Interactive Data Mining And Visualization On Multi-Dimensional Data

By

CHU, Hong Ki

Supervised By :

Prof. M. H. Wong

Submitted to the Division of Department of Computer Science & Engineering

in partial fulfillment of the requirements for the

degree of Master of Philosophy

at the

Chinese University of Hong Kong

June, 1999

多維數據之互動數據開採及形象化

摘要

數據開採一向以來都是計算機科學的熱門題材，不少研究員花了很多努力去尋找怎樣可以從數據中攝取有用知識的有效方法。在眾多數據開採的題目中，最能引起研究員興趣的題目可算是：怎樣從大型數據庫中發掘關聯規則及數據分佈的形態。在眾多被建議的演算法中，確有不少成功的例子。但這些數據開採的演算法都擁有同類的缺點，就是缺乏與使用者之間的互動及沒有提供充足的勘察予使用者。在這篇論文中，本人將會介紹一個名為「互動數值資料分析」的演算法。

「互動數值資料分析」的目標是解決兩個最常見的數據開採疑難，它們分別是數值屬性的開採關聯規則及聚類分佈之發現。這個演算法是屬於遞增模式。在開採過程中，能提供高度的使用者互動性。當與數據顯示器結合後，使用者更能探索到被發掘出的規則或聚類。若要處理高維數據時，還可以應用空間削減技巧來達到高性能的數據聚類過程。總括來說，「互動數值資料分析」是一個高效率的數據開採演算法。

論文作者：朱康岐
哲學碩士二年級學生
香港中文大學電算機科學及工程學系

Acknowledgments

My deepest thanks go to my parents and my brothers. They support me on everything related to my life, and my work.

My special thanks to Prof. Wong Man Hon, CUHK, who was my supervisor in both of my final year project and researches. He gave me a lot of suggestions and all of them are extremely important to my research. Without his help, I am sure that I will not have the chance to present our paper in the conference IDEAS 99 coming summer.

My great gratitude goes to Prof. Ada Fu W. C. and Prof. K. S. Leung, who marked my term papers and participated in the term presentations. I woild like to acknowledge their helpful comments on my works.

Moreover, I have to express my gratitude to the Department of Computer Science & Engineering, CUHK. I have spent almost six years of wonderful time in the department. I would especially like to thank all the staff in the department for providing good equipments.

Finally, I thank my fellow collegues, who were extremely helpful in my studies. They are Chu Kam Wing, Choi Chun Hing, Kwong Wang Wai, Cheng Chun Hung and Chan Kin Pong. I had learnt many things from them through discussions. Sepcial thanks to Choi Chun Hing, Cheng Chun Hung, Kwong Wang Wai, Kam Po Shan, Tse Ming Fun, Cuo Ping, Li Xue Wun and Li Yuan Yuan, who helped me a lot in the tutorial works. I would especially like to thnak Chan Kin Pong for his valuable suggestions to my researches. My special thanks to Lam Shan Shan and Yueng Siu Man, who helped me to prove read my thesis.

... and they bring happiness to my university life.

Interactive Data Mining And Visualization On Multi-Dimensional Data

submitted by

CHU, Hong Ki

for the degree of Master of Philosophy
at the Chinese University of Hong Kong

Abstract

Data Mining has been a hot topic in computer science. Many researchers have been putting lots of efforts on how to extract explicit knowledge from large databases. Among the problems in data mining, discovering *association rules* and finding useful patterns in large databases has attracted lots of interest in recent years. There are many successful algorithms being proposed. However, most of the proposed algorithms for data mining problems are suffering from the same demerit: lack of user interaction and exploration. In this paper, a new algorithm called *Interactive Data Analysis on Numeric-data: IDAN* is being introduced.

IDAN is aimed to solve the two of the most common problems in data mining: mining association rules and discovering clustering patterns with numeric attributes. This algorithm is incremental and providing more user interaction in the mining process. At the same time, it allows the user to explore the rules or clusters being found when integrated with a visualizer. By applying dimensionality reduction techniques, IDAN can provide good performance for clustering high dimensional dataset.

Contents

List of Figures

Chapter 1

Introduction

Recent progress in data retrieving technologies has made us much easier to collect massive amounts of data. Organizations ranging from private business to government bureaucracies are now able to gather a large amount of information, which they are interested. Many of them have accumulated a huge size of data over recent decades. Successful organizations treat such data, as important pieces of infrastructure. They can help them in business decision, like setting up marketing strategies. However as the size of data grows, it is much more difficult for us to analyze such a large amount of data. Methodologies of discovering useful information from these data are highly in demand. This spurs a tremendous interest in the fields of *knowledge discovery* and *data mining*.

The knowledge discovery or data mining can be defined as the process of nontrivial extraction of implicit, previous unknown and potentially useful information from data in databases [9]. An example of information, which is most likely to be interested, is discovering correlation, or co-occurrences of transactional events. We classify this type of problems as mining *association rules*. These rules being discovered are very useful, which can discover interesting relationship among attributes exist in the dataset. A typical example is the basket data model. People are interested in the relations between items bought by customers.

They are looking for rules, which tell them the occurrence of a particular item will probably lead to the occurrence of another item. An example of such an association rule might be that 98% of customer that purchase bread also buy eggs. We call bread as the antecedent and milk as consequent of the rule. The following are some of the examples of associations may be interested:

1. Finding all association rules with 'milk' as consequent. These rules may be very helpful for the shop to plan what to do to boost the sale of a particular items.
2. Finding all association rules with 'chocolate cookie' as antecedent. These rules may be useful for the shop to identify which items will be affected if it discontinue to sell chocolate cookie.

Discovering association rules is very useful when we are hunting for interesting relationships that might contain in database. Useful association rules may be very helpful for us to predict behavior. However, there are many associations rules within the database and not all of them are meaningful to us. So the problem of mining association rules involves not only identifying the rules, we also need to decide which rules is important and useful to us.

Among the techniques in the field of data mining, *cluster analysis* is used to discover interesting distribution and patterns in the underlying data. Clustering analysis is one of the means for us to classify groups of objects by exploring data. Organizing data into sensible groups is one of the most fundamental modes of understanding and learning. In everyday life, children learn how to distinguish between cats and dogs, between tables and chairs, between men and women, ..., etc, by continuously improving subliminal classification system. Classification has always played an essential role in science. The following are some of the examples:

1. In biology, people try to classify living organisms into different kingdoms or classes by investigating the common features of a particular kind of living things.

2. In astronomy, people try to classified stars into various category by measuring their light intensity and their surface temperature.

3. In marketing, researchers try to identify market segments which are groups of customers with similar needs.

In the past, clustering were usually performed in a subjective way, by relying on the perception and judgment of the researchers. They use a specific way, which is well suited to their situations, to perform clustering. In this paper, we will concentrate on clustering objects by means of different measurements or attributes. We use a multi-dimensional vector to represent the measurements of an object.

1.1 Problem Definitions

Now we have some brief idea on the problems of mining association rules and clustering database and how these problems is related to the everyday life. Before we go into the details, we try to introduce the mathematical representation of these problems.

Association Rules

For the problem of discovering association rules, we can formulate the problem in a systematic manner. Given a database T with *attributes* $\mathcal{A} = \{A_0, A_1, A_2, ..., A_n\}$, generally an association rule can be expressed in the form:

$$C_1 \in X \Rightarrow C_2 \in Y$$

where $C_1, C_2 \in \mathcal{A}$ such that $C_1 \cap C_2 = \emptyset$, and X, Y are some values within the domains of C_1 and C_2 respectively.

We called C_1 is the *antecedent* and C_2 the *consequent* of the rule. We can extract over thousands of rules from the database T. There are some measures called *support* and *confidence* on the association rules which reflect the *goodness* of an association rule. The support measures the probability for a record to satisfy both the antecedent and consequent parts of the rule. It is defined as:

$$support = P(C_1 \in X \wedge C_2 \in Y)$$

The confidence is somehow measuring the probability for a record to satisfy the consequent part given that it satisfies the antecedent part. It can be expressed as:

$$conf = P(C_2 \in Y | P_1 \in X) = \frac{support}{P(C_1 \in X)}$$

The support and confidence are having different physical meanings. Rules with high support value do not guarantee to have high confidence. Usually, a 'good' rule should have both high support and confidence.

Clustering

Clustering analysis is the process of classifying objects into subsets. Suppose there are n objects which are denoted by the set $\mathcal{X}$.

$$\mathcal{X} = \{x_1, x_2, x_3, ..., x_n\}$$

where x_i is the ith object. A clustering, $\mathcal{C}$, of $\mathcal{X}$ separates $\mathcal{X}$ into k subsets $\{C_1, C_2, C_3, ..., C_k\}$ satisfying the following:

$$C_i \neq \emptyset \ \ for \ i = [1, k]$$

$$C_1 \cup C_2 \cup ... \cup C_k = \mathcal{X}$$

$$C_i \cap C_j = \emptyset \ \ for \ i, j = [1, k] \ and \ i \neq j$$

We call $\mathcal{C}$ a clustering of $\mathcal{X}$. A clustering is a collection of non-empty sets so that each set must contain at least one object. Besides no object belongs to more than one set. The union of all the sets must equal to the set of all objects. Usually we cluster the objects by looking at the measurements or attributes associated with the objects. Such information is often represented by a multi-dimensional vector.

In this paper, we will concentrate on how to find out the clustering in databases. Each record in the database is treated as a multi-dimensional point by applying simple transformations. We treat the clustering problem as follows: given n data points in a d-dimensional metric space, separate or divide data points into k different groups such that data point within the same group are much more 'similar' to each other than data points in different clusters. The similarity between two data points is usually measured by the distance between them. Data points that are close to each other are similar to each other.

1.2 Experimental Setup

In the research, I have done a series of experiments. These experiments are being done to test the effectiveness and efficiency of the new proposed algorithm. All of the experiments are being done on Ultrasparc 5 machines, with 512MB of main memory. The experiments are written in C++ and the the operation system of the machines are SunOS 5.6.

For the experiments, I have tried to use synthetic databases of different number of records and dimensions. Databases with various distributions are being examined. Besides, I have also tried the dataset used in the Serendip Data Mining Project from Bell Labs. The purpose of using the dataset is to test the effectiveness of the new algorithm.

The databases are in dimensions from 2 to 20 for clustering problem and are in dimensions from 2 to 9 for association rules discovery. The details of the experiments and the result being concluded will be discussed later.

1.3 Outline of the thesis

The thesis is outlined as follows.

In Chapter 2 , I will first state the previous works in the data mining, especially the problem of clustering and the mining of association rules. At the same time, I will try to point out the disadvantage of the traditional approaches and the motivation of my research.

