been proposed [SK97] [ABKS98] [Ish99]. These algorithms based on general quadratic distance can be also applied to the SCM and they may function effectively on the SCM. Moreover, the SCM may be superior to other methods on elliptical distance like the superiority of the SCM for search performance on Euclidean distance. The evaluation of search performance of the SCM on elliptical distance is important issue.

References

[ABKS98] Mihael Ankerst, Bernhard Braunmüller, Hans-Peter Kriegel, and Thomas Seidl: "Improving Adaptable Similarity Query Processing by Using Approximations", in *Proc. of the 24th International Conference on Very Large Data Bases (VLDB)*, New York City, NY, August 1998.

[AMN$^+$94] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu: "An Optimal Algorithm for Approximate Nearest Neighbor Searching", in *Proc. of ACM-SIAM Symposium on Discrete Algorithms*, pp. 573–582, 1994.

[AMN$^+$98] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu: "An Optimal Algorithm for Approximate Nearest Neighbor Searching Fixed Dimensions", *J. of the ACM*, Vol. 45, No. 6, pp. 891–923, November 1998.

[And84] T. Anderson: *An Introduction to Multivariate Statistical Analysis*, John Wiley, 1984.

[BBKK97] S. Berchtold, C. Bohm, D. A. Keim, and H.-P. Kriegel: "A Cost Model For Nearest Neighbor Search in High-Dimensional Data Space

", in *Proc. ACM Symp. on Principles of Database Systems*, pp. 1–12, March 1997.

[Ben75] Jon Louis Bentley: "Multidimensional Binary Search Trees Used for Associative Searching", *Comm. of the ACM*, Vol. 18, No. 9, pp. 509–517, September 1975.

[BFG$^+$96] Jeffrey R. Bach, Charles Fuller, Amarnath Gupta, Arun Hampapur, Bradley Horowitz, Rich Humphrey, Ramesh Jain, and Chiao-Fe Shu: "Virage Image Search Engine: An Open Framework for Image Management", in *Proc. of SPIE: Storage and Retrieval for Image and Video Databases IV*, Vol. 2670, pp. 76–87, 1996.

[BKK96] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel: "The X-tree: An Index Structure for High-Dimensional Data", in *Proc. of the 22nd International Conference on Very Large Data Bases (VLDB)*, pp. 28–39, Bombay, September 1996.

[BKS93] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger: "Efficient Processing of Spatial Joins Using R-trees", in *Proc. ACM SIGMOD International Conference on Management of Data*, pp. 237–246, May 1993.

[BKSS90] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger: "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles", in *Proc. ACM SIGMOD Conf.*, pp. 322–331, Atlantic City, NJ, May 1990.

[BKSS94] Thomas Brinkhoff, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger: "Multi-Step Processing of Spatial Joins", in *Proc.*

ACM SIGMOD International Conference on Management of Data, pp. 197–208, May 1994.

[BU77] Rudolf Bayer and Karl Unterauer: "Prefix B-Trees", *ACM Trans. on Database Systems*, Vol. 2, No. 1, pp. 11–26, March 1977.

[Cle79] J. H. Cleary: "Analysis of an Algorithm for Finding Nearest Neighbors in Euclidean Space", *ACM Trans. on Mathematical Software*, Vol. 5, No. 2, pp. 183–192, June 1979.

[Com79] D. Comer: "The Ubiquitous B-Tree", *ACM Computing Surveys*, Vol. 11, No. 2, pp. 121–138, June 1979.

[Dim97] A. Dimai: "Spatial Encoding using Differences of Global Features ", in *Proc. of SPIE: Storage and Retrieval for Image and Video Databases V*, Vol. 3022, pp. 352–360, 1997.

[Fal96] Christos Faloutsos: *Searching Multimedia Databases by Content*, Kluwer Academic, 1996.

[Fal97] Christos Faloutsos: "Searching Multimedia Databases by Content ", in *International Symposium on Digital Media Information Base (DMIB'97)*, November 1997.

