Indexing and storage techniques for satellite images in a parallel architecture

Wu, Li

2006

Wu, L. (2006). Indexing and storage techniques for satellite images in a parallel architecture. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/2536>

<https://doi.org/10.32657/10356/2536>

Nanyang Technological University

Downloaded on 26 Apr 2023 19:19:31 SGT

INDEXING AND STORAGE TECHNIQUES FOR SATELLITE IMAGES IN A PARALLEL ARCHITECTURE

WU LI

**SCHOOL OF COMPUTER ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY**

2006

QA
76.58
W959
2006

Indexing and Storage Techniques for Satellite Images in a Parallel Architecture

Wu Li

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Master of Engineering

2006

ACKNOWLEDGEMENT

Acknowledgement

Firstly, I want to thank my supervisor Dr. Timo Rolf Bretschneider for his emotional support, encouragement and academic guidance. My graduate study could not be finished successfully without his supervision.

Moreover, I would like to sincerely thank the team members of the $(RS)^2I$ group, Li Yikun, Liu Bo, Vladimir Buntilov and Lee Ken Yoong for their support. I feel lucky for having had the opportunity to work in such a young and creative team.

Thanks to all the other friends in the Parallel and Distributed Computing Centre, NTU. They make my stay an enjoyable period of time.

Finally, I want to embrace and thank my MUM. I always remember her saying: "No matter what happens, I will stay with you."

The copyright for all shown satellite images in this thesis is with Space Imaging, Japan. Imagery was obtained by the IKONOS satellite.

Contents

Acknowledgements	i
Abstract	x
1 Introduction	1
1.1 Motivation	1
1.2 Objective	4
1.3 Organisation of the Thesis	5
2 Literature Review	6
2.1 Content-Based Retrieval of Images	6
2.2 Image Databases	7
2.3 Techniques for High Dimensional Indexing	8
2.3.1 Quadtree	9
2.3.2 R-Tree	12
2.3.3 R*-Tree	17
2.3.4 SS-Tree	20

CONTENT

2.3.5	SR-Tree	21
2.4	Review of Current Image Databases and Approaches	23
2.4.1	QBIC	23
2.4.2	IDIRS	23
2.4.3	Titan	25
2.4.4	GeoBrowse	26
2.4.5	Clustered SS-Tree System	27
2.4.6	SRSI	27
2.4.7	Summary	28
3	Preliminaries	31
3.1	Features with Varying Characteristics	31
3.1.1	Spectral Features	31
3.1.2	Texture Features	32
3.1.3	Varying Characteristics	33
3.2	VP-Tree	34
3.3	Earth Mover's Distance	38
3.4	De-clustering Techniques on Parallel Platforms	40
3.4.1	Random De-clustering	40
3.4.2	Round Robin De-clustering	41
3.4.3	Proximity Index De-clustering	41
4	Comparative Study of R-Tree Family	42

CONTENT

4.1	Introduction	42
4.2	Data Sets and Experimental Platform	42
4.3	Query Performance Analysis	43
4.4	Insertion Costs	46
4.5	File Storage	47
4.6	Summary	47
5	VP-EMD Tree	49
5.1	Introduction	49
5.2	VP-EMD Tree	50
5.3	Experimental Results	51
6	Approximative Indexing	54
6.1	Indexing Structure	54
6.2	Middle Subset Trade-off	55
6.3	Experimental Results	56
7	Data Distribution	64
7.1	Parallelisation of Indexing Strategy	64
7.2	New De-clustering Method	65
7.3	Retrieval Algorithm	67
7.4	Experiments	68
7.4.1	Comparison of De-clustering Speeds	68

CONTENT	
7.4.2	Comparison of Accuracy Performances 71
7.4.3	Retrieval Results 74
8	Conclusions 76
9	Possibilities for Future Directions 79
	Bibliography 82
	Acronyms 86
	Terminology 88
	Publications 89

List of Figures

1.1	Model of an image database	2
1.2	Prototype of the $(RS)^2I$	3
1.3	Modules of the $(RS)^2I$	4
2.1	Indexing structure of a 2-dimensional quadtree	10
2.2	Neighbours in a 2-dimensional quadtree	12
2.3	Indexing structure of a 2-dimensional R-tree	13
2.4	<i>MINDIST</i> and <i>MINMAXDIST</i> in 2-dimensional space	15
2.5	Example for overlaps in an R-tree	18
2.6	Example for 2-dimensional R*-tree	19
2.7	Example for a 2-dimensional SS-tree	21
2.8	Example for a 2-dimensional SR-tree	22
2.9	Web-based IDIRS system architecture	24
2.10	Graphical User Interface of GeoBrowse [21]	26
2.11	User interface of SRSI [17]	28
2.12	Querying result for SRSI [17]	29

LIST OF FIGURES

3.1	Original and classified satellite images	32
3.2	2-dimensional vantage point [9]	34
3.3	Using EMD as similarity measurement	39
4.1	Querying performance of inner nodes	44
4.2	Querying performance of leaf nodes	44
4.3	Querying performance of all nodes	45
4.4	Comparison of inserting costs	47
4.5	Comparison of file storage	48
5.1	Comparison of query time for texture teatures	52
5.2	Comparison of query time for spectral features	52
6.1	Middle subsets of the VP-EMD tree	55
6.2	Retrieval results without trade-off strategy	57
6.3	Retrieval results with middle trade-off strategy	57
6.4	Comparison of trade-off impact on query time for tecture features	57
6.5	Accuracy with ten nearest references for texture features	58
6.6	Accuracy with 50 nearest references for texture features	59
6.7	Comparison of trade-off impact on query time for spectral features	60
6.8	Accuracy with ten nearest references for spectral features	61
6.9	Accuracy with 50 nearest references for spectral features	61
6.10	Comparison of trade-off impact on query time for colour histogram	62

LIST OF FIGURES

6.11	Accuracy with ten nearest references for colour histogram	62
6.12	Accuracy with 50 nearest references for colour histogram	63
7.1	Parallel VP-EMD tree for three computing nodes	65
7.2	Planar schematic of the first VP-EMD tree level	66
7.3	De-clustering time for texture features using ten computing nodes	69
7.4	De-clustering time for texture features using 20 computing nodes	70
7.5	De-clustering time for spectral features using ten computing nodes	70
7.6	De-clustering time for spectral features using 20 computing nodes	71
7.7	Accuracy performance for texture features using ten computing nodes	72
7.8	Accuracy performance for texture features using 20 computing nodes	72
7.9	Accuracy performance for spectral features using ten computing nodes	73
7.10	Accuracy performance for spectral features using 20 computing nodes	73
7.11	Retrieval results (top-ten): (a) RR de-clustering strategy, (b) VP de-clustering strategy, (c) PI de-clustering strategy	75
9.1	Image database based on a Grid environment	80

List of Tables

2.1	Indexing result of quadtree	11
2.2	Comparative study of the introduced image database systems	29
3.1	Spectral feature	32
3.2	Texture feature	33
5.1	Building costs of a VP-EMD tree	53

Abstract

This thesis investigates multi-dimensional indexing techniques for content-based image retrieval, proposes a new indexing structure for an extended framework of features with varying dimensions / lengths, develops an effective approximative indexing strategy, and brings forward an effective de-clustering approach for a cluster architecture.

Spectral features and texture features are utilised for content-based satellite image retrieval. Compared with other features, they have a special characteristic due to the additional weight information, incomplete defined order and variable length. For these special properties, the conventional multi-dimensional indexing techniques, which employ partitioning methods to divide the multi-dimensional vector space into partitions, are unsuitable. Hence, a new structure is needed for more complex organised features.

Distributed database techniques are employed to manage huge amounts of satellite images. In order to distribute images to different computing nodes efficiently and effectively, a new de-clustering for spectral features and texture features is necessary.

In this thesis, four components regarding multi-dimensional indexing techniques are discussed. Firstly, a comparison and analysis of the R-tree and its family, that are among the most popular multi-dimensional indexing structures, is provided. The second part is the introduction to the proposed VP-EMD tree, which is a structure that is suitable for features with variable lengths and incorporates the concept of similarity. Thirdly, based on the VP-EMD tree, an effective approximative indexing algorithm is proposed that trades-off retrieval speed versus accuracy. Finally, an effective de-clustering strategy on a cluster architecture is brought forward. This approach improves the de-clustering speed significantly and provides a satisfactory de-clustering result.

Chapter 1

Introduction

1.1 Motivation

The rapid progress of computer technology has brought forward high quality equipment to capture digital pictures, one direct result is a boom in digital imagery. Currently, digital images are widely used in many research areas, such as remote sensing, medical imaging and entertainment. As a result, these huge amounts of image data led to the requirement for stable and effective database systems to manage the images.

Traditional database management systems can organise alphanumeric data in interrelated collections so that data retrieval and storage are convenient and efficient. However, the traditional database technology is not well suited to the management of multimedia information because of two main obstacles. The first obstacle is that traditional indexing techniques cannot directly describe the content of images precisely. For example, the acquisition date and the owner of an image can be clarified very well by text, while what is shown in an image is not describable straightforwardly. For example, an image of a sunset may be described quite differently by two persons. This problem results from the difficulty of capturing the abundant content of images using a small number of keywords, a difficulty which is compounded by the subjectivity of human perception [6].

Figure 1.1: Model of an image database

In order to overcome this obstacle, content-based image retrieval (CBIR) was proposed. The CBIR employs predefined functions to extract features from images automatically and uses these features to describe the content of the corresponding images. The colour histogram is one instance for effective features used in many CBIR systems. With this strategy, manual annotations are substituted by numerical features extracted by a computer.

The second obstacle for traditional database systems is that indexing techniques, such as the B-tree, can only be used for one dimensional data, while most CBIR systems employ multi-dimensional information for content-based retrieval.

Based on above introduction, a complete image database system stores images and supplies the users with effective query functionality to index the stored images. Such a system normally supports two kinds of queries, namely querying by text annotations and querying by content. A model of an image database is shown in Figure 1.1. The shown system is divided into two parts, i.e. the database system which contains the images and the part which interacts with the users to query images. In such a system, the indexing

Figure 1.2: Prototype of the $(RS)^2I$

engine, which is indicted by the dotted rectangle, is a major component.

The $(RS)^2I$ - Retrieval System for Remotely Sensed Imagery - project has the aim to provide satellite imagery to the non-professional remote sensing community. It is based on three fundamental issues: feature extraction and representation, multi-dimensional indexing and system design. The system classifies raw satellite images and extracts features, then the multi-dimensional indexing engine processes retrieval quests and sends back the result to an interface that handles the interaction with the users. Figure 1.2 shows the prototype of the $(RS)^2I$, while Figure 1.3 depicts the various modules and workflow.

One task to be solved in the design of the $(RS)^2I$ is to find an effective indexing technique for remotely sensed satellite images, since the data amount is very large. For example, in the near future it is expected that a satellite produces one Terabyte of image data per day. To manage such huge image data effectively, an efficient indexing technique is necessary. This requirement leads to the research on multi-dimensional

Figure 1.3: Modules of the $(RS)^2I$

indexing techniques.

From satellite images, the $(RS)^2I$ extracts spectral and texture features [19] that have varying lengths, partial order and additional weight information, these characteristics leads to the research on a new indexing strategy.

1.2 Objective

In order to solve the multi-dimensional indexing problem and accelerate the query performance, a number of data structures have been brought forward. These structures employ indexing trees. Currently, different tree structures are used in various projects. Since the $(RS)^2I$ needs an effective indexing technique for remotely sensed satellite images, a comparison and analysis of the current indexing structures is necessary. Therefore, one main objective of this thesis is to compare and analyse current indexing techniques.

The $(RS)^2I$ uses spectral and texture features from satellite images and employs them to compare the similarities among them. The special characteristics of the two features are that the elements in the features have no particular order and that they have an additional information in the form of weights. Moreover, the lengths of the

feature vectors are variable. These three attributes render traditional indexing structures focusing on partitioning the data according to coordinates unsuitable for the $(RS)^2I$. Therefore, a new indexing structure is needed. Hence, the second objective of this thesis is to develop an effective and efficient indexing structure for the described features.