IDAN can also solve the problem of discovering quantitative association rules, In Chapter 3, I will talk about the modification on the algorithm and how it works.

In Chapter 4, I will introduce the new algorithm - Interactive Data Analysis on Numeric-data (IDAN). I will talk about the key idea of the algorithm and how the algorithm can handle the problem of discovering clustering patterns on large numeric data. I will also talk about how we can apply IDAN on high dimensional datasets.

Besides, I have done a series of experiments. In Chapter 5, I will state how the experiments are being done and the result I discovered from these experiments.

In Chapter 6, I will introduction some techniques on visualizing different kind of data.

Finally, I will conclude and discuss our works in Chapter 7. I will make a summary on what I have done in the research and what can be done in the future.

Chapter 2

Survey on Previous Researches

In this chapter, I will try to summarize the previous researches on the field of data mining. At the same time, I would like to discuss what I have learn from the previous researches and what makes me start my research in data mining.

2.1 Association rules

Given a database T with *attributes* $\{A_0, A_1, A_2, ..., A_n\}$, an association rule can be expressed in the form of $C_1 \Rightarrow C_2$ where C_1 is called the *antecedent* and C_2 is the *consequent* of the rule. The value of an attribute A_i of a record R in T is denoted by $R.A_i$. For Boolean association rules, both the antecedent and consequent are restricted to be a set if of *items* (A_i ='yes'). We simply denote *item* as $I \in \{A_0, A_1, A_2, ..., A_n\}$. We say that a record R supports C_1 if C_1 holds in R, i.e. $\forall i \in C_1, R.i =$ 'yes'.

There are two quantities *support* and *confidence* that are used to measure the quality of an association rules. The *support* of rule $C_1 \Rightarrow C_2$ is valued as s if $s\%$ of records in T are supporting C_1 and C_2. The *confidence* of rule $C_1 \Rightarrow C_2$ is valued as c if $c\%$ of records that are supporting C_1, also support C_2. A rule is classified as "good" in the database T if the rule is with *support* and *confidence* excess the thresholds *minsup* and *minconf* that are set by the user in advance. These rules are very common in basket-data-type retail transactions.

The problem can be decomposed into two subproblems:

1. Finding all combinations of items that have transaction support above the minimum support. Call those combinations *large itemsets.*

2. Use the large itemsets to generate the desired rules. For example, if $ABCD$ and AB are large itemsets, then we can determine if the rule $AB \Rightarrow CD$ holds by computing the ratio $c = support(ABCD)/support(AB)$. We say that the rule holds only if $c \leq$minimum confidence.

Among the two subproblems, the first one is much more difficult and the computation is much more. Agrawal, Imielinski and Swami presented an efficient algorithm [1, 28] to discover large itemsets called *Apriori Gen* and had proved very successful. The key ideas of Apriori Gen is that we in fact no necessary to explore all the combinations of the items due the 'upward bounding' properties of large itemsets. For example, if an item $ABCD$ is said to be large, then all the item which are combinations of A, B, C, D such as AB, AC, AD, BC, BD, CD ABC, ACD and BCD are also large.

Boolean association rules are quite restrictive. They can be used in the database with Boolean attributes only. With some modification, categorical attributes can also be processed. However, databases in the real world usually have numeric attributes, Boolean association rules are impractical in such situations. In order to tackle the problem, quantitative association rules were introduced [31, 10]. Quantitative association rules retain the same format as Boolean association rules. The difference between them is that in quantitative association rules, the antecedent and consequent can be conjunctions of item in the form of $(A_i \in [l_i, u_i])$ where $l_i \leq u_i$ and any value between l_i and u_i is within the domain of A_i. We simply denote an *item* as $I \in< A, l, u >$. A record R support C_1 if C_1 holds in R, i.e. $\forall i \in C_1, i(l) < R.i(A) < i(u)$. The definitions of *support* and *confidence* are similar to the ones mentioned above in Boolean association rules.

Recently researches on quantitative association rules concentrate on how to divide the attributes in different ranges so that the problem can be transformed as a problem of discovering association rules with categorical attributes only. As a result, we can use the existing algorithm to mine quantitative rules. Of course, the problem of dividing the domain of an attributes into several ranges is not an easy task. There are many ways to do so and the most common one is the equal-depth bucket approach.

2.2 Clustering

In recent years, many successful clustering algorithms for large datasets have been proposed [18] [20]. These techniques are widely adopted in spatial databases [23], which data points usually form different patterns. Moreover clustering is also an important technique for pattern recognition [7] [11] and machine learning [26] [4]. Some researchers proposed to use *medoids*, the most centrally located point in clusters, to represent the various clusters. The objective is to find out the k best medoids, which can optimize the similarity within clusters under a predefined criterion function. The algorithm is based on randomized search and the problem can be transformed into a graph-searching problem. Later CLARANS [27] was introduced, which improved the performance of traditional *k-medoid* algorithms. CLARANS can successfully cluster data with high quality. However it requires several passes over the dataset, therefore the runtime cost may be unacceptable for large dataset. There are some modifications [8] on CLARANS such as applying R*-Tree [2] to improve I/O efficiency.

The algorithms mentioned above are mainly based on the randomized search. Later, researchers proposed a new algorithm to solve the problem. Instead of randomized search, the algorithm BIRCH [33] is proposed to adopt a hierarchical approach to solve the problem. BIRCH improved I/O complexity by pre-clustering the original data into maximum possible and finest possible subclusters, which can fit in the main memory. At the same time, it makes use of a *CF-tree*, which is a balanced tree structure similar to SS-tree [32]. Instead of using medoids to represent the whole cluster, BIRCH treats each node in the CF-tree as a single cluster. Each node stores the mean of all the points represented in the cluster. Similar to BIRCH, CURE [13] proposed later, also uses a hierarchical structure: k-d tree to make it easier to cluster large set of data. It first draws a set of random samples from the database and partitions the random sample. It starts with each point as a separate cluster and then merges the closest cluster to form a new single cluster. To improve the I/O complexity, CURE uses a heap in each cluster to increase the performance on searching the closest cluster for merging. Rather than using medoids or centroids, CURE uses a number of representatives in the cluster to describe the whole cluster.

In this paper, we are proposing a new algorithm for clustering large datasets. This algorithm is named as Interactive Data Analysis on Numeric-data: IDAN. IDAN clusters large dataset in an incremental and interactive manner. It can partition large dataset into different clusters with good qualities and at the same time it provides good performance. When dealing with high dimensional database, IDAN makes use of dimensionality reduction techniques so that it can perform as good as working of database with few dimensions.

2.3 Motivation

Among all the algorithms of discovering association rules, almost all of them are running in the way that we put the database, Minimum support threshold and minimum confident threshold as input of the algorithm. After running for a period of time, the algorithm outputs all the rules that are satisfying the minimum support and minimum confident thresholds. The process is fully automatic and is running in black box style. No user interaction is required. Since the thresholds play an important role in the choosing of the correct rules, so if the thresholds are wrongly set, then the qualities of the rules being discovered will be greatly affected. The whole process has to be repeated from the very beginning if the user want to change any of the thresholds.

Another problem is that most of the traditional algorithms for discovering quantitative association rules are be applicable to dynamic databases. If there are new data inserted in the database, the mining algorithm has to be started from the very beginning. This is very time consuming and impractical.

For the problem of clustering, most of the traditional approaches are lack of user interaction and exploration. In the whole process of discovering patterns of clustering, the user usually needs to specify some parameters, like the sampling size and the total number of clusters to be discovered. Similar to the problem of mining association rules, the clustering process is also just like a black box. Once the user has input these parameters, the next thing the user has to do is just to sit right in front of the machine and wait for the result. However the clustering process is highly dependent on the quality of data. Different data may require different thresholds in order to provide good clustering result. It is impossible for the user to know the exact values of the parameters in advance without running the process for number of times or exploring the data distribution visually. Once the thresholds are wrongly set, the clustering process has to start from the very beginning. So users have to pay a very expensive cost. The problem mentioned above is mainly due to the black-box manner of the traditional clustering algorithms. The traditional algorithm focus too much on the correctness and the efficiency of the algorithm, on the other hand these algorithms are ignoring the importance of user participation.

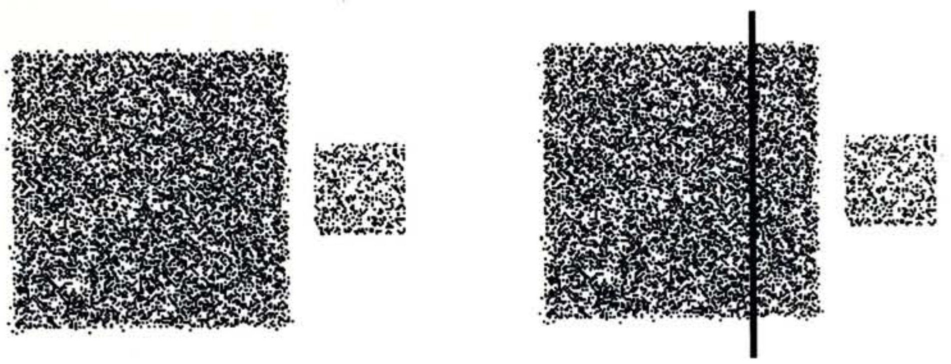
Moreover, some traditional clustering algorithms are suffering from another problem called *mis-clustering* . Traditional clustering algorithms can be classified into *partitional* and *hierarchical* approaches. Partitional clustering algorithms try to discover k partitions by optimizing a certain criterion function. Most of them are making use of the square-error as the standard of judging the similarity within the cluster. The square-error criterion can be summaries as below:

$$E = \sum_{i=1}^{k} \sum_{\vec{p} \in C_i} \| \vec{p} - \vec{m_i} \|^2 .$$

where $\vec{m_i} = \sum_{\vec{p} \in C_i} \vec{p}$ and C_i is the set of points in i th cluster .

However, this measure sometimes will result in mis-clustering. Normally clustering using square-error criterion works well when the clusters are well separated with similar sizes. However when the clusters are in large differences in size and geometry, the square-error criterion will intend to split big cluster into smaller ones so that the sizes of clusters are similar. This situation can be illustrated in figure 2.1.

Traditional hierarchical clustering algorithms also suffer from the problem of mis-clustering. They usually treat each data point as a single cluster at the beginning. Then a pair of clusters are selected and merged together to form a new single cluster. The merging process is repeated until the total number of clusters is reduced to k. The algorithms will choose the pair of clusters, which are the nearest to each other, among all the pairs for merging. The most commonly used measures of distance between cluster C_i and C_j are d_{mean}, d_{max} and d_{min}.