[FBF77] Jerome H. Friedman, Jon Louis Bentley, and Raphael A. Finkel: "An Algorithm for Finding Best Matches in Logarithmic Expected Time", *ACM Trans. on Mathematical Software*, Vol. 3, No. 3, pp. 209–226, September 1977.

[FBF$^+$94] Christos Faloutsos, Ron Barber, Myron Flickner, Jim Hafner, Wayne Niblack, Dragutin Petkovic, and William Equitz: "Efficient and Effective Querying by Image Content", *Journal of Intelligent Information Systems*, Vol. 3, No. 3/4, pp. 231–262, July 1994.

[FBY92] William B. Frakes and Ricardo Baeza-Yates, editors: *Information Retrieval – Data Structures & Algorithms –*, Prentice-Hall, 1992.

[FR89] Christos Faloutsos and Shari Roseman: "Fractals for Secondary Key Retrieval", in *Proc. ACM Symp. on Principles of Database Systems*, pp. 247–252, March 1989.

[Fre87] M. Freeston: "The BANG File: A New Kind of Grid File", in *Proc. ACM SIGMOD Conf.*, p. 260, San Francisco, CA, May 1987.

[FSA+95] Myron Flickner, Harpreet S. Sawhney, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, and Peter Yanker: "Query by image and video content: the QBIC system", *IEEE Computer*, Vol. 28, No. 9, pp. 23–32, September 1995.

[Fuk90] K. Fukunaga: *Introduction to Statistical Pattern Recognition*, Academic Press, 1990.

[GG98] Volker Gaede and Oliver Güther: "Multidimensional Access Methods ", *ACM Computing Surveys*, Vol. 30, No. 2, pp. 170–231, June 1998.

[Gue93] O. Guenther: "Efficient Computation of Spatial Joins", in *Proc. IEEE Int'l. Conf. on Data Eng.*, p. 50, Vienna, Austria, April 1993.

[Gut84] A. Guttman: "R-Trees: A Dynamic Index Structure for Spatial Searching", in *Proc. ACM SIGMOD Conf.*, pp. 47–57, Boston, MA, June 1984, Reprinted in M. Stonebraker, Readings in Database Sys., Morgan Kaufmann, San Mateo, CA, 1988.

[Har92] D. Harman: “Relevance Feedback and Other Query Modification Techniques”, in W. B. Frakes and R. Baeza-Yates, editors, *Information Retrieval – Data Structures & Algorithms –*, chapter 11, pp. 241–263, Prentice-Hall, 1992.

[HGH$^+$97] Arun Hampapur, Amarnath Gupta, Bradley Horowitz, Chiao-Fe Shu, Charles Fuller, Jeffrey R. Bach, Monika Gorkani, and Ramesh Jain: “Virage Video Engine”, in *Proc. of SPIE: Storage and Retrieval for Image and Video Databases V*, Vol. 3022, pp. 188–198, 1997.

[HS92] Eric Hoel and Hanan Samet: “A Qualitative Comparison study of Data Structure for Large Line Segment Database”, in *Proc. ACM SIGMOD International Conference on Management of Data*, pp. 205–214, June 1992.

[HS95] G. R. Hjaltason and H. Samet: “Ranking in Spatial Database”, in *Proceedings of the 4th Symposium on Spatial Databases*, pp. 83–95, Portland, Maine, Aug. 1995.

[HSE$^+$95] James L. Hafner, Harpreet S. Sawhney, William Equitz, Myron Flickner, and Wayne Niblack: “Efficient Color Histogram Indexing for Quadratic Form Distance Functions”, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 17, No. 7, pp. 729–736, July 1995.

[HSW88a] A. Hutflesz, H. W. Six, and P. Widmayer: “The Twin Grid File: A Nearly Space Optimal Index Structure”, *Lecture Notes in CS*, Vol. 303, p. 352, April 1988.

[HSW88b] A. Hutflesz, H. W. Six, and P. Widmayer: “Twin Grid Files: Space Optimizing Access Schemes”, in *Proc. ACM SIGMOD Conf.*, p. 183, Chicago, IL, June 1988.

[ISF98] Yoshiharu Ishikawa, Ravishankar Subramanya, and Christos Faloutsos: "MindReader: Querying databases through multiple examples", in *Proc. of the 24th International Conference on Very Large Data Bases (VLDB)*, pp. 218–227, New York City, NY, August 1998.