Since the $(RS)^2I$ aims to handle huge amount of satellite images, a single processor architecture is obviously not suitable because of its limited capability for processing large amounts of data. Hence the third objective is to adapt the proposed approach to a parallel architecture.

Based on the above mentioned requirements, three objectives are summarized as: (1) comparison and analysis of current multi-dimensional indexing techniques, (2) developing an effective and efficient indexing structure for the variable lengths features employed in the $(RS)^2I$ project, and (3) implementing a parallelised version.

1.3 Organisation of the Thesis

This thesis is organised as follows: Chapter 2 provides the literature review. Chapter 3 illustrates the preliminaries. Chapter 4 compares and analyses current multi-dimensional indexing techniques. A new indexing structure namely the VP-EMD tree, which combines the advantage of the VP-tree [36] and the Earth Mover's Distance (EMD) [28], is introduced in Chapter 5 together with an in-depth analysis and performance evaluation. Subsequently, Chapter 6 proposes an approximate version using the trade-off to balance query speed against query accuracy. Chapter 7 discusses the parallelised version of the VP-EMD tree. Finally, conclusions are given in Chapter 8 followed by Chapter 9 on future research possibilities.

Chapter 2

Literature Review

2.1 Content-Based Retrieval of Images

In the information age, computers are not only equipped with networking devices, but also furnished with CD-ROMS, scanners, video cameras and various other multimedia equipments. The emergence of multimedia, the boom of large digital archives, and the rapid development of the Internet have greatly inspired research efforts in exploring systems for effective content-based image retrieval.

A large amount of research has been devoted to develop mechanisms for efficient retrieved images via content from an image database. In this so-called content-based retrieval, the user describes the desired content in terms of visual characteristics, and the system retrieves images that best match this description [37]. Therefore, content-based retrieval is a type of retrieval by similarity.

Image content can be defined on different levels of abstraction [16]. At the lowest level, an image is a collection of pixels. However, pixel-level content is seldom employed in retrieval tasks. Instead the raw data can be used to produce numeric descriptors capturing specific visual characteristics called features. Features are extracted from the whole image describing global visual characteristics or from portions of an image describing local

characteristics. In general, a feature-level representation of an image requires significantly less space than the image itself. Upon the features extracted from images, some distances, such as the Euclidean distance, are utilised to compute the similarity between images, i.e. their representations. An image with a smaller distance is taken as more similar than another image with a larger distance. Currently, content-based retrieval always means employing features extracted from images to identify the similarity. This thesis focuses on the feature-level retrieval.

2.2 Image Databases

As introduced in the previous section, with the development of modern computer and telecommunication technologies, huge archives of multimedia data have been created. The situation is similar to the rapid increase in the amount of alphanumeric data during the beginning of the computing age, which led to the development of database management systems (DBMS). Traditional DBMSs organise alphanumeric data into interrelated collections so that the information retrieval and storage become convenient and efficient. However, this approach is not well suited to the management of multimedia information. The variety of multimedia data and the large quantity of media objects are not fully supported by traditional database management techniques. In order to manage the widely available multimedia information effectively, efficient methods for storage, browsing and indexing have to be developed [5].

The general structure of a content-based image database system is shown in Figure 1.1, whereby the image database contains three kinds of data: raw images which are used for visual display, the numerical features extracted from images which are employed to support the content-based image retrieval, and the text annotations which can be used for common queries as in a traditional DBMS. The indexing engine speeds up the retrieval and makes the system scalable. For the users' convenience, content-based image

systems always employ a graphic interface to receive a query and to show the indexing results. The interface translates queries into internal form and submits them to the system. Furthermore, in order to improve the result, users can always interact with the system using their feedback.

The indexing engine deals with the huge amounts of data and accelerates the query performance. This thesis focuses on the indexing structure and its application.

2.3 Techniques for High Dimensional Indexing

Currently, most image database systems are similarity-based. Certain image feature spaces are used in these systems to represent images. Upon the spaces similarity metrics are defined to retrieve the images, whereby images are described as points or objects in the multi-dimensional spaces. For example, in SRSI [17], an image is represented by a 20-dimensional feature vector. A distance between two points determines the degree of similarity of the corresponding images.

Content-based image indexing systems commonly support two types of querying schemes: k -nearest neighbour search and range search. The first search approach searches the k -nearest data objects of corresponding images with respect to the reference point, while the latter search finds all the points, i.e. corresponding images, within a region of the multi-dimensional feature spaces specified by the user. With respect to the $(RS)^2I$ project, only the k -nearest neighbour search was studied since this is the most frequently utilised approach.

In the past decades, many indexing techniques have been adopted to speed up the indexing procedure. According to Gong [12], three attributes affect the performance of an indexing structure:

Dynamic or static: Dynamic indexing data structures support various indexing ma-

nipulations, i.e. insertion, update and deletion can be intermixed with queries. Contrary, static indexing structures require that the complete data sets and complete indexing strategies are given in advance.

Memory-resident or disk-resident: Memory-resident indexing structures create indices in the computer memory, while disk-resident indexing structures are not volatile, and can accommodate a large scale dataset.

High or low-dimensionality: Traditional indexing methods utilised in DBMSs, e.g. the B-Tree and its variants, can only retrieve one-dimensional data, while many recently proposed multi-dimensional indexes can manipulate a variable number of dimension. The number of dimensions affects the indexing performance and is a crucial factor which has to be considered in the design.

In this section, five representative indexing structures were discussed: quadtree, R-tree, R*-tree, SS-tree and SR-tree. The quadtree is the first structure proposed to index multi-dimensional data [3]. The R*-tree, SS-tree and SR-tree are variants of the R-tree, and these structures are most widely used in content-based image retrieval citebenchmark.

2.3.1 Quadtree

Quadtrees [3] define a hierarchical data structure whose common property is that they are based on the principle of recursive decomposition of the indexing space. It is a tree structure where each node is split along all d dimensions, leading to 2^d children. A quadtree consists of the following two types of nodes:

- 1: Non-leaf nodes: Each non-leaf node of a quadtree has a 2^d children, corresponding to the 2^d subquadrants of the region represented by the node.
- 2: Leaf nodes: Leaf nodes represent regions that consist of points that are not further subdivided.

Figure 2.1: Indexing structure of a 2-dimensional quadtree

In a quadtree, each leaf node provides a bucket in which to store data points, while each non-leaf node represents pointers to its subtrees, which is yielded as a result of data space partitioning. When a non-leaf node P is overfilled, the region represented by P is further partitioned, resulting in the splitting of P . There is no overlap between any pair of regions; thus any data point has only one path between itself and the root. Figure 2.1 shows an example for a 2-dimensional quadtree.

Let $D_0 * D_1 * \dots * D_{k-1}$ be the entire k -dimensional feature space. Generally, a balanced quadtree is provided with the following characteristics:

- 1: All leaf nodes appear on the same level.
- 2: In any of the non-leaf nodes, the regions are non-overlapping, and their union is a region.
- 3: In the root, the union of its region equals to the whole feature space, $D_0 * D_1 * \dots * D_{k-1}$.
- 4: Let (region, np) be a non-leaf node, and region₁, region₂, ..., region_n be the regions in the child nodes referred to by np; then region = region₁ $\cap$ region₂ $\cap$... $\cap$ region_n.

5: Let (region, np) be a non-leaf node, and point₁, point₂, ... ,point_n be the points in the leaf node referred to by np, then point_i ∈ region, 1 ≤ i ≤ n.

In the beginning, a 3-dimensional quadtree was established to study the basic features of this indexing technique. The database contained 1,000 uniformly distributed points. In this model, a term "neighbour" is introduced. In order to explain the idea easily, a 2-dimensional example is shown in Figure 2.2. The central node A has two kinds of neighbours, the nodes with gray colour and the nodes with dark colour. This idea can be generalized to a k-dimensional space (k>2). Similarly, a node in the 3-dimensional quadtree has three kinds of neighbours. The closest neighbours, the second closest neighbours and the third closest neighbours. With the definition and usage of neighbours in the quadtree, the indexing result is shown in Table 2.1. The first column lists the query points, the second column lists the indexing result using quadtree without any neighbours, the third column lists the result using quadtree with one shortest neighbour and the last column lists the result with all neighbours. For example, in the first row , the tuple (60,60,60) indicates the query point, 3/27 indicates that without any neighbour, the program retrieved three real most closest points after querying 27 points totally. From the result, it is shown that quadtree speeds the query for low dimensional data in a uniformly filled feature space.

Table 2.1: Indexing result of quadtree

Query Point	No/Check	One/Check	Complete/Check
60 60 60	3/27	6/108	10/216
89 98 79	8/27	10/54	10/521
100 100 100	10/27	10/27	10/521
156 157 155	8/8	8/8	10/320
201 202 200	0/0	5/18	10/50
75 86 95	8/27	10/54	10/521
142 156 187	5/8	5/8	10/320
168 178 205	0/0	0/0	10/128
245 86 87	8/27	10/54	10/329
198 253 86	6/27	9/45	10/200

Figure 2.2: Neighbours in a 2-dimensional quadtree

For a modern content-based image query system, the number of features is obviously more than three. In fact, it varies from ten to 100 or higher, and the points in the corresponding feature space are often clustered in small sub-spaces. In this situation, it becomes very difficult to balance the quadtree. Another insufficiency is that, for a d -dimensional quadtree, the amount of children is exponential with the number of the dimensions, therefore the speed of query is counteracted.

2.3.2 R-Tree

The R-tree is a dynamic indexing structure initially proposed for data objects of non-zero size in high-dimensional spaces [13]. It represents data objects by intervals in several dimensions. An R-tree is a balanced tree structure and is helpful to reduce the number of visited nodes [13]. If M represents as the maximum number of entries that can be fit into one node, and $m(2 \leq m \leq M/2)$ is the minimum number of entries in a node. The example for an R-tree can be seen in Figure 2.3.

A record entry of the form (I, np) is used to represent an internal node, where np is the address of a lower node in the R-tree and I is the minimum bounding rectangle that covers all the rectangles of entries included in the respective child nodes. The leaf node of an R-tree can be represented as (I, id) , where id refers to a data object of a child node,

Figure 2.3: Indexing structure of a 2-dimensional R-tree

while I is the minimum bounding n -dimensional rectangle that covers rectangles of all the entries in the child nodes.

An R-tree is established based on the clustering of all the data objects in the corresponding data space. Every internal node P denotes a mass of data objects with the corresponding boundary. When an internal node spills over, the splitting algorithm partitions this node into two or more lower nodes. For each node, the number of minimum branching factor is m , so the height of an R-tree containing N data objects is at most $\lceil \log_m N \rceil - 1$, and the maximum number of nodes is $\lceil \frac{N}{m} \rceil + \lceil \frac{N}{m^2} \rceil + \dots + 1$ [13].

Node Splitting

When a node contains more than M entries, it is split into two nodes. At present, three different algorithms, exhaustive algorithm [13], quadratic algorithm [13] and linear split

algorithm [29] were proposed for the splitting. Among the above algorithms, quadratic split algorithm has the best performance in building cost and indexing performance [12]. A quadratic cost algorithm is proposed to find a small area split, but is not guaranteed to find the smallest one, because the computational cost is exponential with M , the maximum number of entries that can be fit into one node, and the computational cost increases fast if M becomes large.

***K*-Nearest Neighbour Search**

In the original paper for R-tree [13], only range search was proposed. Based on it, the k -Nearest Neighbour search (k NN) [27] was built using the data structure of the R-tree. In k NN, some important terms are defined.

Definition : Given a point P in the n dimensional feature space $E(n)$ and an Minimum Bounding Rectangle (MBR) $R(S, T)$ in the same multi-dimensional space, let r_i , s_i , t_i and p_i represent for location on the i -th coordinate, the $MINDIST(P, R)$ is defined as:

$$MINDIST(P, R) = \sum_{i=1}^n |p_i - r_i|^2$$

where

$$r_i = \begin{cases} s_i & \text{if } p_i < s_i \\ t_i & \text{if } p_i > t_i \\ p_i & \text{if } otherwise \end{cases}$$

One important property of $MINDIST$ is that it is the possible minimum distance from a query point P to an data object contained in the closest facet of a MBR R .