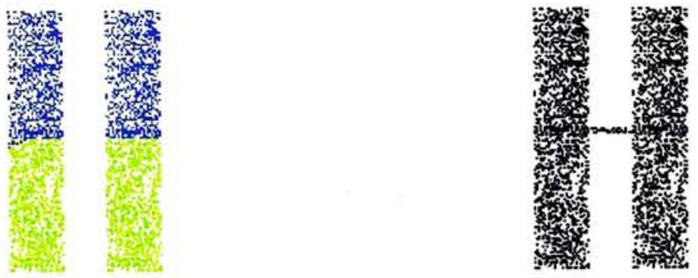

(a) Two clusters with difference in size

(b) Big cluster is split and the small cluster is merged with part of big cluster

Figure 2.1: Problem of mis-clustering

$$
\begin{aligned}
d_{mean}(C_i, C_j) &= \| \vec{m_i} - \vec{m_j} \| \\
d_{max}(C_i, C_j) &= \max_{\vec{p} \in C_i, \vec{q} \in C_j} \| \vec{p} - \vec{q} \| \\
d_{min}(C_i, C_j) &= \min_{\vec{p} \in C_i, \vec{q} \in C_j} \| \vec{p} - \vec{q} \|
\end{aligned}
$$

where $\vec{m_i} = \sum_{\vec{p} \in C_i} \vec{p}$ and C_i is the set of points in the i th cluster .


(a) Distance measure d_{mean} and d_{max}

(b) Distance measure d_{min}

Figure 2.2: Examples of Mis-clustering

When applying different distance measures, we will obtain different clustering patterns. These algorithms can usually partition the data points very well if the data points are well-separated and the clusters are compact with respect to the distance between clusters. However problems arise when the clusters are close together, or the shape and sizes are not hyper-spherical. The distance measures d_{mean} and d_{max} will force the clusters to become hyper-spherical. Long clusters will be split to form smaller clusters since the smaller ones like hyper-spheres more than the long one. The effect is demonstrated in figure 2.2(a). Distance measure d_{min} can work well for clusters with non-spherical or arbitrary shape. However algorithm using d_{min} as distance measure is too sensitive to the existence of outliers. These algorithms will probably merge the two clusters shown in figure 2.2(b) to a single one due to the existence of data point between the two clusters.

Chapter 3

IDAN on discovering quantitative association rules

Data are collections of facts. To provide useful and meaningful information, just to gather these facts different places is not enough. We have to work on a pro-processing stage after all of the essential data are collected. This stage is often regarded as data analysis. Due to the huge amount of data, the task of analyzing these data is usually very difficult. In the early days, the job of data analysis was usually left to the user and the system could at most provide a little help on summarizing the data such as calculating the minimum, maximum, average and the deviation of the data being worked on. Obviously, computers were not playing a significant role in data analysis. On account of the increasing importance of data analysis, people paid more and more efforts on developing powerful algorithm, which could do as much analytical jobs as possible.

Up to now we have spent a lot of time on describing the problems. In the coming chapters, I am introducing a new algorithm called Interactive Data Analysis on Numeric Data - IDAN. This algorithm is an interactive approach of data analysis. By using this algorithm, user can take more control on the knowledge discovery process. Besides, IDAN can work on both the problem of discovering quantitative association rules and the problem of discovering clustering patterns within the dataset. In this chapter, I will concentrate on the details of the algorithm and how it can be applied on the problem of association rules. For the clustering problem, I will discuss it on the chapter 4

3.1 Briefing

IDAN can be divided into two phases: the tree-building phase and the visualization phase. The first phase is mainly concentrating on the construction of an efficient index structure on association rules. This phase runs in an incremental manner. The visualization phase can support interactive browsing of data and extracting useful association rules from the index structure constructed in the previous phase. The key idea of IDAN is the use a hierarchical structure to represent different quantitative association rules with different levels of support. This hierarchical structure is named as **A-Tree**. A-Tree is a height-balanced tree. As mentioned before, the quantitative association rule is built from elements in the form of ($A_i \in [l_i, u_i]$). It is obvious that a hyper rectangular block can represent a quantitative association rule. The structure of an A-Tree node is very similar to the one in *R-Tree* [14]. The choice of using the structure of a R-tree node as the skeleton of A-tree node is not a must. There are many spatial indexing structure can be used. All of them are suitable for the task provided that they are hierarchical in nature and they can access hyper rectangles efficiently. *R-tree* [14], *R^*-tree* [2], *Quad tree* [30] and *X-tree* [3] are some of the examples. The reason of choosing *R-tree* is mainly due to the simplicity of insertion of data and efficiency of spatial access of hyper rectangles. In addition, A-tree node contains two more attributes named as *support* and *opposite*. These additional attributes will give us how good the association rule represented by the node is. I will cover the details of the structure of an A-Tree node in section 3.2.

Once the A-tree is built, we can visualize the association rules discovered. This phase consists of an information visualizer. This can help us to explore the data tuples in the database. The visualizer can show us how the data tuples are distributed and at the same time, we can easily figure out the association rules graphically. Since most of the computation is done in the previous phase, the visualizer will only take care of how to display the information stored in the A-tree graphically. Therefore the visualizer can response quickly and we can also make use of the user interface to tune the *minsup* and *minconf* threshold interactively.

3.2 A-Tree

The structure of an A-tree node is shown in figure 3.1. In the structure of the node of A-tree, the fields *lower* and *upper* are used to store each item of a quantitative association rule. The dimension of these fields is determined by the number of numeric attributes of the database that we are working with. *Size* is used to store the number of tuples that are referenced by the the node. The field *support* is used to hold the number of tuples that are supporting the association rule represented by the node (bounded by lower and uppper). Note that the fields *size* and *support* are measuring different quantities. The first one is referring to the number of leaf nodes of the subtree rooted by the node. The second one, on the other hand, is concerning with the number of tuples spatially fall inside the bounding hyper rectangle represented by the node. *Opposite* is used to hold the number of tuples that support the antecedent part but not fall inside the hyper rectangular block represented by the node. *Parent* and *ptr* are pointers that point to the parent and the children of the node respectively. Each node can have at most N_MAX children. Except the root node, the number of children should be more than or equal to N_MIN. The *support* and *opposite* are

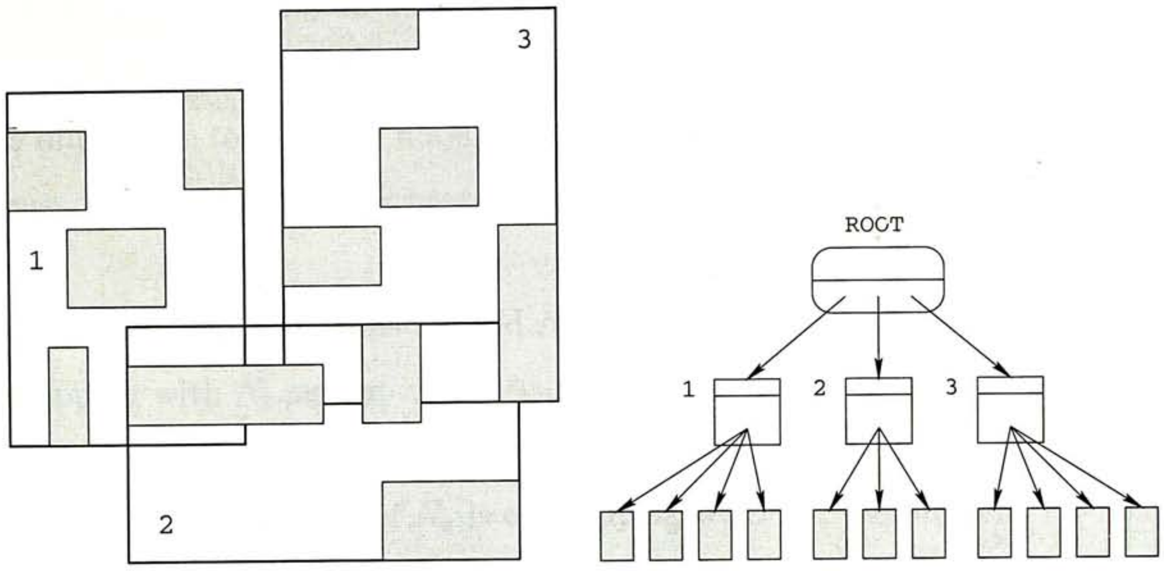
```
struct A_node{
   double lower[DIMENSION];  // li
   double upper[DIMENSION];  // ui
   int size                  // number of tuples reference
   int support;              // total support of the node
   int opposite;             // number of tuples opposite the node
   int attribute;            // indicate the type of the node
                             // root, internal or leaf
   struct A_node *parent;    // point to the parent
   struct A_node **ptr;      // point to the children
}
```

Figure 3.1: Structure of an A-tree node

the individual properties of the node and they are not the *minsup* and *minconf* thresholds, which are user-define properties.

Data tuples in the database are mapped into N-dimensional points. These data points are stored in the leaf nodes. Therefore the number of leaf nodes in the tree is equal to the total number of tuples in the database. Each node is representing a hyper rectangle and the parent node must be a hyper rectangle bounding all the hyper rectangles of its children. The hyper rectangle of the parent node may not be the minimum bounding rectangles(*MBR*) but it is advisable to use MBR since it can reduce both the dummy area and overlap area with other nodes. This will increase the searching and inserting performance of A-tree.

An A-tree will be built dynamically as new data objects are inserted. The size (total number of nodes) of the tree will be directly affected by the parameters N_MIN and N_MAX. The larger N_MIN and N_MAX are, the smaller the tree is in size. Different values of N_MIN and N_MAX can affect the performance of building up the tree. An appropriate selection of the parameters is to make the size of a node to fully occupy a disk page. This will minimize the page accessing time when searching an existing node and inserting a new node. The overall data structure is shown in figure 3.2.



(a) Representation in 2D feature space

(b) Tree representation used in memory or on disk

Figure 3.2: The A-tree structure

Up to now, we have described the structure and the basic requirements of an *A-tree* node. We will then talk about the nature of the *A-tree* node.

Lemma 1 Given a node N and n_1, n_2, n_3, ... n_m are the children of N, then $N.size$ is given by

$$\sum_{i=1}^{m} n_i.size$$

The size of a leaf node is always equal to 1. The above lemma can be proved by induction easily. The proof is simple and is left to the readers. This property can help us to calculate the size of a node efficiently when we are inserting new node into the tree.

Definition 1 Given two nodes N_a and N_b, N_a is overlapping with N_b iff

$$Max(N_a.l_i, N_b.l_i) \leq Min(N_a.u_i, N_b.u_i), for\, i = [1, N]$$

where N = Dimension of the data points.