[Ish99] Yoshiharu Ishikawa: "Analysis of an Elliptical Distance Approximation Method for Efficient Similarity-based Retrieval", in *Proceedings of Data Engineering Work Shop '99*, 1999, In Japanese.

[KF94] Ibrahim Kamel and Christos Faloutsos: "Hilbert R-tree: An Improved R-tree using Fractals", in *Proceedings of the Twentieth International Conference on Very Large Databases*, pp. 500–509, Santiago, Chile, 1994.

[KS88] H. P. Kriegel and B. Seeger: "PLOP-Hashing: A Grid File without Directory", in *Proc. IEEE Int'l. Conf. on Data Eng.*, p. 369, Los Angeles, CA, February 1988.

[KS97] Norio Katayama and Shin'ichi Satoh: "The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries", in *Proc. ACM SIGMOD International Conference on Management of Data*, pp. 369–380, May 1997.

[KS98] Norio Katayama and Shin'ichi Satoh: "Application of Multidimensional Indexing Methods toward Massive Processing of Multimedia Information", in *Proc. of 4th Symposium on Intelligent Information Media*, pp. 133–140, December 1998, (in Japanese).

[LCC99] Ju-Hong Lee, Guang-Ho Cha, and Chin-Wan Chung: "A Model for k-Nearest Neighbor Query Processing Cost in Multidimensional Data

Space", *Information Processing Letters*, Vol. 69, No. 2, pp. 69–76, January 1999.

[Li97] R. Li: "Mobile Mapping: An Emerging Technology for Spatial Data Acquisition", *Photogrammetric Engineering & Remote Sensing(PE&RS)*, Vol. 63, No. 9, pp. 1085–1092, September 1997.

[LJF94] King-Ip Lin, H. V. Jagadish, and Christos Faloutsos: "The TV-Tree: An Index Structure for High-Dimensional Data", *The VLDB Journal*, Vol. 3, No. 4, pp. 517–542, October 1994.

[MCM92] B. Manjunath, R. Chellappa, and C. Malsburg: "A Feature Based Approach to Face Recognition", in *Proc. IEEE Conf. on CVPR*, pp. 373–378, 1992.

[MN95] H. Murase and S. Nayar: "Visual Learning and Recognition of 3-D Objects from Appearance", *International Journal of Computer Vision*, Vol. 14, No. 1, pp. 5–24, 1995.

[MP99] Nikos Mamoulis and Dimitris Papadias: "Integration of Spatial Join Algorithms for Processing Multiple Inputs", in *Proc. ACM SIGMOD International Conference on Management of Data*, 1999.

[NBE$^+$93] Wayne Niblack, Ron Barber, William Equitz, Myron Flickner, Eduardo H. Glasman, Dragutin Petkovic, Peter Yanker, Christos Faloutsos, and Gabriel Taubin: "The QBIC Project: Querying Images by Content, Using Color, Texture, and Shape", in *Proc. of SPIE: Storage and Retrieval for Image and Video Databases*, Vol. 1908, pp. 173–187, 1993.

[NHS84] J. Nievergelt, H. Hinterberger, and K. C. Sevcik: "The Grid File: An Adaptable, Symmetric Multikey File Structure", *ACM Trans. on Database Sys.*, Vol. 9, No. 1, p. 38, March 1984.

[NN96] S. Nene and S. Nayar: "Closest Point Search in High Dimensions", in *Proc. IEEE Conf. on CVPR*, pp. 859–865, 1996.

[NS86] Randal C. Nelson and Hanan Samet: "A Consistent Hierarchical Representation for Vector Data", in *Proc. of ACM SIGGRAPH*, pp. 197–206, Dallas, August 1986.

[NS87] R. C. Nelson and H. Samet: "A Population Analysis for Hierarchical Data Structures", in *Proc. ACM SIGMOD Conf.*, pp. 270–277, San Francisco, CA, May 1987.

[Oja83] E. Oja: *Subspace Methods of Pattern Recognition*, Research Studies Press, 1983.