Definition : Given a point P in n dimensional feature space $E(n)$ and an MBR $R = (S, T)$ in the same multi-dimensional space, let r_i , s_i , t_i and p_i represent for location on the i^{th} coordinate, the $MINMAXDIST(P, R)$ is defined as:

Figure 2.4: *MINDIST* and *MINMAXDIST* in 2-dimensional space

$$MINMAXDIST(P, R) = \min_{1 \leq k \leq n} (|p_k - rm_k|^2 + \sum_{i \neq k, 1 \leq i \leq n} |p_i - rM_i|^2)$$

where

$$rm_k = \begin{cases} s_k & \text{if } p_i \leq \frac{s_k + t_k}{2} \\ t_k & \text{if otherwise} \end{cases}$$

and

$$rM_k = \begin{cases} s_i & \text{if } p_i \geq \frac{s_i + t_i}{2} \\ t_i & \text{if otherwise} \end{cases}$$

One important property of *MINMAXDIST* is that it is the possible max distance from a query point P to an data object contained in the closest facet of a MBR R .

Figure 2.4 shows *MINDIST* and *MINMAXDIST* in a 2-dimensional space.

With the definition of *MINDIST* and *MINMAXDIST*, the algorithm for searching the k -NN objects through R-Tree can be stated as:

Step 1: Set the original k closest distances infinity in K Closest Distance(KCD).

Step 2: At internal levels, compute the distance to all child nodes(*MINDIST*) and order the distance from close to far, getting a Visiting Child Order(VCR).

Step 3: An MBR with *MINDIST* greater than the *MINMAXDIST* of k -th MBR M^k is discarded because it cannot contain the k -NN.

Step 4: Discard the MBRs with *MINDIST* greater than the k -th distance stored in KCD.

Step 5: At leaf nodes, discard the nodes with distances to the query point P greater than the *MINMAXDIST* of the k -th MBR. Compute the distance from query point P to the data objects and store the k closest objects and k closest distance in KCD.

Step 6: Repeat from Step 2 until all nodes are visited.

Insertion in R-Tree

The procedure of insertion in R-trees is described as finding the most suitable rectangle to contain the point that has to be inserted, splitting the corresponding rectangle if any overflow appears, and modulating other rectangles if needed. The whole strategy can be stated by four steps.

Step 1: From the root node R , compute the *MINDIST* from the insertion point P to the children MBRs of R , and set the MBR with shortest *MINDIST* as the new root. Repeat step 1 until leaf node is met.

Step 2: For the leaf node L got from step 1, if the number of points it contained is less than M (the max number of points can be set in one leaf node), insert the insertion point here. If overflow happened, use the split algorithm in the previous introduction to split node L and get new nodes L_1 and L_2 .

Step 3: If the parent node of L , denoted as P_L has enough space, add new pointers from P_L to new nodes, L_1 and L_2 , or split P_L to get P_{L1} and P_{L2} , pointing to L_1 , L_2 , respectively. If P_L is the root, go to Step 4, or iterate Step 3.

Step 4: If the root node meets an overflow, create a new root node to contain the new nodes got from the splitting of the old root.

Deletion in R-Tree

With the insertion algorithm above, the deletion can be implemented in two strategies. Find and delete the query point; Insert the other data objects in the corresponding rectangle if underflow happen. When node elimination occurs, this elimination is propagated upward if necessary.

Step 1: Use the k -NN algorithm to locate the query point P in the corresponding rectangle R , then delete it. If not found, exit.

Step 2: In the corresponding rectangle, if underflow occurs, transfer the other nodes in the corresponding rectangle to a previous set area S (the original set is empty) and delete R .

Step 3: Go upward to the parent node of R , $\text{Parent}(R)$, if this node meets an underflow, transfer the corresponding data objects to S and delete the children nodes of $\text{Parent}(R)$. Substitute the current $\text{Parent}(R)$ with $\text{Parent}(\text{Parent}(R))$.

Step 4: Repeat from step 2 if underflow occurs.

Step 5: Adjust the rectangles. To ensure that every rectangle is the minimum bounding for the data objects.

Step 6: Use insertion algorithm to insert the data objects in S .

2.3.3 R*-Tree

The R-tree was the first data structure proposed for high-dimensional feature space, but with the main disadvantage is that overlap can be found in the structure [4]. Figure 2.5 gives an example for the overlap of a R-tree.

Figure 2.5: Example for overlaps in an R-tree

The overlaps result in a spatial query that may have to visit several nodes before ascertaining the presence or absence of a particular rectangle. This leads to inefficient querying. In order to solve this problem, several variants have been proposed. Among them, the R*-tree [4] is a successful one. Figure 2.6 gives an example for an R*-tree in a 2-dimensional space.

According to the original paper on R*-trees [4], overlaps can be minimized by using an improved Node Splitting algorithm and Insertion algorithm. Before the two are introduced, some important properties are highlighted first. If there are two rectangles that are permitted to contain m (the minimum number) to M (the maximum number) data objects, and $M+1$ ordered data objects (from small to large, according to the value of a given axis) are distributed into the two rectangles, then the number of possible distributions is $\{(M+1) - m\} - m + 1 = M - 2m + 2$ since the minimum number of rectangles that can be contained in the first node is m , and the maximum number is $M + 1 - m$.

Node Splitting Algorithm

The node splitting algorithm of the R*-tree can be described as the following:

Step 1: For each axis of the k -dimensional feature space, arrange the data objects according to the value of that axis, from small to large.

Figure 2.6: Example for 2-dimensional R*-tree

Step 2: From the above discussion, $M - 2m + 2$ ways of distribution can be found.

Step 3: For all the distributions, the sum of margin-values are computed. The axis with the minimum margin-values are selected as the split axis.

Step 4: In the selected split axis, from the $M - 2m + 2$ different distributions, select the one which guides the smallest overlap. Resolve ties by adopting the one with the smaller area.

From the split algorithm, it is known that the overlap and useless margin is farthest avoided.

Insertion Algorithm

R*-trees also adopt an improved insertion algorithm to get better performance than R-trees. Instead of *MINDIST* or *MINMAXDIST*, the R*-tree considers the minimum

overlap enlargement to contain a data object.

Step 1: Set R to be the root of the R^* -tree.

Step 2: If R is a leaf node, insert the data object and use node split if needed.

Step 3: If R is a internal node, choose the rectangle N which needs the smallest overlap enlargement in order to contain the new data object. Resolve ties with the one needs smaller area enlargement and smaller area respectively.

Step 4: Set $R=N.np$, and repeat from Step 2.

2.3.4 SS-Tree

The SS-tree [33] has a similar configuration as the R-tree, but it employs minimum bounding spheres instead of rectangles. This difference leads to some better representations, such as better storage utilization, because only $n + 1$ values are needed to record the sphere while at least $2n$ values are needed to record a rectangle. Minimum bounding spheres also lead to a better cluster performance. Figure 2.7 gives an example for an SS-tree in the 2-dimensional space. In an SS-tree, the center of a sphere is a pair of $(centroid, radius)$, where centroid is the mean value of the feature vectors in its child nodes and radius is the distance from the centroid to the furthest feature vector. The split algorithm has to calculate the coordinate variance along each dimension from the centroid of its children and chooses the dimension with the highest variance for splitting. Then the split point has to be located in such a way that it minimises the sum of variances on each side of the split. For the insertion algorithm, the closest leaf node to the data object is selected to contain the new data. The split algorithm has to be used if an overflow occurs.

The main advantages of the SS-tree is that less information is needed for storing

Figure 2.7: Example for a 2-dimensional SS-tree

bounding spheres rather than bounding box. The disadvantage is that the bounding spheres create more overlap than bounding rectangles.

2.3.5 SR-Tree

The SR-tree [17] is an extension of the R*-tree and the SS-tree. The center of a bounding sphere $x(x_1, x_2, \dots, x_d)$ can be computed as:

$$x_i = \frac{\sum_{k=1}^n C_k.x_i * C_k.w}{\sum_{k=1}^n C_k.w} (1 \leq i \leq D),$$

where $C_k.x_i$ denotes the i -th coordinate of the center of the child C_k , and $C_k.w$ denotes the number of points contained in the subtree whose top is the child C_k .

The radius of a bounding sphere can be computed as:

$$r = \min(d_s, d_r)$$

where d_s is the maximum distance from the center of a parent node to the bounding

Figure 2.8: Example for a 2-dimensional SR-tree

spheres of its children and d_r is the maximum distance from the center of a parent node to the bounding rectangles for its children.

Generally, the idea can be taken as combining the advantages of R*-tree and SS-tree. Bounding rectangles permit to divide points into small-volume regions. However, they have much longer diameters than bounding spheres. Bounding spheres permit to divide points into short diameter regions. However, they tend to have larger volumes than bounding rectangles. Figure 2.8 is a SR-tree in the 2-dimensional space.

The leaf node of SR-tree can be denoted as $L: (E_1 \dots E_n)$ ($mL \leq n \leq ML$), E_i denotes (p, data) . Each entry contains a point p and its attribute data. An internal node of the SR-tree can be denoted as $N: (C_1, \dots, C_n)$ ($mL \leq n \leq ML$), where C_i denotes $(S, R, w, \text{child-pointer})$. Each entry contains a bounding sphere S , a bounding rectangle R , the number of points w and a pointer to the child_pointer.

2.4 Review of Current Image Databases and Approaches

In order to have a direct understanding of the image database systems, this section reviewed five content-based image database systems or approaches. They are QBIC, IDIRS, Titan, GcoBrowse, clustered SS-tree system and the similarity retrieval system of images (SRSI). The emphasis of the review is the individual indexing strategies adopted by the different systems.

2.4.1 QBIC

IBM's Query By Image Content (QBIC) is a famous content-based image database system in the world. QBIC allowed users to find pictorial information in large image databases based on colour, shape, texture and sketches.

QBIC utilises two components to achieve the functionality, namely database population and database query. The database population is the process of creating an image database. During the population, images are processed to extract features describing their content. During the query, the user composes a query graphically. Features are generated from the graphical query and then used as an input to the matching engine which finds the similar images.

QBIC employs a R^* -tree as the underlying structure to accelerate the query performance, that means, image data were sorted and stored in the R^* -tree.

2.4.2 IDIRS

IDIRS [8] is a web-based image database indexing and retrieval system. It uses content-based image retrieval techniques based on a Beowulf-Class parallel computing engine (Super Abacus) to effectively and efficiently shorten the retrieval time.

Figure 2.9: Web-based IDIRS system architecture

The IDIRS is a multi-dimensional image database indexing and retrieval system with 4,700 JPEG coded images. The overall architecture of the IDIRS is shown in Figure 2.9 and can be divided into three major parts: (1) image database management, (2) parallel computing and (3) web interface component.

The IDIRS supports the user to browse the whole image database and to edit or modify the attributes of images for indexing purposes. Users are allowed to make a query based on either keywords or a reference image for searching through the database. When a query based on keywords is entered, the database management server works according to the traditional image retrieval paradigm, i.e. the similar images are retrieved by their corresponding annotations. If the user proposes a content-based query by selecting a reference image, the database management server employs different content-based image retrieval algorithms according to the user's demand. The IDIRS supports three kinds of content-based retrieval methods: RGB colour histogram matching, YUV (a lu-

minance and chrominance based colour space) colour histogram matching and discrete Cosine transform (DCT) coefficients energy histogram matching in the JPEG domain. Since JPEG is the most commonly used format for true colour natural images, retrieval algorithms on the intermediate decoded DCT coefficients [18] can speed up the matching process compared to full decompression.

In the IDIRS, MySQL is adopted as the image DBMS. The user interface is developed with CGI in C language. In order to speed up the search process, all content-based image retrieval algorithms are parallelised.

The main problem of the IDIRS is that it does not take any specific indexing structures to accelerate the querying performance. So even though parallelisation is employed in IDIRS, the performance is not the most excellent [8].