The relation *overlap* in definition 1 is commutative. The amount of overlapping space will greatly affect the searching and inserting performance in A-tree. It is impossible to make all node not overlap with each others. We are going to discuss how the overlapping properties affect the performance.

Lemma 2 Given two nodes N_a and N_b, if N_a is overlapping with N_b then N_a is overlapping with $N_b.parent$

Proof: From definition 1, if N_a is overlapping with N_b, then for any i = [1,N]:

$$Max(N_a.l_i, N_b.l_i) \leq Min(N_a.u_i, N_b.u_i)$$

Since N_b spatially fall completely inside $N_b.parent$, therefore the following must hold

$$\begin{aligned} N_b.parent.l_i &\leq N_b.l_i \\ N_b.parent.u_i &\geq N_b.u_i \end{aligned}$$

Therefore,

$$\begin{aligned} Max(N_a.l_i, N_b.parent.l_i) &\leq Max(N_a.l_i, N_b.l_i) \\ &\leq Min(N_a.u_i, N_b.u_i) \\ &\leq Min(N_a.u_i, N_b.parent.u_i) \end{aligned}$$

In lemma 2 we can figure out that the overlap relative apply on the parents of the nodes. if node N_a overlaps with N_b, N_a must also be overlapping with $N_b.parent$, $N_b.parent.parent$, ... and so on. The overlapping relationship will propagate upwards. So N_a is overlapping with all the ancestors of N_b.

Lemma 3 Given two nodes N_a and N_b, if N_a is not overlapping with N_b then N_a does not overlap with and nodes in the sub-tree rooted at N_b

Proof: By taking the counter positive of lemma 2, we find that if N_a is not overlapping with $N_b.parent$, N_a does not overlap with N_b. Since N_a is not overlapping with N_b, then N_a does not overlap with any of the siblings and children of N_b. So N_a does not overlap overlap with any descendants of $N_b.parent$. This property is very important because with this property, we can prune out a lot of sub-trees when we are trying to locate all the overlapping nodes.

Lemma 4 Given a node N and n_1, n_2, n_3, ... n_m are the children of N, then

$$N.support = \sum_{i=1}^{m} n_i.size$$

if N does not overlap with any nodes of the same depth.

From lemma 3, if N is not overlapping with any nodes of the same depth, then there are no leaf nodes other than its children spatially fall inside the bounding rectangle specified in N. Since all leaf nodes spatially fall inside the bounding rectangle of N are the children of N, $N.support$ is therefore equal to $N.size$. In this case, the support of a node can be directly determined by the size of its children. Therefore the computing time is limited by the number of children. However it is impossible to have no overlapping nodes. For a more general situation, the support of a node have the property in lemma 5.

Lemma 5 Given a node N and n_1, n_2, n_3, ... n_m are the children of N, then

$$N.support \geq \sum_{i=1}^{m} n_i.size$$

Lemma 6 Given a node N, $N.support$ can be determined by N's children and the nodes overlapping it. Nodes not overlapping with N will not contribute any to $N.support$. i.e. all the leaf nodes in the subtree rooted in the node do no support the association rule represented by N.

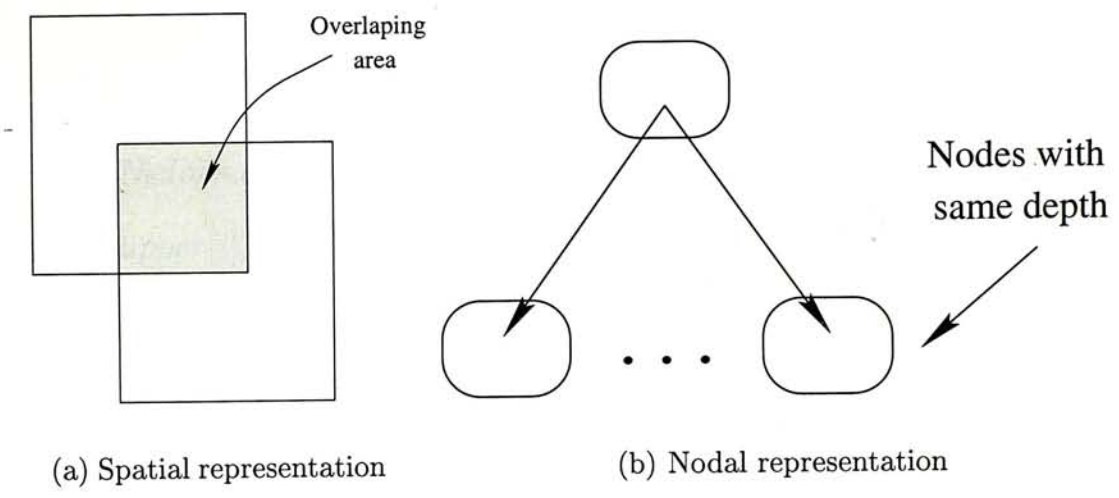


Figure 3.3: Overlapping nodes

Let N_1 and N_2 be nodes in the A-tree. If N_2 is not overlapping with N_1, then from lemma 3 we know that all the descendants of N_2 will not overlap with N_1. Therefore all the leaf nodes in the subtree rooted at N_2 will fall inside the hyper rectangle represented by N_1. As a result, no tuple referenced in N_2 supports the association rule represented by N_1.

We have spent quite a lot of time on the attribute *support* of an *A-Tree* node. Now it is the time for *opposite* attribute. Before we can calculate the *opposite*, we have to go back to the definition of association rule first. An association rule can be divided into two parts: the *antecedent* and *consequent*. Let $\mathcal{A}$ denote the set of attributes in the *antecedent* part and $\mathcal{C}$ denote the set of attributes in *consequent* part of an association rule. Then $\mathcal{A} \cap \mathcal{C} = \phi$ must hold otherwise the rule is not a valid one. To construct an *A-tree*, the user have to specify the set $\mathcal{C}$ first. Without knowing the set $\mathcal{C}$, we cannot calculate the *opposite* of a node. All the attributes other than the ones in $\mathcal{C}$ will form the set $\mathcal{A}$. Without lost of generality, the other of the attributes in the database can be rearrange such that all the attributes in $\mathcal{A}$ comes before the attributes in $\mathcal{C}$. Instead of specifying which attributes are belonging to $\mathcal{C}$, the user can define an integer I such that for all $i < I, A_i \in \mathcal{A}$. Now we have to define what is meant by *opposite*.

Definition 2 A node N_a is said to be **opposing** to an association rule represented by node N_b iff the following conditions hold

$$N_b.lower[i] \leq N_a.lower[i] \leq N_a.upper[i] \leq N_b.upper[i] \quad , for \quad i \in \mathcal{A}$$
$$Min(N_a.upper[i], N_b.upper[i]) < Max(N_a.lower[i], N_b.lower[i]) \quad , for \quad i \in \mathcal{C}$$

In general, there are two conditions for opposing. First of all, the antecedent part of N_a must fall completely inside the antecedent part of N_b. Secondly, the consequent part of N_a fall outside the consequent part of N_b and they do not overlap with each other. The opposite of a node N is the number of leaf nodes in an A-Tree opposing the association rule represented by N. Instead of counting the opposite of a node directly, we try to make a simple calculation. Consider all the records that are supporting the antecedent part of N only, we can classify these tuples into two classes: the supporting tuples and the opposing tuples. We have discussed the calculation on the support of a node previously so the number of supporting tuples can been found. If we can count to number of tuples supporting the antecedent part of N, we can calculate the opposite of a node.

$$N.opposite = \#\, of\, leaf\ nodes\ supporting\ the\ antecedent\ part\ of\ N - N.support$$

The way of finding the number of leaf nodes supporting the antecedent part of a node N is very similar to finding *N.support* mentioned previously. The only difference is that *N.support* requires the leaf nodes to fall inside the hyper rectangle for all attributes but now it only requires those attributes found in the antecedent part.

3.3 Insertion Algorithm

A-tree will be built dynamically when data are added into the database. The flow of the insertion algorithm can be summarize as in figure 3.4. When a new tuple is really to be added to the database, the first step is to identify the appropriate leaf node that the new tuple is going to be inserted. This step is being done by the procedure ChooseLeaf. ChooseLeaf will recursively traversal the tree until a leaf node is found. Then ChooseLeaf will return the parent of the leaf node just found. During the traversal of the tree, if there are more than one child in the node, we will first calculate the increase in margin after the insertion of the tuple and break the tie by considering the increase in area. The reason of choosing margin for comparison instead of using area is that using area may lead to some faulty attribution on the association rule. This situation occurs when the shape of the node is 'thin'.

After choosing an appropriate node, the next job is to insert the new tuple into it. If the leaf is not full yet, the new tuple can directly insert to the leaf. However, if the leaf is already saturated, overflow treatment has to be done. In this stage, the overflow treatment procedure will split the saturated node into two. There are many alternatives can be done, such as reinsertions of the overflow nodes [2]. A good overflow treatment procedure can improve the performance of future insertion and the performance of query.

To split a node into two, we first sort the children of the splitting node in ascending order by a particular dimension. We then choose a dimension such that the sum of area of all combinations will be the smallest. Then choose the index $i = [N\_MIN, count]$ such that the area sum of the two newly formed nodes is the smallest. After the splitting, the two newly formed nodes will be inserted to the parent recursively. The parent node may be overflow again. The overflow treatment will be carried on until no overflow happens again.