[Ouk85] M. Ouksel: "The Interpolative-Based Grid File", in *Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Sys.*, p. 20, Portland, OR, March 1985.

[PMS94] A. Pentland, B. Moghaddam, and T. Starner: "View-Based and Modular Eigenspaces for Face Recognition", in *Proc. IEEE Conf. on CVPR*, pp. 84–91, Seattle, Washington, June 1994.

[RKV95] N. Roussopoulos, S. Kelley, and F. Vincent: "Nearest Neighbor Queries", in *Proc. ACM SIGMOD International Conference on Management of Data*, pp. 71–79, May 1995.

[RLS93] D. Rotem, J. Li, and J. Srivastava: "Algorithms for Loading Parallel Grid Files", in *Proc. ACM SIGMOD Conf.*, p. 347, Washington, DC, May 1993.

[Rob81] J. T. Robinson: "The K-D-B-Tree: A Search Structure For Large Multidimensional Dynamic Indices", in *Proc. ACM SIGMOD Conf.*, p. 10, Ann Arbor, MI, April-May 1981.

[Sam84] H. Samet: "The Quadtree and Related Hierarchical Data Structures ", *ACM Computing Surveys*, Vol. 16, No. 2, p. 187, June 1984.

[SAT99] T. Satou, A. Akutsu, and Y. Tonomura: "Video Corpus Construction and Analysis", in *Proc. of IEEE International Conference on Multimedia Computing and Systems (ICMCS)*, pp. II–479–485, Florence, June 1999.

[SK97] Thomas Seidl and Hans-Peter Kriegel: "Efficient User-Adaptable Similarity Search in Large Multimedia Databases", in *Proc. of the 23rd International Conference on Very Large Data Bases (VLDB)*, pp. 506—515, Athens, August 1997.

[SRF97] Timos Sellis, Nick Roussopoulos, and Christos Faloutsos: "Multidimensional Access Methods: Trees Have Grown Everywhere", in *Proc. of the 23rd International Conference on Very Large Data Bases (VLDB)*, pp. 13—14, Athens, August 1997.

[SW84] Hanan Samet and Robert E. Webber: "On Encoding Boundaries with Quadtrees", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 6, No. 3, pp. 365–369, May 1984.

[SW85] Hanan Samet and Robert E. Webber: "Storing A Collection of Polygons Using Quadtrees", *ACM Trans. on Graphics*, Vol. 4, No. 3, pp. 182–222, July 1985.

[SYU98] Yasushi Sakurai, Masatoshi Yoshikawa, and Shunsuke Uemura: "High-Dimensional Nearest Neighbor Search Based on Virtual Bounding Rectangles", in *Proc. of The 5th International Conference of Foundations of Data Organization (FODO'98)*, pp. 258–267, November 1998.

[SYU99] Yasushi Sakurai, Masatoshi Yoshikawa, and Shunsuke Uemura: "Nearest Neighbor Search based on Virtual Bounding Rectangles for Multi-Dimensional Datasets", *Transactions of Information Processing Society of Japan*, Vol. 40, No. SIG3(TOD1), pp. 68–79, February 1999, (in Japanese).

[TP91] M. Turk and A. Pentland: "Face Recognition Using Eigenfaces", in *Proc. IEEE Conf. on CVPR*, pp. 586–591, 1991.

[TY98] Noburou Taniguchi and Masashi Yamamuro: "Multiple Inverted Array Structure for Similar Image Retrieval", in *Proc. of IEEE International Conference on Multimedia Computing and Systems (ICMCS)*, pp. 160–169, June 1998.

[WJ96a] David A. White and Ramesh Jain: "Similarity Indexing: Algorithms and Performance", in *Proc. of SPIE: Storage and Retrieval for Image and Video Databases IV*, Vol. 2670, pp. 62–73, 1996.

[WJ96b] David A. White and Ramesh Jain: "Similarity Indexing with the SS-tree", in *Proc. of IEEE 12th International Conference on Data Engineering*, pp. 516–523, 1996.