2.4.3 Titan

Titan [7] is a parallel database designed for handling remote sensing data. It consists of a front-end, which interacts with the users, and a back-end that performs manipulations. The front-end consists of a single host computer and can be located anywhere on the network. The back-end is made up of a 16 processor IBM SP-2.

Titan partitions its data set into data blocks and uses a simplified R-tree to index these chunks, whereby the index is stored on the front end which is used to build a plan for the retrieval and processing of the required data blocks. This strategy has chosen a good performance for both large queries and small queries [7]. The Titan index consists of spatial bounds and retrieval keys for the data blocks. For expediency, a simplified R-tree was implemented. The leaf nodes were arranged in a z-ordering fashion [24] before the index was built. Sorting the leaf nodes spatially allows inner node keys in the index to better approximate the spatial extent of their children and reduces the overlap. As a

Figure 2.10: Graphical User Interface of GcoBrowse [21]

result, the searching performance is more efficient than without these modification [7].

2.4.4 GeoBrowse

GeoBrowse [21] is an effective integrated software system for image retrieval, browsing and data mining. It supports content based queries on large databases of remotely sensed images. GeoBrowse is based on the abstract services and distributed objects paradigm [21]. It consists of a Graphic User Interface (GUI), an object-relational database management system, and a scientific problem solving environment. An example for the GUI is shown in Figure 2.10.

GeoBrowse makes vast satellite data resources accessible to non-professional users. It extracts multiple features, i.e statistical attributes and properties for all images in the database and uses them for statistical indices. Multiple queries are posed on one or more of these features, to detect patterns or similarities in the entire image collection.

The GeoBrowse uses the Informix-Universal Server (IUS) database to manage the remote sensing images. IUS is an extensible object-relational database management

system that was designed to handle rich, complex data types. The VA-file [32] indexing technique was employed in GeoBrowse and increased the speed of similarity searches effectively.

2.4.5 Clustered SS-Tree System

The clustered SS-tree system [25] is an effective indexing model to retrieve high dimensional data and supports approximate nearest neighbour search.

The main idea is to cluster data into different computing nodes with a independent SS-tree as the underlying structure in each node, and then to apply filtering of index nodes based on the measurement of angles among high dimensional data points for efficient searching. This is because most of the subtrees are filtered out early during the retrieval process. However, this approach does not always guarantee a 100% correctness and focuses more on processing speed than retrieval accuracy.

2.4.6 SRSI

The Similarity Retrieval System of Images (SRSI) [17] was developed by the authors of the SR-tree as a proof of concept. In this system, similarity is measured in terms of colour histograms, i.e. the Hue-Saturation-Intensity colour space is used. It is divided into nine subspaces: black, grey, white and six further colours. Histograms of four sub-regions, i.e. upper-left, upper-right, lower-left and lower-right, are calculated in order to take account of the composition of images. Afterwards the four histograms are concatenated to compose a 36-dimensional feature vector. Then, feature vectors are reduced to 20-dimensional vectors with the principal component analysis. Similarity among images is measured by the Euclidean distance. This image database uses an SR-tree as indexing structure to accelerate the query performance.

Figure 2.11: User interface of SRSI [17]

Figure 2.11 shows the user interface of the SRSI. In the interface, a user selects a sample image to query the image database. Figure 2.12 shows the corresponding query result. The result is divided into two parts: the upper window shows the result images and CPU time as well as I/O accesses using the SR-tree structure. The lower window shows the corresponding result without any indexing structure. From the comparison of the two sub-windows, the precision performance of a SR-tree structure indicates to be effective while accelerating query performance significantly.

2.4.7 Summary

The above subsections introduces six different image database systems. Table 2.2 provides the brief summary among the different approaches.

The above systems showed that the design and implementation of an image database mainly focuses on two aspects: the features and the indexing technique. Dif-

Figure 2.12: Querying result for SRSI [17]

Table 2.2: Comparative study of the introduced image database systems

Database Systems	Indexing Structure	Data Size	Features
QBIC	R*-Tree	NA	Colour, shape and texture
IDIRS	None	4700 images	Colour, texture
Titan	R-Tree	24GB	Colour histogram
GeoBrowse	VA-file	NA	Colour, shape
Clustered SS	SS-Tree	8GB	NA
SRSI	SR-Tree	40745 images	Colour histograms

ferent image databases have different purposes and different users. Therefore, different features and indexing techniques are employed.

Indexing techniques are very useful for accelerating the query performance in image databases. The QBIC utilises the R*-tree to accelerate the query performance. The SRSI shows that with a SR-tree structure the query performance improved evidently in comparison to a solution without indexing structures. Titan also employs a simplified R-tree to accelerate the query performance, while the IDIRS does not use indexing techniques at all. However, the purpose of the latter systems is different since it was not designed for huge amount of data.

A detailed introduction of the used and further selected indexing techniques can be found in Chapter 4.

Chapter 3

Preliminaries

3.1 Features with Varying Characteristics

In developing a complete image database system, the feature space selection must be addressed since it determines what image features or combination of image features are used for image matching and retrieval purposes. In particular the choice of feature types affects the query performance. As a result it is also necessary to analyse the features before utilising a particular tree structure in order to determine a suitable approach for the acceleration of the query retrieval.

Content-based satellite image retrieval techniques utilise diverse features of satellite images to enable a powerful and straightforward procurement. In the $(RS)^2I$ project, two kinds of features were exploited, namely the spectral distribution and the textures of the equivalent grayscale image.

3.1.1 Spectral Features

Spectral features were extracted by locating the dominant spectral clusters of an image in the multispectral space [19]. For example, Figure 3.1 shows the original satellite images and the classified image. Table 3.1 gives the corresponding statistical information. In

Figure 3.1: Original and classified satellite images

Table 3.1: Spectral feature

Cluster No.	Cluster centre	Weight(Pixel amount)
1	89,24,45	69442
2	93,91,106	1940
3	85,53,71	18618

total three dominant spectral clusters were located and their centres were utilised to represent the corresponding clusters, the number of pixels corresponding to each cluster was utilised to represent the weight of the respective cluster. The second column of Table 3.1 depicts the spectral cluster centres.

3.1.2 Texture Features

Texture is a property of image regions. For satellite images, textures, such as urban areas, agricultural areas, water areas and mountain areas can easily be found. The texture information of the satellite images can provide enormous information for content-based satellite image retrieval. For the $(RS)^2I$ an effective algorithm to extract the texture features of satellite images was implemented. Table 3.2 shows the texture information for the image depicted in Figure 3.1 [19], whereby column two depicts the cluster centres.

Like spectral features' elements, texture features' elements do not have an order.

Table 3.2: Texture feature

Cluster No.	Cluster value	Weight(Pixel amount)
1	4.64,4.11,6.23,10.40 6.42,5.77,4.64,5.12	3756
2	7.10,3.52,3.55,3.02 3.63,3.17,4.27,3.64	8243
3	3.96,3.51,4.08,4.20 8.46,4.11,4.10,3.38	6128
4	2.63,2.39,2.75,2.50 2.82,2.65,2.93,2.63	23593
5	5.25,4.34,5.08,4.40 5.52,6.14,10.32,5.65	4033
6	3.15,3.83,3.41,4.04 3.97,8.12,4.14,3.91	6524
7	2.85,5.14,2.91,4.71 2.75,3.41,2.78,3.85	12302
8	3.86,4.03,7.94,4.01 3.82,3.15,3.99,3.40	6988
9	4.63,4.26,3.61,4.24 3.75,4.44,4.67,9.24	5138
10	7.78,9.88,6.55,5.13 5.16,5.31,4.81,5.60	3951

3.1.3 Varying Characteristics

Both spectral features and texture features can be presented as $(C_1, W_1), \dots, (C_n, W_n)$, where C_i and W_i are the multidimensional class centres and scalar weights information of the i^{th} cluster, respectively. These features have two varying characteristics that are defined in the following.

Definition : Two arbitrary C_i and C_j only denote the location in the feature space, the order of (C_i, W_i) and (C_j, W_j) is exchangeable. The exchange does not affect the feature, i.e. the spectral feature's elements have no order.

Different satellite images have arbitrary lengths. That is, the number of clusters

Figure 3.2: 2-dimensional vantage point [9]

for texture features is not fixed.

3.2 VP-Tree

The VP tree was first proposed by P. Yianilos and is based on the partitioning of relative distances between data points and a particular vantage point [36].

A simple 2-dimensional example is shown in Figure 3.2(a), where V is the vantage point and r is the median of the distance values which is used to partition all the other points into two subsets: S_1 and S_2 according to which distance to V is smaller than or equal to r , and larger than r , respectively. Consider the query for the nearest neighbour of a querying point q , with the requirement that the maximum distance between q and its nearest neighbour is smaller than a given distance, denoted as d . With this requirement, the distance between q and V , denoted as $\text{dist}(q, V)$, is classified into three cases, $\text{dist}(q, V) \leq r - d$, $\text{dist}(q, V) > r + d$ and $r - d < \text{dist}(q, V) \leq r + d$. The third case requires search both S_1 and S_2 . While for the first case, only S_1 and for the second

case only S_2 needs to be explored. Thus effectively pruning one half of the search space. This pruning of the retrieval space is based on the triangular inequality. Specifically, if $\text{dist}(q, V) \leq r - d$, for $p \in S_2$, the distance between p and q is lower-bounded by [9]

$$\begin{aligned}
 \text{dist}(p, q) &\geq ||\text{dist}(p, V)| - |\text{dist}(q, V)|| \\
 &\geq ||\text{dist}(p, V)| - (r - d)| \\
 &> |r - r + d| \\
 &= d
 \end{aligned}$$

Therefore S_2 can be ignored if $\text{dist}(q, V) \leq (r - d)$. Similarly, if $\text{dist}(q, V) > r + d$, for $p \in S_1$, the distance between p and q is lower-bounded by [9]

$$\begin{aligned}
 \text{dist}(p, q) &\geq ||\text{dist}(q, V)| - |\text{dist}(p, V)|| \\
 &\geq ||\text{dist}(q, V)| - r| \\
 &> |r + d - r| \\
 &= d
 \end{aligned}$$

Therefore S_1 can be pruned if $\text{dist}(q, V) > r + d$. If S_1 and S_2 are approximately the same in size, each pruning step effectively cuts the search space in half.

To each resulting partition, this partitioning strategy is recursively employed until the number of data points in a partition becomes small enough and the cost of a linear scan is acceptable.

Chiueh showed that the choice of vantage points and the value of d affect the performance of the VP-tree notably [9]. A suitable vantage point divides points in the set equally. This minimizes the number of data points in the searching path, and therefore leads to exploring less subtrees. However, the ideal vantage is not to be determined straightforwardly. Therefore, randomized algorithms are used to approximate the ideal vantage point. The algorithm has been proposed by Chiueh [9].

Step 1: Pick a set of candidate vantage points randomly from the data set.

Step 2: For each candidate vantage point

(1): Pick a set of sample objects randomly;

(2): Compute the distance from the candidate vantage to every sample object;

(3): Calculate the mean and the standard deviation values;

End for

Step 3: Choose the candidate vantage point with the maximum standard deviation

Another important parameter that affects the performance of a VP-tree is the choice of d , which represents the tradeoff between the likelihood of locating the nearest neighbours and the searching costs. If the value of d is small, the concentric regions become smaller, and therefore the probability of having to explore both subtrees is reduced. On the other hand, the choice of a smaller d always reduces the maximum allowable distance between a query point and its nearest neighbour, then in some cases suitable points may be abandoned. In order to find a suitable d , the system analyses the distance distribution before determining d . It is also useful to modify the d during the search process. More details can be found in [9].

When a query for the nearest neighbour of a giving point has to be satisfied, the distance between the query point and the vantage point associated with the root is calculated to determine which subset to search. This procedure is performed recursively until the nearest neighbour is found or no such nearest neighbour whose distance to the query point is smaller than the predefined d exists.