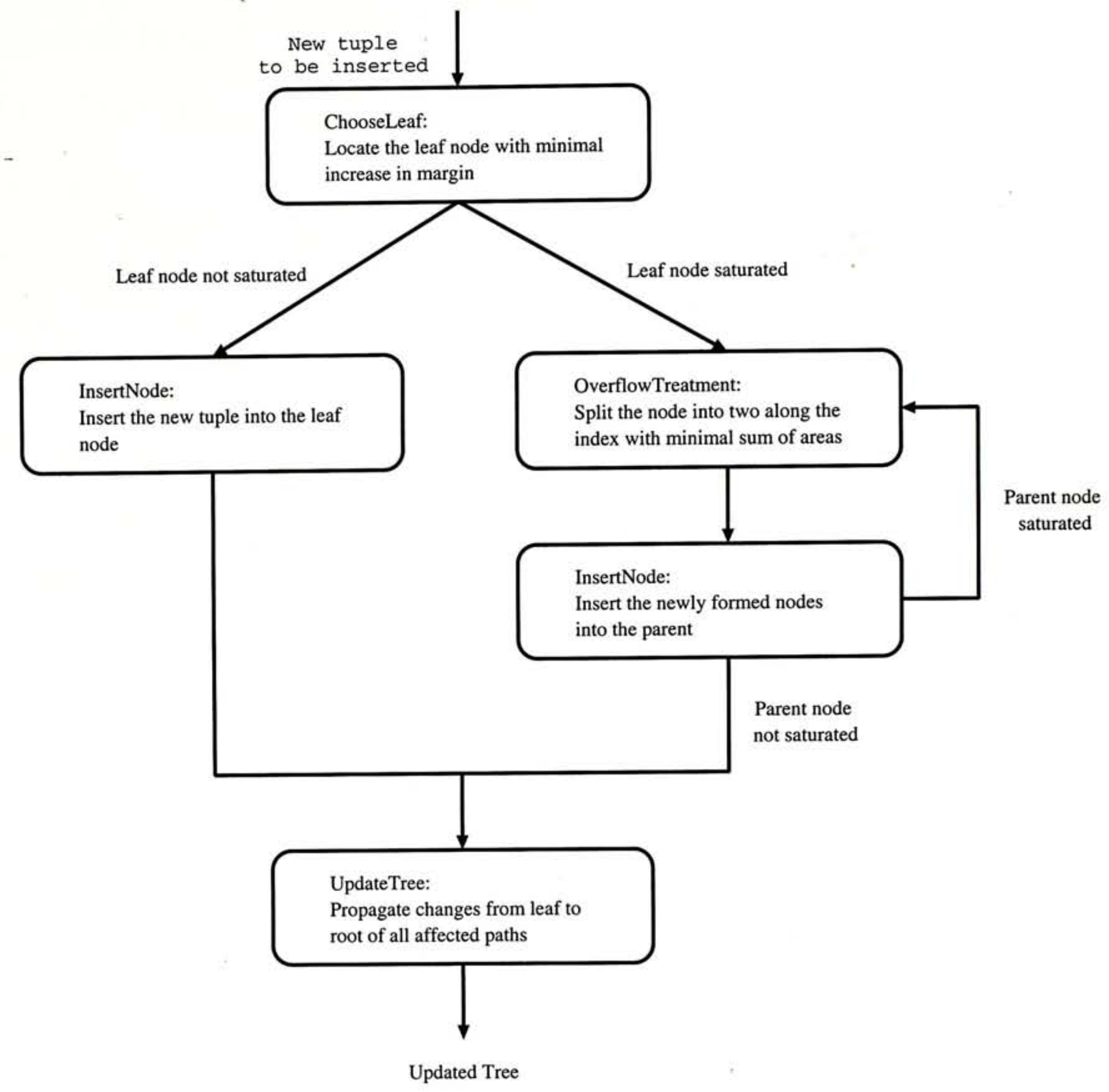
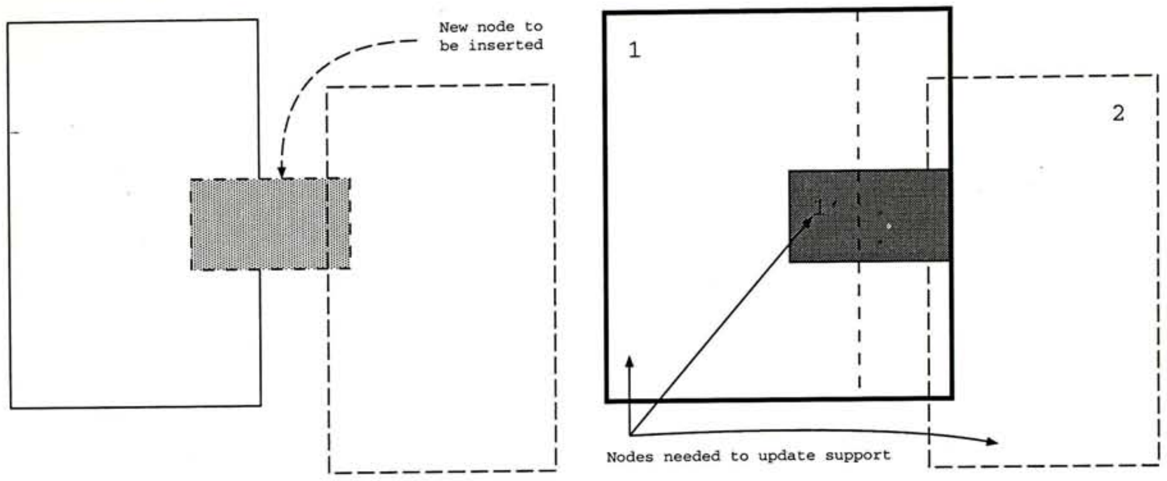

New tuple to be inserted
ChooseLeaf: Locate the leaf node with minimal increase in margin
Leaf node not saturated
Leaf node saturated
InsertNode: Insert the new tuple into the leaf node
OverflowTreatment: Split the node into two along the index with minimal sum of areas
Parent node saturated
InsertNode: Insert the newly formed nodes into the parent
Parent node not saturated
UpdateTree: Propagate changes from leaf to root of all affected paths
Updated Tree

Figure 3.4: Flow of insertion algorithm

Once the new tuple is inserted into the tree, we have to propagate the change from leaf to root. There are several things have to be modified. First of all, we have to modify the *lower* and *upper* attributes since the volume of the hyper rectangular block will expand or shrink after the insertion or splitting of nodes. We can calculate the minimal bounding rectangle of a node from its children. It is not necessary for us to update all the nodes in the tree. We can simply update the nodes being inserted (the new node and the nodes after splitting) and their ancestors only.After updating the lower and upper attributes, the next thing we have to update after insertion is the attribute *size*. The size of node has been discussed in detail in the previous section. We know that the size of a node is determined by its children only. We can modify the size of the nodes involved in the insertion by backward tracing from the newly inserted node.

In the updating stage, the most difficult thing is to update the *support* and *opposite* fields. When a new node is being inserted into the tree, the support of its ancestors of course have to be update. However since the hyper rectangle of the nodes have been modified, other nodes have to be updated as well. A simple solution to the problem is to parse the tree node by node and recalculate the support and opposite for each on the nodes. This requires too much computation and disk accessing time therefore it is impossible and impractical to do so. In fact, not all the nodes in the tree need updating. We can prune out those nodes and as a result we can improve the performance.

In the previous section, we have proved that not all the nodes in the tree have contribution on the support of a particular node. Only those node overlapping with it will contribute to its support. By using this property, we can test the subtree before we parse it. If it does not overlap with the affected nodes, we can simply ignore them. The way we are using is to make use of three queues: the affected queue, support queue and opposite queue. When we do the insertion, we put the nodes that are being affected into the affected queue. These nodes include the newly inserted node and the nodes involved in splitting. After we have update the fields lower, upper and size, we then parse the tree from the root to leaf. We put those nodes overlapping with the ones in the affected queue into the support queue. At the same time we put the nodes overlapping the antecedent part of the ones in the affected queue into opposite queue. Now we have already filtered out the nodes that need updating. The thing that we have to do is to calculate the support and opposite in the support and opposite queues respectively.



(a) Before insertion (b) After insertion

Figure 3.5: Update support

3.4 Visualizing Association Rules

After integrated with an information visualizer, we can simply display the association rules found by parsing the A-tree node by node. Of course, not all the nodes are informational since some of the nodes are with very low support or confidence. Therefore, we have to find an easy way to select the rules that are interesting to the user. The *minsup* and *minconf* thresholds are playing an important role in this stage.

Before we parse the A-tree, we already know the exact number of tuples stored in the tree. There is no need to count the number of tuples since the total number of tuples is already found in the size or support fields of the root node. While we are parsing the A-tree, we first calculate the support and confidence value of the node. The support and the confidence of the node can be easily formulated as follow:

$$support = \frac{s}{T}$$

$$confidence = \frac{s}{o + s}$$

where s is the support count of the node and o is the opposite count of the node

Before plotting the hyper rectangular block represented by the node, we first compare the calculated support and confidence with the support and confidence thresholds set by the user. If the calculated ones are at least equal to the thresholds, we can plot the hyper rectangular block directly, otherwise, we discard the whole sub-tree. This is due to the properties of an increasing support count along the path from leaf to root. There is no simple pruning algorithm for confidence since the confidence does not follow the increasing property like the support count. However the pruning of support count has already cut out a large number of candidates so without pruning out the those nodes with insufficient confidence, the performance is still satisfactory. There are some examples on the exploration of the association rule found in figures 3.6 – 3.9.