[WKSS96] H. D. Wactlar, T. Kanade, M. A. Smith, and S. M. Stevens: "Intelligent Access to Digital Video: Informedia Project", *IEEE Computer*, Vol. 29, No. 5, pp. 46–52, May 1996.

[WNM+95] Jian-Kang Wu, A. Desai Narasimhalu, Babu M. Mehtre, Chian-Prong Lam, and Yong Jian Gao: "CORE: A Content-Based Retrieval Engine for Multimedia Information Systems", *Multimedia Systems*, Vol. 3, No. 1, pp. 25–41, February 1995.

[WSB98] Roger Weber, Hans-J. Schek, and Stephen Blott: "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces", in *Proc. of the 24th International Conference on Very Large Data Bases (VLDB)*, pp. 194–205, New York City, NY, August 1998.

List of Publications

Journal Paper

1. Yasushi Sakurai, Masatoshi Yoshikawa and Shunsuke Uemura: "Nearest Neighbor Search based on Virtual Bounding Rectangles for Multi-Dimensional Datasets", *Transactions of Information Processing Society of Japan*, Vol. 40, No. SIG3 (TOD1), pp. 68-79, February 1999, (in Japanese).

2. Yasushi Sakurai, Masatoshi Yoshikawa, Shunsuke Uemura and Haruhiko Kojima: "An Indexing Technique Using Relative Approximation for High-Dimensional Data", *IEICE Transactions on Information and Systems*, (in Japanese)(submitted).

International Conference

3. Yasushi Sakurai, Masatoshi Yoshikawa and Shunsuke Uemura: "High-Dimensional Nearest Neighbor Search Based on Virtual Bounding Rectangles", *Proc. of The 5th International Conference of Foundations of Data Organization (FODO'98)*, pp. 258-267, November 1998.

4. Yasushi Sakurai, Masatoshi Yoshikawa, Shunsuke Uemura and Haruhiko Kojima: "The Subspace Coding Method: A New Indexing Scheme for

High-Dimensional Data", *The 16th IEEE International Conference of Data Engineering (ICDE 2000)*, February 2000, (submitted).

Domestic Conference

5. Yasushi Sakurai, Masatoshi Yoshikawa and Shunsuke Uemura: "High Dimensional Nearest Neighbor Searching by R-tree Using Bit Coding", *IPSJ SIG Notes*, 98-DBS-116-66 (Vol. 98, No. 58), pp. 303-310, also in *Proc. of Fukui Workshop of 'Scientific Research on Priority Areas: Research and Development of Advanced Database Systems for Integration of Media and User Environments'*, pp. 426-433, July 1998, (in Japanese).

6. Yasushi Sakurai, Masatoshi Yoshikawa, Shunsuke Uemura and Haruhiko Kojima: "Similarity Search by Coding for High-Dimensional Data", *IPSJ SIG Notes*, 99-DBS-119-18 (Vol. 99, No. 61), pp. 103-108, July 1999, (in Japanese).

Other Publications

Journal Paper

7. Yasushi Sakurai and Shunsuke Uemura: "3D Measurement of Artifacts and Database of Cylindrical Range Images", *Japan Society for Archaeological Information*, Vol. 2(1), 1996, pp. 64-69, March 1997, (in Japanese).

Domestic Conference

8. Yasushi Sakurai, Hidehiko Iwasa, Haruo Takemura, Naokazu Yokoya and Takashi Kato: "Face Recognition by Eigen-space Method Using Cylindrical

Range Data", *IEICE Technical Report*, PRU95-193 (Vol. 95, No. 469), pp. 23-28, January 1996, (in Japanese).

9. Yasushi Sakurai, Mitsuhiko Okayasu, Naokazu Yokoya, Shunsuke Uemura: "3D Measurement of Artifacts and Database for Archaeological Information", *Japan Society for Archaeological Information*, pp. 80-85, March 1996, (in Japanese).

10. Yasushi Sakurai, Hidehiko Iwasa, Haruo Takemura, and Naokazu Yokoya: "Robust Face Recognition Regardless of Facial Direction and Expression Using Range Images", *Meeting on Image Recognition and Understanding (MIRU'96)*, pp.I-181-186, July 1996, (in Japanese).