The above binary VP-tree construction can be generalized to a n -ary VP-tree structure. The data set S is split into n subsets corresponding to the distances between the selected vantage point and other data points. The subsets can be denoted as S_i , $i = 1$ to n . Each of the S_i 's has approximately the same number of data points. As for the binary VP-tree, r_i is used to indicate the boundary distance so that $r_{i-1} < \text{dist}(s, v) \leq r_i$ for all $s \in S_i$. With the same partitioning strategy, each S_i 's is recursively partitioned into n smaller subsets.

Update mechanisms, i.e. insertion and deletion algorithms, are available for the VP-Tree and were proposed by Fu [11]. Because in the case of the $(RS)^2I$ only inserting images is of interest, the algorithm is briefly introduced as follows:

Step 1: If any sibling leaf node of node L is not full, redistribute all data objects under P among the leaf nodes.

Step 2: Else if L has a parent node P , and P has room for one more child, split the leaf node L . Assume L is the k -th child of P . Retrieve all objects in L , and assume that S is the set of objects searched with the new object. Order the objects in S with respect to their distances from P 's vantage point v . Divide S into two subsets of equal cardinality, namely SS_1 and SS_2 . Shift the boundary distances and pointers of P so as to replace L with SS_1 and make room for the new leaf node SS_2 .

Step 3: Else, if a nearest ancestor A of L has room, let B be the immediate child of A , which is also an ancestor of L ,

(a) if any sibling nodes of B is not full, locate the nearest non-full sibling node C and redistribute the objects among the subtrees between B and C inclusively.

(b) else since A is not full, split the non-leaf child B .

Step 4: Else, either L is the root node or cannot find any ancestor node

of L which is not full, then split the root node into two new nodes S_1 and S_2 , and insert the new data object according to the strategy as discussed above for splitting either a leaf node or a non-leaf node. A new root node is created with S_1 and S_2 as the child nodes.

More details are available in Reference [11].

3.3 Earth Mover's Distance

In the previous sections, spectral features and texture features were briefly introduced. The $(RS)^2I$ extracts these two features and uses them to compare the similarities among satellite images. The comparison leads to the requirement of finding a way to compute the distance between two spectral features or two texture features. The traditionally employed Euclidean distance is not suitable, because both the spectral features and texture features have weights, and the lengths of the feature descriptors are not fixed as Section 3.1.3 introduced. In this case, a different solution for computing the distance is necessary. Hence the Earth mover's distance (EMD) [28] was employed as a distance measure function. It is defined as the following.

Consider two distributions:

$$x = \{(x_1, w_1), (x_2, w_2), \dots, (x_m, w_m)\}$$

and

$$y = \{(y_1, u_1), (y_2, u_2), \dots, (y_n, u_n)\},$$

where x_i and y_j are the cluster centers, w_i and u_j are the scalar weights. Define w^S as the total weight of x and u^S as the total weight of y . The EMD can be formally defined to find a flow $F = [f_{ij}]$ between x_i and y_j which minimizes the total cost. $D = [d_{ij}]$ is the ground distance matrix where d_{ij} is the ground distance between x_i and y_j . Thus the

Figure 3.3: Using EMD as similarity measurement

formula of the overall cost is defined as:

$$WORK(x, y, F) = \sum_{i=1}^m \sum_{j=1}^n d_{ij} f_{ij} \quad (3.3.1)$$

The following constraints have to be fulfilled simultaneously, i.e. Equation (3.3.2) ensures that the flow is always positive, while Equation (3.3.3) guarantees that not more resources from the source i can be obtained than are available. The same holds true for the sink j that cannot be overloaded according to Equation (3.3.4). The last constraint in Equation (3.3.5) specifies the maximal possible flow that is bounded by the minimal total energy of the participating vectors:

$$f_{ij} \geq 0, 1 \leq i \leq m, 1 \leq j \leq n \quad (3.3.2)$$

$$\sum_{j=1}^n f_{ij} \leq w_i, 1 \leq i \leq m \quad (3.3.3)$$

$$\sum_{i=1}^m f_{ij} \leq u_j, 1 \leq j \leq n \quad (3.3.4)$$

$$\sum_{i=1}^m \sum_{j=1}^n f_{ij} = \min \left(\sum_{i=1}^m w_i, \sum_{j=1}^n u_j \right) \quad (3.3.5)$$

Thus, the formula of the EMD can be defined as:

$$EMD(x, y) = \frac{\sum_{i=1}^m \sum_{j=1}^n d_{ij} f_{ij}}{\sum_{i=1}^m \sum_{j=1}^n f_{ij}} \quad (3.3.6)$$

It was shown that EMD can be employed to compute the distance between two spectral or texture features [19]. In this application, the scalar weight is used as the amount of work and the cluster centers of features are used as locations. Figure 3.3 shows an example for using EMD to compute the distances of two spectral or texture features.

3.4 De-clustering Techniques on Parallel Platforms

Highly parallel database systems are beginning to replace the traditional database management systems, especially for database systems with huge volume of data. When implementing a parallel image database, one important task is to distribute the images to different computing nodes effectively, this procedure is called de-clustering. Currently, three popular de-clustering strategies have been proposed [30].

3.4.1 Random De-clustering

Every single datum is assigned to a computing node with the same probability, hence a good balance is guaranteed. No information about the previous allocation is needed, so the space complexity is $O(1)$ and the time to decide to which node the datum has to be assigned is constant. Hence, the time complexity is $O(N)$ since it increases linearly with the size of the data, where N is the size of the data.

3.4.2 Round Robin De-clustering

The round robin (RR) approach guarantees high load balance through distribution of data objects to computing nodes in a circular order. Previous distribution information is not needed in the following operations, so the space and time complexity of RR are the same as for Random de-clustering, i.e. $O(1)$ and $O(N)$, respectively.

3.4.3 Proximity Index De-clustering

The proximity index (PI) technique was proposed by Schnitzer and Leutenegger [30]. In the beginning a set of data objects with the number of computing nodes are selected randomly, and each data object in the set is assigned to a computing node. The remaining data objects are assigned to computing nodes which are most dissimilar. For the PI de-clustering strategy, memory for the allocation of all earlier assigned data objects is needed, hence, the space complexity is $O(N)$. For each assigned data object all earlier allocations are examined, so the time complexity is $O(N^2)$.

Chapter 4

Comparative Study of R-Tree Family

4.1 Introduction

Currently, many different multi-dimensional indexing structures have been proposed in content based image retrieval systems. Among them, the structures of the R-tree family, especially the R-tree, R*-tree, SS-tree and SR-tree are the most referenced [22]. However, by now no comparison and analysis of the query performance, dynamic operations and file storage of these indexing strategies has been provided. Since the three performances are important references to evaluate a indexing structure, a query performances comparison among the R-tree family is included in this chapter. To the best knowledge of the author, this is the first performance comparison among the R-tree family members.

4.2 Data Sets and Experimental Platform

In order to give detailed comparisons, seven data sets were used with dimensions varying from 16 to 64. Each data set contained 10,000 distributed points forming clusters that themselves followed Gaussian distributions. These data sets were obtained by modifying the original datasets from the database archive of the University of California, Irvine [1]. In the comparison, 1,000 query points were employed to query each data set and the ten

closest points were retrieved for each retrieval. The query performance of the leaf nodes, the inner nodes, and the whole sets of nodes are shown for particular comparisons.

The comparison was done on a PC with a PIII 733 MHZ CPU, 512 MB memory running a Red Hat Linux 6.2 platform. The GiST [14] framework was employed. One useful function supported by GiST is that, the disk I/O is easily recorded, hence in the experiments the I/O block numbers instead of processing time was used as the metric to compare performances.

4.3 Query Performance Analysis

In order to differentiate details among the different tree structures, the comparison was performed for the inner nodes, leaf nodes and all nodes, separately. Figure 4.1, Figure 4.2 and Figure 4.3 show the query performance of inner nodes, leaf nodes and all nodes, respectively. The retrieval of inner nodes represents the query performance in inner nodes level, which is used with retrieval of leaf nodes to analyse the overlaps of a tree structure.

An R-tree is built based on partitioning all the points into smaller sub-spaces, each node in the tree stores between m and M entries, whereby m represents the minimum number of entries that can be stored into a single node and M represents the maximum number of entries. Therefore, the worst space utilisation in an R-tree is m/M . However, since an R-tree does not forbid overlaps of minimum bounding rectangles in different leaf nodes, there may be more than one path from the root to one data object. Therefore, an R-tree does not guarantee the most effective query performance. Figure 4.2 exhibits the comparison in terms of I/O accesses on the leaf nodes level and shows that the performance in leaf nodes of the R-tree is worse than the R*-tree and the SR-tree. The unsatisfactory performance on the leaf node level also affects the overall query

Figure 4.1: Querying performance of inner nodes

Figure 4.2: Querying performance of leaf nodes

Figure 4.3: Querying performance of all nodes

performance on all nodes level, which is displayed in Figure 4.3.

The R*-tree is a variant of the R-tree and employs more complex inserting and splitting algorithms to minimise the overlaps. As a result, there is strictly one path from the root to each data object, which leads to a better query performance in the leaf node level. Figure 4.2 shows this advantage. While the R*-tree shares a similar structure with the R-tree, the advantage is not obvious in where the query performance is measured over all nodes like highlighted in Figure 4.1 and Figure 4.3.

A SS-tree has a similar structure in comparison to an R-tree, but employs minimum bounding spheres instead of minimum bounding rectangles to contain data objects. This difference results in better data clustering which means a SS-tree has less inner nodes. This is depicted in Figure 4.1, where the SS-tree has the best inner node query performance. Though a SS-tree leads to better clustering, the utilisation of minimum bounding spheres also results in larger volumes and leads to larger overlaps among leaf

nodes. The large overlap deteriorates the query performance of the leaf node level and hence affects the query performance. Figure 4.2 and Figure 4.3 show that the SS-tree has the worst performance both in the leaf nodes level as well as in the overall measurement. The SR-tree is the latest structure in the R-tree family that employs both minimum bounding rectangles and minimum bounding spheres to partition data objects. Using this combination, the SR-tree inherits the benefits and overcomes the disadvantages of the SS-tree, R-tree and R*-tree. Figure 4.3 highlights that the SR-tree has the best query performance among the member of the R-tree family.

4.4 Insertion Costs

All mentioned structures support dynamic operation, such as insertion and deletion. Among the dynamic operations, insertion is most crucial since the content of the image database is rather growing than shrinking. Hence, a comparison of the insertion costs is necessary. In the experiment, 1,000 distributed points were inserted into an existing R-tree, R*-tree, SS-tree and SR-tree, respectively. Each existing tree contains 10,000 data objects. The results are shown in Figure 4.4. Clearly the SS-tree has the best insertion performance and the R*-tree has the worst dynamic property while the R-tree and SR-tree perform in the middle range. The SS-tree has the best inserting performance due to the simplified insertion algorithm [22], while the R*-tree employs a complex insertion algorithm to achieve better query performance and thus has the worst inserting capability. The insertion performance of the SR-tree is between the R*-tree and the SS-tree since it is a combination of the two structures.

Because $(RS)^2I$ seldom deletes satellite images from the database, deletion algorithms were not compared.

Figure 4.4: Comparison of inserting costs

4.5 File Storage

The storage requirement is another important attribute to evaluate a structure whereby the same seven data sets were used to measure the R-tree, R*-tree, SS-tree and SR-tree. The results are shown in Figure 4.5, whereby the R-tree always needs the least amount of storage while the SR-tree always has the largest requirement. The performance of the R*-tree and SS-tree are in-between.

4.6 Summary

There are six conclusions about the R-Tree family made based on the above experiments:

- 1: Figure 4.1, Figure 4.2 and Figure 4.3 show that R*-trees improve the query performance upon R-trees, but the improvement is not tremendous. The improvement comes from the modified node splitting algorithm and insertion algorithm.

Figure 4.5: Comparison of file storage

- 2: Figure 4.1 also depicts that SS-trees have better data clustering properly and less inner nodes, but also produce larger overlap, which was indicated by Figure 4.2 and Figure 4.3.
- 3: Figure 4.2 and Figure 4.3 display that SR-trees have the best query performance among the R-tree family.
- 4: Figure 4.4 depicts that the SS-tree has the best dynamic property.
- 5: Figure 4.5 shows that the R-tree needs the least amount of storage.
- 6: Figure 4.3 and Figure 4.5 highlights that the R*-tree has the best performance if both the storage and query performance are evaluated.