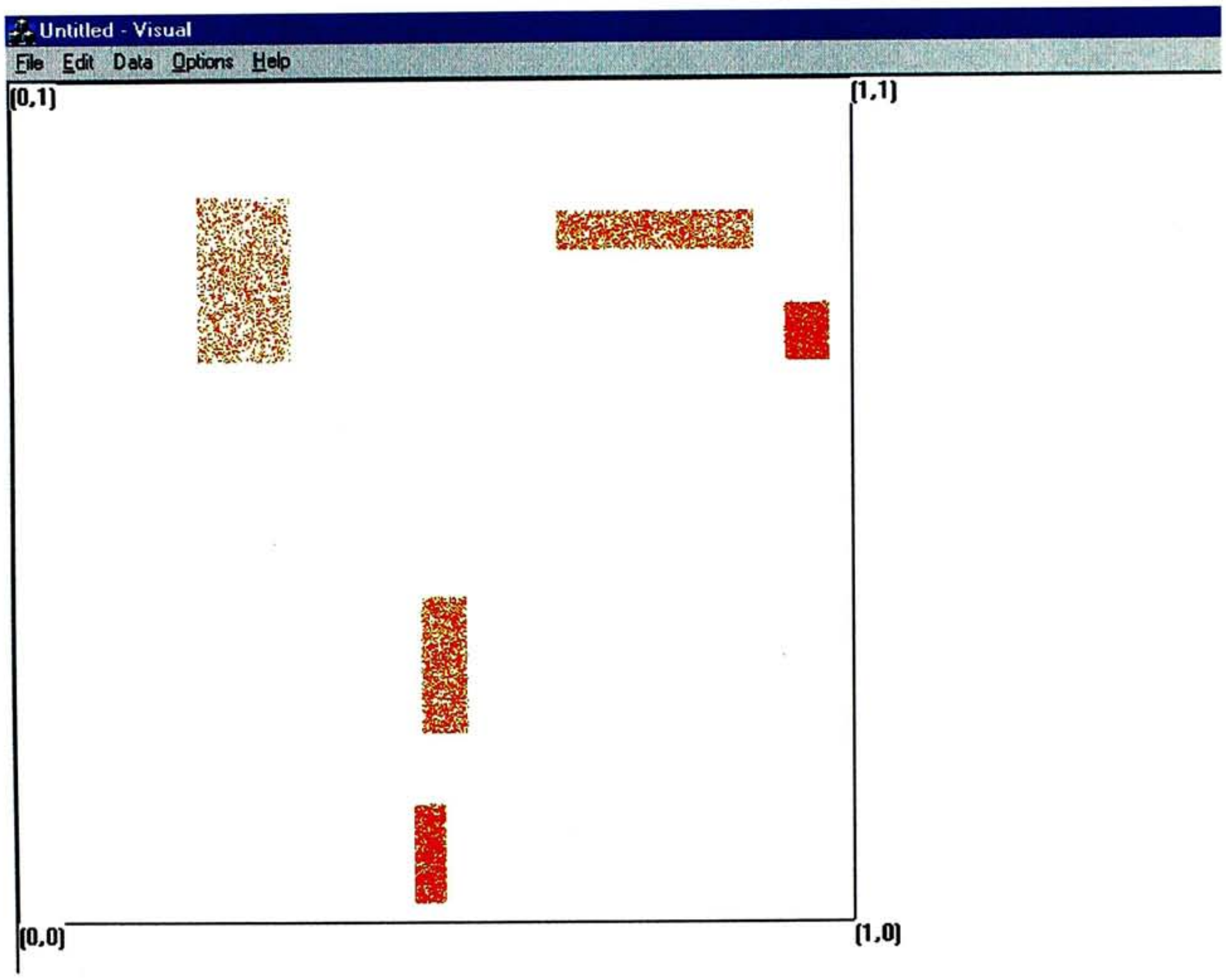
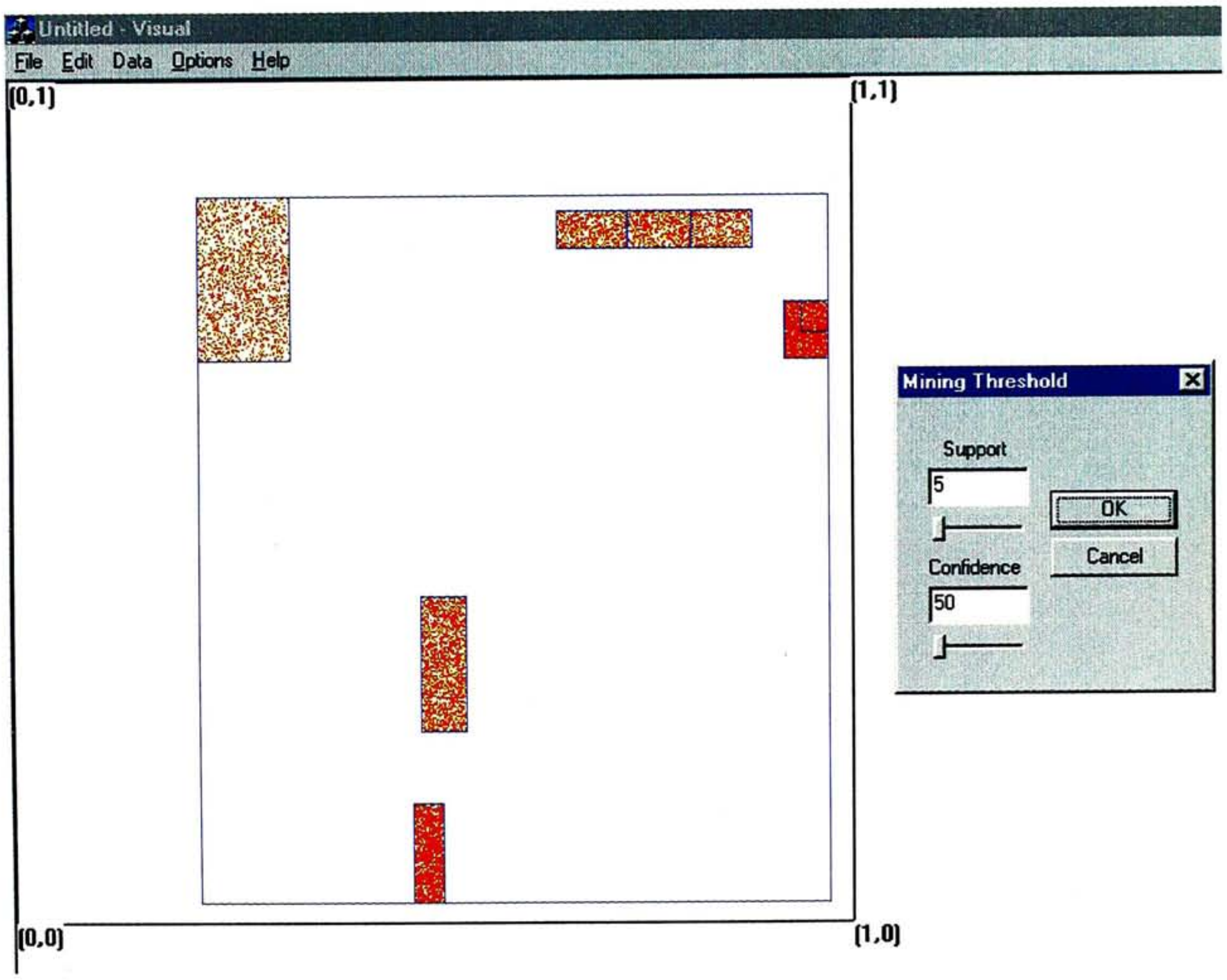
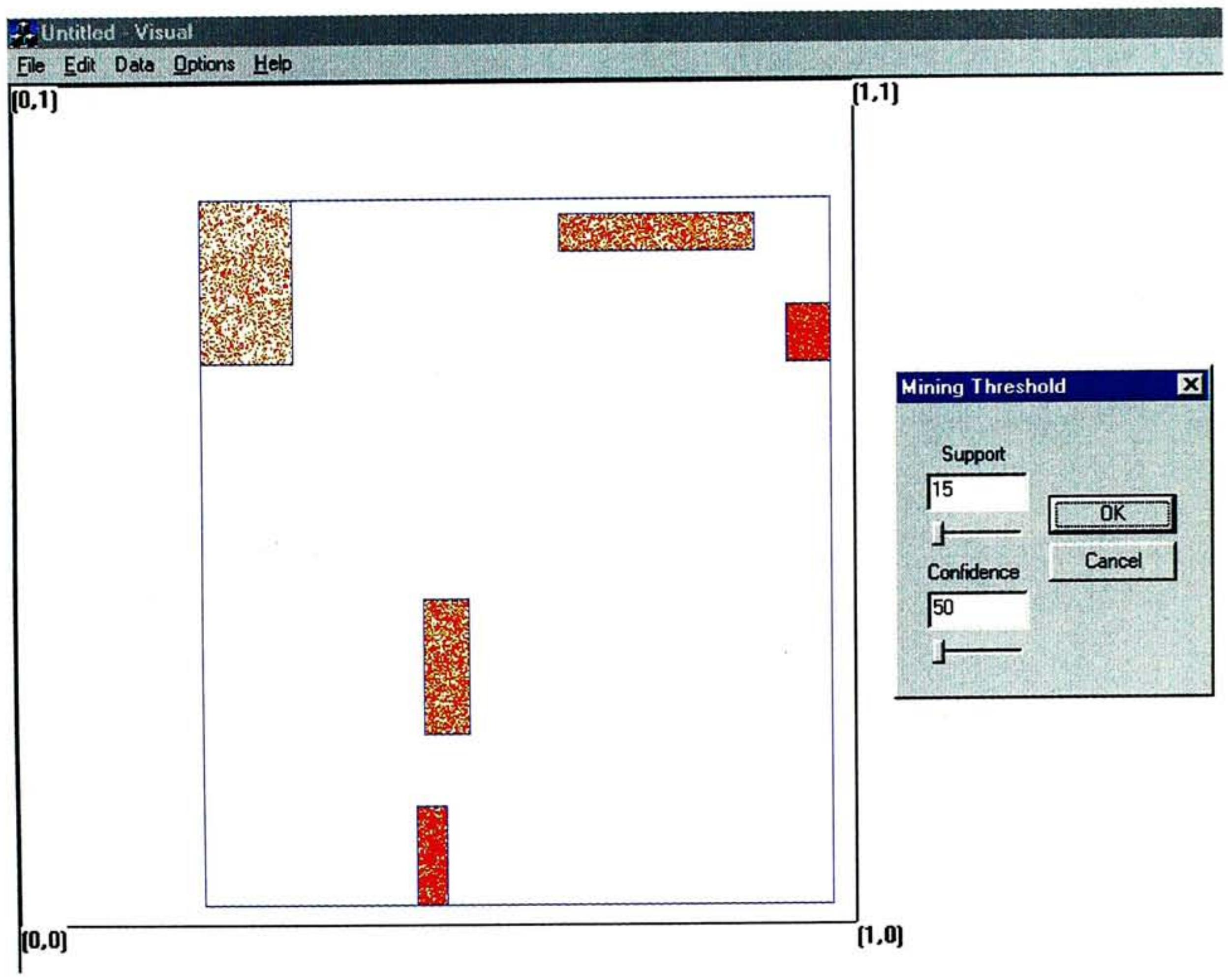


Figure 3.6: Original dataset



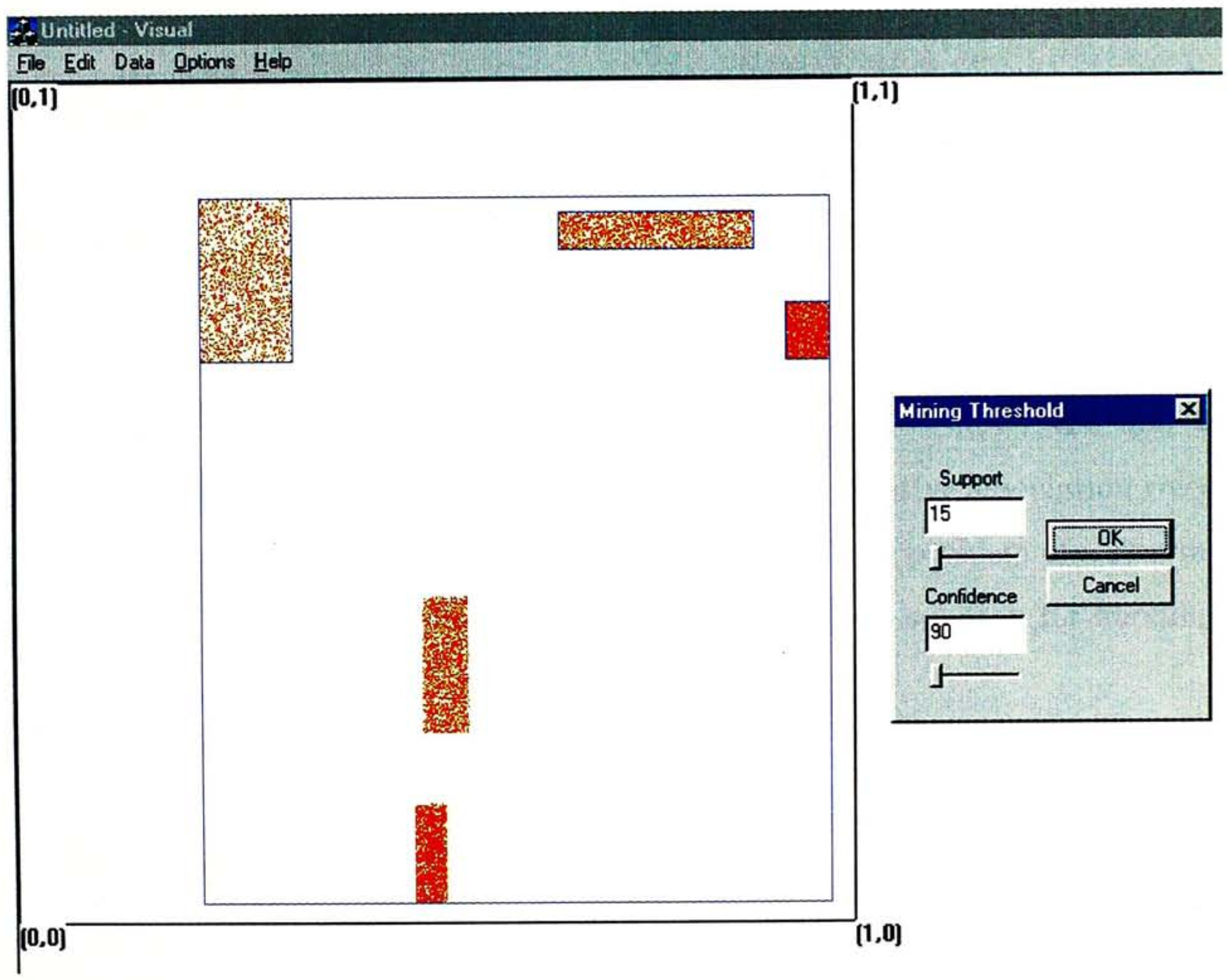
Support
5
Confidence
50

Figure 3.7: Rules with support 5% confidence 50%



Support
15
Confidence
50

Figure 3.8: Rules with support 15% confidence 50%



Support
15
Confidence
90

Figure 3.9: Rules with support 15% confidence 90%

Chapter 4

IDAN on discovering patterns of clustering

In the previous chapter, we have discussed the key idea of IDAN and how to apply the algorithm in the problem of discovering quantitative association rules. In this chapter, I will talk about how to apply IDAN on the problem of clustering and the corresponding modifications. Also, I will suggest solution for handling high dimensional datasets.

4.1 Briefing

IDAN adopts an interactive approach of analysis on numerical data. Its aim is to discover clustering patterns of high-dimensional metric data. IDAN can be divided into two phases: the tree-building phase and the visualization phase. The first phase is to build an efficient index structure on the dataset in an incremental manner. The visualization phase can support both interactive browsing of data and interactive formulation of the clustering being discovered in the previous phase. Once the tree-building phase finishes, the process is not necessary to start again when the user changes any of the parameters such as the total number of cluster to be discovered. Figure 4.1 shows the overview of the two phases involved in IDAN.

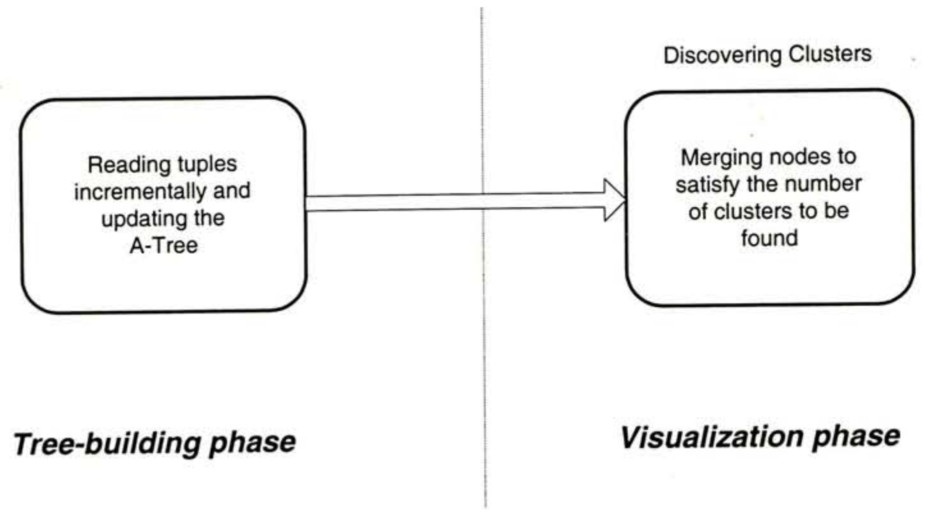

Discovering Clusters
Reading tuples incrementally and updating the A-Tree
Merging nodes to satisfy the number of clusters to be found
Tree-building phase
Visualization phase

Figure 4.1: Overview of IDAN

The key idea of IDAN is the use of two data structures: *R-Tree* [14] and *A-Tree.* The first one is an index structure that stores all the data points in the database. It is a height-balanced tree that can support storing and retrieving multi-dimensional data objects efficiently . The second data structure is proposed by us, which is aimed to represent the clustering information within all the data points in the database. We will go into the details of the structure of an A-Tree node in the section 4.2. The relationship between the two data structures is that there are links between the leaf nodes of the two structure. Since both of the trees keep the data points on the leaf level, we can simply link up the leaves by pointers if the leaves are referring to the same data point.

Once the A-Tree is built, we can visualize the discovered clustering patterns. The visualization phase consists of an information visualizer, which can help us to explore the data in the database. The visualizer can show us how the data distributed and at the same time, we can easily figure out the clustering patterns. Since most of the computation is done in the tree-building phase, the visualizer involves little computation and mainly concentrates on displaying the information stored in the A-Tree graphically. Therefore the visualizer can response quickly and allow the user to modify the parameters or thresholds interactively through its interface.

4.2 A-Tree

There are three kinds of nodes in an A-Tree: root node, internal nodes and leaf nodes. There is only one root node in an A-Tree, which is the ancestor of all the other nodes in the tree. The leaf nodes are used to store the data tuples. Data tuples in the database are mapped into N-dimensional points. These data points are stored in the leaf nodes. Therefore the number of leaf nodes in the tree is equal to the total number of tuples in the database. A leaf node is said to be *referenced* by a node if and only if the node is the ancestor of the leaf node.

The structure of an A-Tree node is shown in figure 4.2. The field *mean*, which is also a N-dimensional vector, is the average of all the leaf nodes being referenced by the node. *Size* is used to store the number of leaf nodes that are referenced by the node. The pointer *parent* is used to indicate the parent node and *ptr* is an array of pointers that point to the children of the node. *Attribute* is used to indicate whether the node is a root, internal or leave node. Each node can have at most N_MAX children. Unlike R-Tree, there is no limitation on the minimum number of children of a node in A-Tree.

```
struct A_node{
   double mean[DIMENSION];   // Mean of data points referenced
   int size                  // number of tuples referenced
   int attribute;            // indicate the type of the node
                             // root, internal or leaf
   struct A_node *parent;    // point to the parent
   struct A_node **ptr;      // point to the children
}
```

Figure 4.2: Structure of an A-Tree node

An A-Tree will be built dynamically when new data objects are inserted. The size (total number of nodes) of the tree will be directly affected by the parameter N_MAX. Different values of N_MAX can affect the performance of building up the tree. In order to minimize the page accessing time when searching an existing node and inserting a new node, selection of the parameter should allow a node to fit in a disk page completely.

4.3 Dimensionality Curse

Several data structures such as R-Tree, SS-Tree, SR-Tree [19] and TV-Tree [25] are intended to provide fast searching in large multi-dimensional database. Experiments show that these data structures can work efficiently on small dimensions (below 10 dimensions). However searching performance of these structures degrades as dimensionality grows. When searching data with high dimensionality, even a sequential search can out perform any searching using these data structures. This phenomenon is so called *dimensionality curse*, which can usually be found among these multi-dimensional data structures.

In order to tackle the problem, the technique of *dimensionality reduction* is commonly employed. The key idea of dimensionality reduction is to remove a certain number of dimensions and at the same time to preserve as much information as possible. We first apply transformation on high dimension data, so that most information converge into a few number of dimensions. These dimensions are used for indexing by the mentioned data structures. Since the number of *chosen* dimensions is very small, so the mentioned data structure can provide very good query performance. There are many transformations such as Singular Value Decomposition (SVD) [29], Discrete Fourier Transform (DFT), Discrete Wavelet Transform (DWT). Different transformations work well on different kinds of data. For instance, DFT and DWT are used in the area of time series data and image databases. For SVD, it studies the whole dataset and tries to maximize the variances in a few dimensions. I will try to introduce the central ideas of these transformations in the coming sections.

4.3.1 Discrete Fourier Transform

Discrete Fourier Transform (DFT) is a variation of Fourier transform (FT). In 1807, Joseph Fourier announced his surprising results and his statement played an essential role in the evolution of mathematicians' ideas. The Fourier transform is just like a mathematical prism, breaking up a function into the frequencies that compose it, as a prism break up light into different colors which are in fact electro-magnetic waves of different frequencies.

Let $f(x)$ be a continuous function of a real variable x. The *Fourier transform* of $f(x)$, denote $\mathcal{F}\{f(x)\}$, is defined by the equation

$$\mathcal{F}\{f(x)\} = F(u) = \int_{-\infty}^{\infty} f(x)e^{-j2\pi ux}dx$$

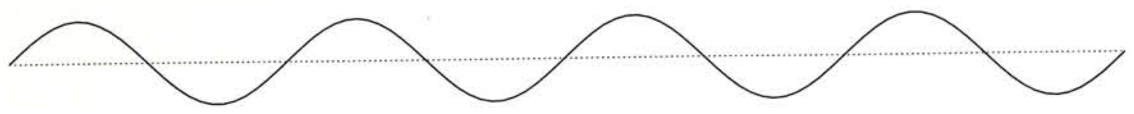
where $j = \sqrt{-1}$.