Chapter 5

VP-EMD Tree

5.1 Introduction

Most of the existing indexing techniques for CBIR employ partitioning methods to divide the feature spaces. Hence, they are only suitable for fixed lengths features since boundaries are key parameters that require absolute coordinates. Other indexing strategies utilize distances among data objects and also require fixed lengths and partial orders. Therefore, current multi-dimensional indexing structures are limited for fixed features and are not suitable for data with varying characteristic. For example, the SR-Tree needs fixed feature lengths and fixed physical locations of features to partition data objects into different rectangles or spheres. However, the requirement for fixed features is strict and limits the feature extraction algorithms. On the other hand a dimension reduction is possible as post-processing step. Examples are given by the principal component analysis (PCA) or the utilisation of vector quantisation (VQ) [38]. The results are acceptable, but the performance suffers tremendously for CBIR system with a high insertion rate since a permanent adaptation is required, i.e. either by constantly repeating the PCA or optimising the codebook for the VQ [34]. Therefore, it is preferable to use the given features directly.

This chapter discusses the problems and solution based on the features of the $(RS)^2I$. The spectral features and texture features introduced in Chapter 3 have varying lengths, a partially unfixed order with respect to the vector elements and additional weight information. Current indexing techniques are not suitable for the considered features. This conflict requires a new structure, which is suitable for data with a higher degree of complexity and thus more flexible for the actual retrieval.

5.2 VP-EMD Tree

As introduced in Chapter 3.1, spectral features and texture features have varying lengths and varying spatial order. They also have additional weight information. Given the texture feature as the example, the definition of the feature space is:

Definition: Two texture features were extracted from two given satellite images. Feature one was represented as $(C_1, W_1), \dots, (C_n, W_n)$, feature two was represented as $(S_1, T_1), \dots, (S_m, T_m)$, n does not equals m strictly. In other words, the length of texture feature is not fixed.

Definition: One texture feature is extracted from one given satellite image and represented as $(C_1, W_1), \dots, (C_a, W_a), \dots, (C_b, W_b), \dots, (C_n, W_n)$ ($1 \leq a < b \leq n$). The order of the a -th and b -th tuple, i.e. cluster, is exchangeable. In other words, $(C_1, W_1), \dots, (C_a, W_a), \dots, (C_b, W_b), \dots, (C_n, W_n)$ and $(C_1, W_1), \dots, (C_b, W_b), \dots, (C_a, W_a), \dots, (C_n, W_n)$ represent the same satellite image in the feature space.

In order to handle this type of data, the VP-EMD tree [34], a member of the vantage point (VP) tree family is proposed here to solve the indexing.

With the idea of the VP tree to focus on the distances among features and the EMD as an effective solution to compute the actual distances [19], it is natural to combine these two concepts to the VP-EMD tree.

5.3 Experimental Results

The VP-EMD tree was implemented on a Sun Blade1000 workstation with a Sun Ultra-sparc III 900M CPU, 1GB memory running a SunOS 5.8 platform. Spectral features and texture features were used to evaluate the query performance, wherein the number of feature points varied from 2,000 to 10,000. To the author's knowledge, there is no other indexing structure that is suitable to index the features with varying characteristics and additional weight information, so Bubble Sort with a time complexity of $O(N^2)$ and Quick Sort with a time complexity of $O(N \log(N))$ were employed to provide a foundation for the comparisons.

Figure 5.1 and Figure 5.2 show the comparison of indexing speed for texture features and spectral features. For each data set, the ten most similar features were queried. The horizontal axis represents the number of points. The vertical axis provides the indexing speed. For example, in Figure 5.1, the VP-EMD tree gets a two second performance for 8,000 points. In other words, for a database contained 8,000 images, the VP-EMD tree needed two seconds to process the index requirement.

The comparison shows that the VP-EMD tree has a much better query performance and depicts that the VP-EMD tree accelerates the querying performance efficiently and reduces the time complexity to $O(N)$. However, an advantage of the sorting approach is that a complete similarity list with respect to the query point is provided while the VP-EMD tree returns only a small subset. Nevertheless this is sufficient for most practical cases and especially for databases with a large amount of data.

The precision of the VP-EMD tree is another important parameter to evaluate the performance. The experimental results for both spectral and texture features showed that the precision of the VP-EMD tree is always 100%. The high precision guarantees that the query results are correct, i.e. always the actual nearest neighbours are retrieved.

Figure 5.1: Comparison of query time for texture teatures

Figure 5.2: Comparison of qury time for spectral features

The building cost of a VP-EMD tree is a notable parameter as well. Table 5.1 shows the building costs for texture and spectral features. For every column in the table, the texture data set and the spectral data set contain the same number of points. But the cluster number of texture features is always ten and each element always contains eight entities, while the number of spectral features varies from three to eight and each element always contains three entities [19], i.e. the texture feature is 90-dimensional while the spectral feature varies from 12-dimensional to 32-dimensional. Hence, texture features always have more data than the spectral features and the texture features always need more building time. For texture features, the building cost varies from 152 seconds to 639 seconds according to the number of points varying from 2,000 to 10,000. The corresponding building cost for spectral features is in the range of 84 seconds to 384 seconds.

Table 5.1: Building costs of a VP-EMD tree

Points	2,000	4,000	6,000	8,000	10,000
Texture Feature	152	273	399	518	639
Spectral Feature	84	169	239	308	384

Chapter 6

Approximative Indexing

In the VP-EMD indexing technique, the query points are divided into three partitions. While for two partitions approximately half of the data set can be pruned for each indexed access, the third subset requires a complete search, which has a negative impact on the query speed. Hence, a new approximative indexing approach is proposed. This chapter introduces the trade-off strategy for applications where the emphasis is on similarity rather than completeness for large databases. The algorithm divides the crucial subset into deeper subsets, and retrieves only parts of them. Finally, experiments show the effective and efficient behaviour in an actual image database.

6.1 Indexing Structure

As introduced in Chapter 5, the VP-EMD tree was proposed to retrieve data with varying characteristics. An example for the usage of a vantage point in the VP-EMD tree is shown in Figure 6.1, whereby the point V is the vantage point selected from the data set by using the algorithm specified by Chiueh [9].

For the inner subset, the triangular inequality guarantees that retrieval of S_1 is enough to find the nearest neighbour [9]. Similarly, for the outer subset, retrieval of S_2 is enough, and hence, approximately half of the points can be discarded. Note that

Figure 6.1: Middle subsets of the VP-EMD tree

the prune procedure can be employed recursively, i.e. with a recursively defined vantage point within the subspace, with a resulting searching complexity of $O(\log_2 N)$ for three partitions [31]. Only for queries that lead to a search in the middle subset, the complete search of both S_1 and S_2 is needed.

6.2 Middle Subset Trade-off

The middle subset trade-off divides the middle subset in the VP-EMD tree into M subsets according to the EMD distances from the contained points to the vantage point V . The user-defined parameter M determines how many subsets of the middle subset are created and which subsets from S_1 and S_2 are retrieved during an access. A simple example for the middle subset trade-off is shown in Figure 6.1, where T_1 , T_2 , T_3 , and T_4 are subsets of the middle subset. If the EMD distance between the query point and vantage point is located close to the inner subset, such as T_1 , and a suitable trade-off value was set, the access can be limited to only S_1 and parts of the middle subset, such as T_3 , to find the nearest data point. Similarly, if the query point is located in the middle subset and close to the outer subset, only S_2 and subsets close to the outer subset are retrieved. Obviously, a trade-off with higher value leads to the access of more points and therefore

achieves a higher accuracy while at the same time increasing the retrieval time.

Definition : In the field of image retrieval, the trade-off represents the balance between retrieval speed and retrieval accuracy. In particular for the middle subset trade-off algorithm, the trade-off represents the number of subsets been searched. A high trade-off value results in high retrieval accuracy and leads to a lower retrieval speed. On the contrary, a low trade-off value represents for a low retrieval accuracy while gaining a higher retrieval speed.

Given Figure 6.1 as an example, the middle subset trade-off algorithm was stated by pseudo code as follows:

Step 1: From the root vantage point V , compute the distance between the query point Q to V .

Step 2: If the distance is less than $r - d$, prune subset S_2 and only search points in subset S_1 . If the distance is larger than $r + d$, prune subset S_1 and only search points in subset S_2 . Repeat from Step 1 until reaching a leaf node.

Step 3: If the distance is between $r - d$ and r , divide subset S_2 into $M/2$ subsets according to the distances to V . Regarding the value of the trade-off set by user, search subset S_1 and parts of subsets. Repeat from Step 1 until reaching a leaf node.

Step 4: If the distance is between r and $r + d$, divide subset S_1 into $M/2$ subsets according to the distances to V . Regarding the value of the trade-off set by user, search subset S_2 and parts of subsets. Repeat from Step 1 until reaching a leaf node.

6.3 Experimental Results

This algorithm was implemented on a Sun Blade 1000 workstation with a Sun Ultrasparc III 900 MHz CPU, 1 GB memory running a SunOS 5.8 platform. In order to provide a

Figure 6.2: Retrieval results without trade-off strategy

Figure 6.3: Retrieval results with middle trade-off strategy

Figure 6.4: Comparison of trade-off impact on query time for texture features

Figure 6.5: Accuracy with ten nearest references for texture features

complete evaluation of the trade-off strategy, common digital images and remotely sensed satellite images were employed.

The colour histogram, spectral features and texture features were used to evaluate the proposed trade-off algorithm, whereby the number of features points varied from 10,000 to 50,000. The modified VP-EMD tree using the trade-off was compared with the original VP-EMD structure in terms of query speed and retrieval accuracy.

In the experiments, each middle subset was divided into four deeper subsets unless the number of points included in the subset reaches the minimum requirement. Figure 6.2 shows the retrieval of 10,000 data points using the texture features without the trade-off strategy. In comparison Figure 6.3 shows the result from the data set with a middle trade-off, whereby the first five images in Figure 6.3 can be found in Figure 6.2. Hence, the query precision of the middle trade-off is 50% with respect to the ten closest retrievals. However, considering this result without the appropriate context, distorts the perception, i.e. the five incorrectly retrieved image have nevertheless a high degree of

Figure 6.6: Accuracy with 50 nearest references for texture features

similarity. Actually, these images are ranked in the range of the ten to 30 most similar images. This is in agreement with the initial assumption that always sufficient images are available, that are judged suitable by the user. Figure 6.4 shows the corresponding query time comparison for the texture features. The curve named high trade-off represents the retrieval of three subsets, the curve named middle trade-off using two subsets, and the curve named low trade-off represents the usage of only one subset. The curve named VP-EMD represents the original structure which does not employ any trade-off. Focusing on the precision Figure 6.5 shows the query accuracy comparison for texture features with the ten nearest points as a reference. In average, nine of the ten images that the high trade-off strategy found are among the real ten most similar images, five of the ten images that the middle trade-off strategy found are among the real ten most similar images, and three images that the low trade-off strategy found are among the the real ten most similar images. In comparison Figure 6.6 emphasizes the same issue with the 50 nearest points as the reference to provide a better understanding of the results from the trade-off strategy. In average, all the ten images that the high and middle trade-off

Figure 6.7: Comparison of trade-off impact on query time for spectral features

strategies found are among the real 50 most similar images, and seven images that the low trade-off strategy found are among the real 50 most similar images.

The same experiment was repeated for the spcetral features, i.e. Figure 6.7 - Figure 6.9 are comparisons for the spectral features similarly, while Figure 6.10 - Figure 6.12 are based on the colour histograms, respectively.

Figure 6.5, Figure 6.8 and Figure 6.11 employ the ten closest points from the original VP-EMD tree as the reference and show that using more subsets leads to a higher accuracy. In comparison Figure 6.6, Figure 6.9 and Figure 6.12 employ the 50 closest points as the reference to give a better understanding of the retrieval accuracy.

In summary, the middle subset trade-off strategy is a good balance between the retrieval time and the retrieval accuracy. In average, for the indexing of the texture features, the high trade-off saves 7% in terms of processing time while maintaining 98% of the retrieval accuracy. The middle trade-off reduces the search time by 12% while providing 46% accuracy and the low trade-off improves the search time by 16% with 28% accuracy. In the spectral feature indexing, the high trade-off reduces the retrieval time

Figure 6.8: Accuracy with ten nearest references for spectral features

Figure 6.9: Accuracy with 50 nearest references for spectral features

Figure 6.10: Comparison of trade-off impact on query time for colour histogram

Figure 6.11: Accuracy with ten nearest references for colour histogram

Figure 6.12: Accuracy with 50 nearest references for colour histogram

by 5% with 76% retrieval accuracy, while the middle trade-off reduces the search time by 13% with 46% accuracy. The low trade-off saves approximately 22% of the otherwise needed retrieval time with 24% accuracy. Similarly, for the colour histogram, the high trade-off saves 6% search time with 76% retrieval accuracy, the middle trade-off saves 12% of the search time with 56% accuracy, and the low trade-off saves 16% search time with 36% indexing accuracy.

Chapter 7

Data Distribution

One important attribute of CBIR systems is that generally the volume of image data is huge. For example, NASA predicted a growth of 1 TB of image data per day and satellite for the near future [35]. If the amount of digitally acquired images increases fast, a centralised image database lacks in efficiency and distribution is almost the only possible solution to handle the huge data amount. In this respect current research work on distributed image database systems showed that approaches with a high degree of parallelism are beginning to displace traditional mainframe computers [2].

In this chapter a suitable de-clustering strategy is proposed for distribution of the given data indexed by a VP-EMD structure. A validation based on actual measurements is also conducted. For simplicity it is assumed that all entities in the distributed architecture are identical. Hence the terminology parallel is preferred hereafter.

7.1 Parallelisation of Indexing Strategy

The VP-EMD tree introduced in Chapter 5 was selected as the underlying indexing structure. The master-slave concept was chosen for the parallelisation, i.e. one dedicated computer is selected as the master machine and stores all internal nodes of the parallel VP-EMD tree. The leaves are distributed over the computing slave nodes. An example

Figure 7.1: Parallel VP-EMD tree for three computing nodes

for a parallel VP-EMD tree that uses three computing nodes is shown in Figure 7.1.

7.2 New De-clustering Method

Definition : De-clustering is the process to distribute data to distributed computing nodes.

One important aspect which is neglected in most research work on de-clustering strategies is the allocation cost [35]. Usually it is not included in the comparison of different approaches. However, in some applications, especially for image databascs with a frequent update characteristic, the allocation time is an important attribute. In the context of CBIR and with the conclusion that similarity described by any kind of metric is a very subjective matter, a less accurate retrieval can be acceptable if this benefits the de-clustering speed. The idea of the de-clustering strategy proposed here is the extension of the VP tree concept itself, which uses a vantage point to partition other points according to their distances to this vantage point. In the VP de-clustering strategy, a vantage point is selected by the same algorithm that was used for the actual tree. For example, if the points in Figure 7.2 are de-clustered and distributed to three computing nodes, then according to the proximity index (PI) [30] algorithm the combinations (2,7,9), (1,4,6)

Figure 7.2: Planar schematic of the first VP-EMD tree level

and (2,5,8) are assigned to different computing nodes. The VP algorithm distributes the information over the computing nodes as (1,4,7), (2,5,8) and (3,6,9).

Given n computing nodes and a data set S containing x data points to be distributed, and use S_i , $i = 1$ to n to denote the subsets, the VP de-clustering strategy was stated as:

Step 1: Find a vantage point V according to the strategy proposed by Chiueh [9].

Step 2: According to the strategy proposed by Chiueh [9], choose r_i , $i = 1$ to n , to indicate the distance so that $r_{i-1} < \text{dist}(s, V) \leq r_i$ for all $s \in S_i$.

Step 3: For a point p whose distance to the vantage point V , donated as $\text{dist}(p, V)$, if $r_{i-1} < \text{dist}(p, V) \leq r_i$, distribute point p to subset S_i .

Step 4: Repeat Step 3 until all points in S are distributed.

Step 5: Distribute the points in the same subset to different computing nodes. Repeat Step 5 until all points are distributed to computing nodes.

7.3 Retrieval Algorithm

Most current parallel image databases retrieve all de-clustered data to find the most accurate result [26], which is suitable if a high degree of precision is needed. However, in databases with a huge image archive, often the retrieval result comprises more images than can be handled by the user. In this case, a less accurate retrieval might also be acceptable as long as sufficient similar images are provided. This relaxation can be exploited in the form that in a parallel architecture, the retrieval procedure can be limited to finding only the most similar images in each computing node, i.e. a local optimum, and then composes a candidate set on the master level. Followed by a global ranking, a retrieval list is produced that provides a most likely sub-optimal solution. However, with the proposed de-clustering approach in the previous section an adequate result can be achieved since neighbouring feature vectors are stored on the same node.

The above retrieval algorithm is also suitable in two modified versions for the k -NN search. In the first approach, the k most similar images from each computing node are retrieved and ranked on the master level to produce the actual list of the k most similar images from the stored data set. Note that this solution guarantees to find the optimal solution. The second approach applies a relaxation and finds the k/n most similar images from each of the n computing nodes. The result is sub-optimal but has the advantage of an improved retrieval time. In particular in systems with a high workload and a high degree of parallelisation such a sub-optimal approach is beneficial for the first few iterations of a query process during which only an approximative retrieval is required.

Given n computing nodes and one query image q to find the k most similar image, the sub-optimal retrieval algorithm was stated as:

Step 1: The master node M passes the query image to all slave nodes.

Step 2: For $i = 1$ to k

in node i find the k/n most similar images to query image q .

send the k/n images from node i to master M .

Step 3: Master node M composes a candidate with the retrieval results from each computing nodes.

7.4 Experiments

The assessment of the VP de-clustering strategy and parallelisation was conducted on a distributed memory cluster which comprised ten computers with each hosting two Intel Xeon 2.6 GHz processors with dedicated 1 GB RAM per processor. Hence, twenty processing nodes were available. For the interconnection among the computers Myrinet was used. An additional computer in the same configuration acted as master.

Different de-clustering approaches were investigated with a varying number of compute nodes and two different feature types for the similarity description, whereby the assessment was based on processing time and retrieval accuracy. The retrieval accuracy was calculated by comparing the retrieval images with reference images. The test data consist of satellite imagery. However, the proposed approach is not specific to satellite data and can be applied to other image types as well as given that feature extraction mechanisms are available.

7.4.1 Comparison of De-clustering Speeds

Based on the comparisons of de-clustering strategies by Schnitzer and Leutenegger [30], the proximity index (PI) approach is the most effective strategy in terms of accuracy, while the round robin (RR) approach is a very fast de-clustering strategy. Hence, both are used for the comparison with the proposed VP-based de-clustering strategy. Figure 7.3 and Figure 7.4 show the comparison for de-clustering speeds using texture features,

Figure 7.3: De-clustering time for texture features using ten computing nodes

which have up to 90 dimensions and follow the characteristic described in Chapter 3. The number of computing nodes and indexed images were varied. The results show that the RR technique always needs the least amount of de-clustering time, beating the VP approach by a small margin. However, the PI strategy is clearly outperformed. Similarly, Figure 7.5 and Figure 7.6 show the same investigation for spectral features with a fixed length of 32 elements for each feature vector. One of the differences is the faster indexing access due to the reduced computational burden in the similarity measurements which directly depend on the dimensionality. However, the general relation among the three candidates is unaffected. Finally, the PI strategy does not provide good scalability with respect to the number of nodes compared with the other two approaches. The VP technique is clearly better and loses only marginally against the RR strategy.

The time complexity of the PI algorithm is $O(N^2)$ [30], while the VP strategy is characterised by an $O(N)$ time complexity [35].

Figure 7.4: De-clustering time for texture features using 20 computing nodes

Figure 7.5: De-clustering time for spectral features using ten computing nodes

Figure 7.6: De-clustering time for spectral features using 20 computing nodes

7.4.2 Comparison of Accuracy Performances

In a second investigation the retrieval accuracy was assessed by retrieving the ten most similar images using the three de-clustering strategies, i.e. RR, VP and PI. The obtained retrieval results were then compared against the actual 10-NN images. Figure 7.7 - Figure 7.10 depict the results for texture and spectral features, respectively, again using two different configurations in terms of processing nodes. On average for the texture features, the RR algorithm has an accuracy of 22% for ten computing nodes, while the VP approach achieves for the same setting an accuracy of 60%. Thus, it even beats the PI technique with its 56% accuracy. Similarly on average for the spectral features, the RR strategy has an accuracy of 26%, the VP approach has an accuracy of 52% and the accuracy of the PI strategy is 54%.

Figure 7.7: Accuracy performance for texture features using ten computing nodes

Figure 7.8: Accuracy performance for texture features using 20 computing nodes

Figure 7.9: Accuracy performance for spectral features using ten computing nodes

Figure 7.10: Accuracy performance for spectral features using 20 computing nodes

7.4.3 Retrieval Results

The trade-off between retrieval accuracy and indexing performance pays off advantageously for content-based image databases with a substantial image archive. To provide a perceptual insight, the retrieval results for the three de-clustering approaches are shown in Figure 7.11. For the experiment 10,000 satellite images were de-clustered over twenty computing nodes. In the retrieval process the most similar images were selected from each computing node and taken by the master for generating the finally presented retrieval result. Figure 7.11(a) shows the retrieval based on the RR approach, whereby the first three images are generally among the actual top-ten images compared to the retrieval on a non-parallelised version. Similarly, Figure 7.11(b) is the result of the VP de-clustering strategy. In this case the first six images are scored from the top-ten similar images. In comparison the individual ranking of the PI de-clustering strategy presented in Figure 7.11(c) looks slightly different. The same accuracy was achieved like for Figure 7.11(b), but not all relevant urban areas are ranked first.

Figure 7.11: Retrieval results (top-ten): (a) RR de-clustering strategy, (b) VP de-clustering strategy, (c) PI de-clustering strategy

Chapter 8

Conclusions

The four major contributions in this thesis are: comparisons and analysis of current multi-dimensional indexing techniques, development of a new multi-dimensional indexing structure for features with varying vectors / lengths and additional weight information, proposal of an effective approximative indexing strategy, and an effective de-clustering algorithm based on a cluster architecture.

Detailed measurements of the R-tree, R*-tree, SS-tree and SR-tree performance were presented. Comparisons of query performance, inserting cost and file storage were utilised to provide an analysis. It was concluded that the SR-tree has the best query performance, the SS-tree has the best dynamic properties and the R-tree needs the least amount of storage, while the R*-tree has the best performance if both the storage and query performance are considered. Hence, different CBIR systems, which have different emphasises, need to evaluate their main purpose and select a suitable structure. However, with respect to the decreasing costs of hardware but the increasing number of users in web-accessible image databases, the focus is on the query performance rather than on storage requirements.

The VP-EMD tree was introduced as an effective multi-dimensional indexing structure for spectral features and texture features with varying lengths, varying par-

tial order and additional weight information. The VP-EMD tree combines the indexing structure of the VP-tree with the EMD as a measure for similarity. It supports k -nearest neighbour search and explores a new way for multi-dimensional indexing techniques by focusing on the similarity between data objects, while most current structures use hyper-rectangles or hyper-spheres to partition the feature spaces and focus on the partitioning. It was found that the utilisation of the EMD is an effective approach to compute distances in terms of similarity. The experimental results show that the VP-EMD tree has a good query performance, high precision and acceptable building costs. Moreover, the VP-EMD tree solves the indexing problem of multi-dimensional data with varying characteristic. However, the experimental results also showed a significant dependency between the building costs and the data size, which is mainly due to the selection of appropriate vantage points.

To address the increased processing costs, a trade-off strategy was presented. It balances the retrieval speed and accuracy by dividing query subsets and filtering unnecessary subsets. Hence, only promising parts of the subset and their closest neighbouring subsets are visited. Experiments showed that the trade-off is an effective and efficient strategy to improve the retrieval speed and to produce satisfactory image retrieval results. The choice of an actual trade-off value can be controlled by the image database designers and the users.

The so-called vantage point de-clustering strategy was proposed to de-cluster feature vectors extracted from images over a cluster architecture. The suggested approach maps the similarity concept from the indexing level to the architectural level and showed through empirical results that for a slight drop in retrieval precision an increased indexing speed can be achieved. Measurements were obtained in relation to the traditionally used PI de-clustering strategy and clearly showed that the achieved retrieval accuracy is equivalent while the complexity of the de-clustering was reduced from $O(N^2)$ to $O(N)$.

Based on the above discussion, the objectives in this thesis: (1) comparison and analysis of current multi-dimensional indexing techniques, (2) development of an effective and effective indexing structure for variable lengths features employed in the $(RS)^2I$ project and (3) implementing a parallelised version were achieved. Besides the $(RS)^2I$ project, the analysis of the R-Tree family is suitable for other CBIR systems, the VP-EMD tree can be used to index other features with variable lengths or unfixed spatial order. The VP de-clustering strategy is applicable to distribute all features and parallelise a CBIR system.

Three publications resulted from above research work and are listed at the end of this thesis.

Chapter 9

Possibilities for Future Directions

The previous chapters analysed multi-dimensional indexing structures to speed up the query performance in content based image retrieval. As an extension this chapter discusses possible future research directions.

The $(RS)^2I$ project aims to supply the access of satellite images to non-professional users. Users can query images with both image annotations and image content, and administrators can insert images into the database or modify the corresponding data. For the huge amount of satellite data, indexing techniques are necessary to accelerate the query performance. Currently, the proposed indexing engine is implemented on a single cluster, however, this is not sufficient for an image database with a vast number of users and the expected growth of image data in the future. Hence, it is necessary to design a suitable platform to achieve flexible access. Grid computing was found to be one alternative, since it is in line with the aim to span across a large number of geographically distributed participants [15]. A content-based image database can be implemented on a Grid platform [10]. Figure 9.1 shows the structure of such a system as an example.

The storage of a content-based image database which contains images, annotations, and features can be implemented in the fabric layer of a Grid system. In other words, this layer contains the physical data. In this layer, distributed computers or clus-

Figure 9.1: Image database based on a Grid environment

ters are used to store huge amounts of images and related information. In such cases, the resource implementation involves internal protocols. Fabric components also implement the local, resource-specific operations which occur on specific image sources as a result of sharing operations at higher levels. Mechanisms are required for managing or transferring data.

The VP-EMD tree indexing strategy and VP de-clustering strategy can be implemented in the middleware as collective services as shown in Figure 9.1. With the VP de-clustering strategy, the database system can distribute images to different clusters effectively. This is also a major difference compared to current Grid image database systems since the current systems lack the capability to allocate features with varying characteristics.

An embedded VP-EMD tree is the second unique feature. The multi-dimensional indexing engine is the emphasis of such a content-based image database on a Grid platform. With the VP-EMD tree, the indexing engine can process queries more effectively.

CHAPTER 9.

Research on the indexing engine for a Grid platform definitely would help building content-based image databases of large sizes.

Bibliography

- [1] <http://kdd.ics.uci.edu/databases/CorclFeatures/CorelFeatures.data.html>.
- [2] M. Abdelguerfi and K. F. Wong. *Parallel Database Techniques*. Wiley-IEEE Computer Society Press. Hoboken, NJ, U.S.A, First edition, 1998.
- [3] L. Balmelli, J. Kovaevi, and M. Vetterli. Quadtree for embedded surface visualization: Constraints and efficient data structures. *Proceedings of the IEEE International Conference on Image Processing*, 2:487–491, 1999.
- [4] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. *Proceedings of the ACM SIGMOD Conference*, pages 322–331, 1990.
- [5] A. D. Bimbo. *Visual Information Retrieval*. Morgan Kaufmann Publishers. San Francisco, CA, U.S.A, 1999.
- [6] V. Castelli and L. D. Bergman. *Image Databases, Search and Retrieval of Digital Imagery*. John Wiley. Indianapolis, IN, U.S.A, 2002.
- [7] C. Chang, B. Moon, A. Acharya, C. Shock, A. Sussman, and J. Saltz. Titan: A high-performance remote-sensing database. *Proceedings of the International Conference on Data Engineering*, pages 375–384, 1997.

BIBLIOGRAPHY

- [8] C. H. Cheung, L. M. Po, and K. M. Wong. Web-based beowulf-class parallel computing on image database indexing and retrieval system. *Proceedings of the International Symposium on Intelligent Multimedia, Video and Speech Processing*, pages 457–460, 2001.
- [9] T. Chiueh. Content-based image indexing. *Proceedings of the International Conference on Very Large Data Bases*, pages 582–593, 1994.
- [10] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal on Supercomputer Applications*, 3:3–15, 2001.
- [11] A. W. Fu, P. M. S. Ch, Y. Cheung, and Y. S. Moon. Dynamic VP-tree indexing for n-nearest neighbour search given pair-wise distances. *VLDB Journal*, 9(2):154–173, 2000.
- [12] Y. Gong. Intelligent Image Databases. pages 12–15, 1998.
- [13] A. Guttman. R-Trees: A dynamic index structure for spatial searching. *Proceedings of the ACM SIGMOD*, pages 47–57, 1984.
- [14] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer. Generalized search trees for database systems. *Proceedings of the International Conference on Very Large Databases*, pages 562–573, 1995.
- [15] B. Jacob, L. Ferreira, N. Bieberstein, C. Gilzean, J. Y. Girard, R. Strachowski, and S. Yu. *Enabling applications for Grid computing with Globus*. International Technical Support Organization, First edition, 2003.
- [16] A. Jaimes and S. F. Chang. A conceptual framework for indexing visual information at multiple levels. *Proceedings of the SPIE Internet Imaging*, 3964:2–15, 2000.

BIBLIOGRAPHY

- [17] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 369–380, 1997.
- [18] J. A. Lay and L. Guan. Image retrieval based on energy histograms of the low frequency DCT coefficients. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processings*, 6:3009–3012, 1996.
- [19] Y. Li and T. Bretschneider. Supervised content-based satellite image retrieval using piecewise defined signature similarities. *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium*, 2:734–736, 2003.
- [20] B. Liu and T. Bretschneider. D-ISMC: A distributed unsupervised classification algorithm for optical satellite imagery. *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium*, 6:3413–3415, 2003.
- [21] G. B. Marchisio, W. H. Li, M. Sannella, and R. Goldschneider. GeoBrowse: An integrated environment for satellite image retrieval and mining. *Proceedings of the IEEE Geoscience and Remote Sensing Symposium*, 2:669–673, 1998.
- [22] N. G. Nascimento. Benchmarking access structures for the similarity retrieval of high-dimensional multimedia data. *Proceedings of the IEEE International Conference on Multimedia and Expo*, 2:1215–1218, 2000.
- [23] P. O. Neil. *Database: Principles, Programming, and Performance*. Morgan Kaufmann Publishers, Inc, Second edition, 2001.
- [24] J. A. Orenstein. A class of data structures for associative searching. *Proceedings of the International Conference on Management of Data and Symposium on Principles of Database Systems*, pages 181–190, 1984.

BIBLIOGRAPHY

- [25] S. Pramanik and J. Li. Clustered SS-tree. *Proceedings of the IEEE International Conference on Multimedia and Expo*, pages 1195–1198, 2001.
- [26] W. Premchaiswadi, N. Premchaiswadi, T. Patnasirivakin, S. Chimlek, and S. Narita. Image indexing technique and its parallel retrieval on PVM . *Proceedings of the International Conference on Digital Image Computing - Techniques and Applications*, pages 289–298, 2003.
- [27] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. *Proceedings of the ACM SIGMOD*, 2:71–79, 1995.
- [28] Y. Rubner, C. Tomasi, and L. J. Guibas. The Earth mover’s distance as a metric for image retrieval. *International Journal of Computer Vision*, pages 99–121, 2000.
- [29] D. Pfoer S. Brakatsoulas and D. Pfoer. Revisiting R-tree construction principles. *Proceedings of the East European Conference on Advances in Databases and Information Systems, Lecture Notes in Computer Science*, 2435:149 – 162, 2002.
- [30] B. Schnitzer and T. Leutenegger S. Master-client R-trees: A new parallel R-tree architecture. *Proceedings of the International Conference on Scientific and Statistical Database Management*, pages 68–77, 1999.
- [31] H. Shao, T. Svoboda, V. Ferrari, T. Tuytelaars, and L.V. Gool. Fast indexing for image retrieval based on local appearance with re-ranking. *Proceedings of the IEEE International Conference on Image Processing*, pages 737–740, 2003.
- [32] R. Weber and H. J. Schek. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. *Proceedings of the International Conference on Very Large Data Bases*, pages 194–205, 1998.
- [33] D. A. White and R. Jain. Similarity indexing with the SS-tree. *Proceedings of the International Conference on Data Engineering*, pages 516–523, 1996.

BIBLIOGRAPHY

- [34] L. Wu and T. Bretschneider. VP-EMD tree: An efficient indexing strategy for data with varying dimension and order. *Proceedings of the International Conference on Imaging Science, Systems, and Technology*, pages 421–426, 2004.
- [35] L. Wu and T. Bretschneider. An effective multidimensional index strategy for cluster architectures. *Proceedings of the International Conference on Image and Video Retrieval, Lecture Notes in Computer Science*, 3568:164–173, 2005.
- [36] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. *Proceedings of the Symposium on Discrete Algorithms*, pages 311–321, 1992.
- [37] A. Yoshitaka and T. Ichikawa. A survey on content-based retrieval for multimedia databases. *IEEE Transactions on Knowledge Data Engineering*, 11:1–23, 1999.
- [38] Y. D. Yu and D. S. Kang. Color image compression based on vector quantization using PCA and LEBLD. *Proceedings of the IEEE TENCON*, 2:15–17, 1999.

Acronyms

VA-File:	Vector approximation file.
CBIR:	Content based image retrieval.
$(RS)^2I$:	Retrieval system for remotely sensed imagery.
DCT:	Discrete cosine transform.
VP:	Vantage point.
EMD:	Earth mover's distance.
RR:	Round robin.
PI:	Proximity index.
QBIC:	Query by image content.
DBMS:	Database management system.
MBR:	Minimum bounding rectangle.
IUS:	Informix-universal server.
KCD:	K closest distance.
VCR:	Visiting child order.

Terminology

Feature: Data extracted from image that represent the image in content-based image retrieval process.

Colour histogram: A colour histogram is a vector $(h_1, h_2, \dots, h_n)$, where each element h_j represents the number of pixels falling in bin j in image M . These histograms are the feature vectors (indexes) to be stored as the index of the image database.

De-clustering: Procedure to distribute data to different computing nodes on A parallel computing architecture.

I/O: Input/Output, the term I/O is used to describe any program, operation or device that transfers data to or from a computer and to or from a peripheral device.

YUV: Luminance and chrominance based colour space.

MINDIST: Possible minimum distance from a query point P to a data object contained in the closest facet of a minimal bounding rectangle R .

MINMAXDIST: Possible maximum distance from a query point P to a data object contained in the closest facet of a minimal bounding rectangle R .

Publications

- 1: Li Wu, Timo Bretschneider, An effective multi-dimensional index strategy for cluster architectures, *Proceedings of the International Conference on Video and Image Retrieval, Lecture Notes in Computer Science*, Vol. 3568, pp. 164-173, Singapore, 2005
- 2: Li Wu, Timo Bretschneider, Comparative analysis of the efficiency of R-tree based indexing strategies, *Proceedings of the International Conference on Information and Knowledge Engineering*, pp. 180-184, Las Vegas, U.S.A, 2004.
- 3: Li Wu, Timo Bretschneider, VP-EMD tree: An efficient indexing strategy for data with varying dimension and order, *Proceedings of the International Conference on Imaging Science, Systems, and Technology*, pp. 421-426, Las Vegas, U.S.A, 2004.