Fourier transform of function $f(x)$ exists if $f(x)$ is continuous and integrable. However, we are not interested in continuous function. Data tuples exist in the form of discrete values rather than as a continuous function. So the discrete version of Fourier transform will be much useful for our problem.

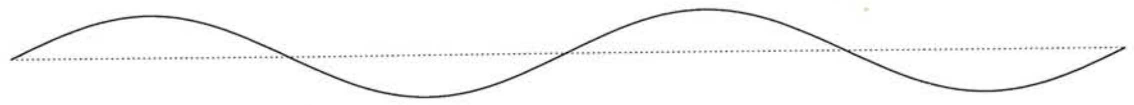
Suppose that a continuous function $f(x)$ is separated into a sequence x_0, x_1, x_2,...,x_{d-1} by taking d samples Δ units apart which

$$x_i = f(x' + i\Delta)$$

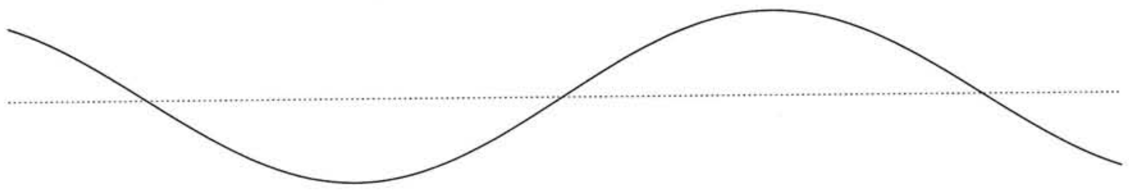
Then the *discrete* Fourier transform applying to the sampled functions is given by



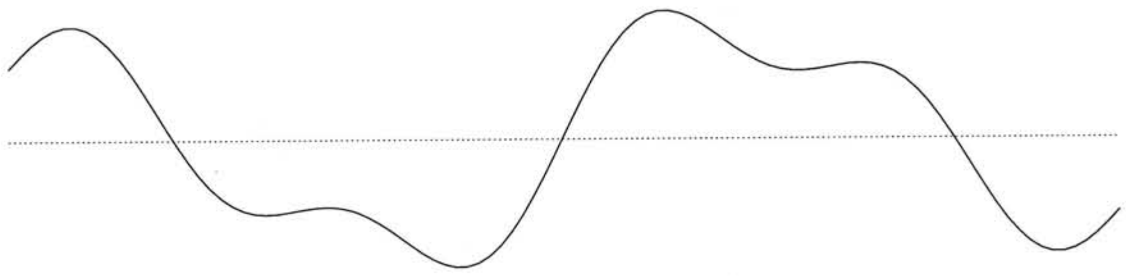
(a) $0.5\, sin(x)$



(b) $0.5\, sin(x/2)$



(c) $sin(x/3)$



(d) Function $f(x)$

Figure 4.3: Fourier transform tries to break up function $f(x)$ into components of different frequencies

$$F_u = \frac{1}{\sqrt{d}} \sum_{i=0}^{d-1} x_i e^{-j2\pi ui/d} \qquad for\ u = 0, 1, ..., d-1$$

DFT is a very useful transformation which works on image databases and time series data. Instead of working on these datasets directly, we have to extract *feature vectors* from them first. Let $\vec{x} = \{x_i\}_{i=0}^{d-1}$ be a d-dimensional feature vector. We treat each feature vector as a finite-length sequence of length d. Then we perform the DFT on the sequence and we get $\vec{F} = \{F_u\}_{u=0}^{d-1}$.

$\vec{F}$ is also a a finite-length sequence of length d. Usually, most of the information will concentrate in the first few values of the sequence. So we can truncate the sequence $\vec{F}$ in the frequency domain to k terms, to form $\vec{F_k}$, which $k \leq d$. After truncation, the dimensionality of the dataset is reduced from d to k.

Discrete Fourier transform turns a sequence into frequency domain. It works well with image database and time-series data. Moreover, there is fast algorithm for DFT with complexity of $O(d \cdot log_2 d)$ and it can be implemented on hardware. So DFT is widely applied on these systems.

4.3.2 Discrete Wavelet Transform

The fundamental idea behind wavelets is to analyze according to scale. In wavelet analysis, the scale that one uses in looking at data plays a special role. Wavelet algorithms process data at different scales or resolutions. The result in wavelet analysis is to "see the forest and the trees."

Wavelet transform is similar to Fourier transform in the sense that both transforms can be viewed as a mapping from function space to a different domain. For wavelet transform, this new domain contains complicated basis functions called wavelets, mother wavelets or analyzing wavelets. So wavelet transform comprises an infinite set. Different wavelet families make different trade-offs between how compact the basis functions are localized in space and how smooth they are.

Among different families of wavelet transform, Haar is the simplest one. So we take Haar wavelet as an example. The Haar transform is based on the Haar function

$$\psi(x) \equiv \begin{cases} 1 & 0 \leq x \leq \frac{1}{2} \\ -1 & \frac{1}{2} \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

and

$$\psi_{jk}(x) \equiv \psi(2^j x - k)$$

For $d = 8$, the Haar functions are defined as

$$\begin{aligned}
\psi_{00} &= \psi(x) \\
\psi_{10} &= \psi(2x) \\
\psi_{11} &= \psi(2x-1) \\
\psi_{20} &= \psi(4x) \\
\psi_{21} &= \psi(4x-1) \\
\psi_{22} &= \psi(4x-2) \\
\psi_{23} &= \psi(4x-3).
\end{aligned}$$

With the help of Haar functions, any function $f(x)$ can be written as a series expansion by

$$f(x) = c_0 + \sum_{j=0}^{\infty} \sum_{k=0}^{2^j-1} c_{jk}\psi_{jk}(x).$$

Let a function be defined on d intervals, with $d = 2^a$. Then an arbitrary function can be considered as an d-dimensional vector $\vec{F}$, and the coefficient in the expansion $\vec{B}$ can be determined by solving the matrix equation below.

$$\vec{F} = W_d \vec{B}$$

With the Haar basis, we can construct Haar transformation matrix W_d of order $d \times d$ by formation of the ith row from elements of $\psi(x)$ The followings are examples of Haar transformation matrices.

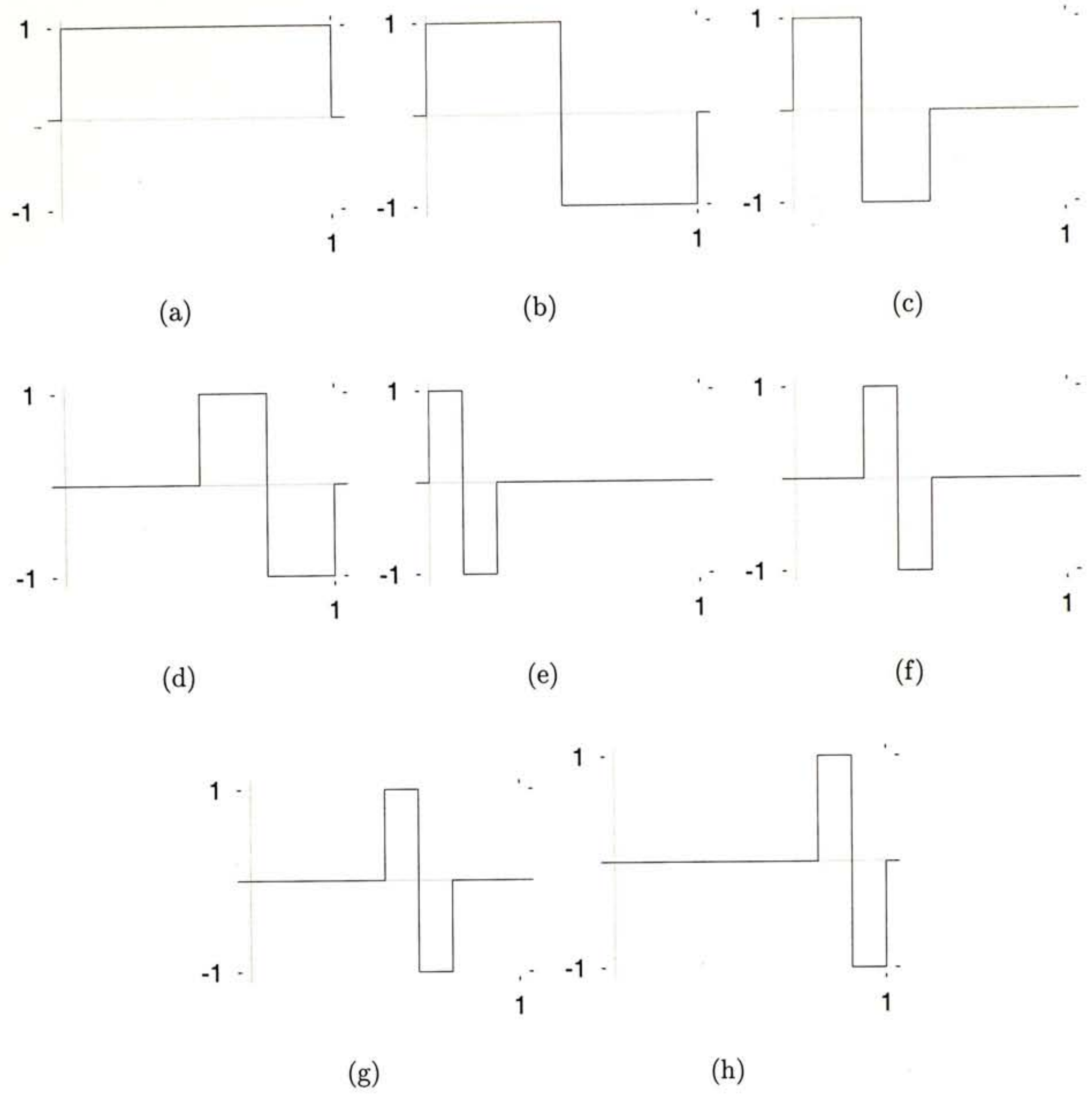


Figure 4.4: Different Haar functions for $d = 4$

4.3.3 Singular Value Decomposition

In IDAN, we adopt singular value decomposition (SVD) as the dimensionality reduction function because SVD is much more suitable for most of the dataset. SVD does not require data to be related in the sense of 'frequency' or 'resolution'. It only studies the distribution of data points. Moreover, SVD works on the whole dataset and it will give higher precision when compared with transformation that processes each data point individually.

$$W_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad W_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

(a) $d = 2$ (b) $d = 16$

Figure 4.5: Haar transformation matrices

SVD methods are based on the theorem of linear algebra. Firstly we represent every data points in an matrix. Suppose we have n d-dimensional data points, it is trivial that we can use an $n \times d$ matrix to express them. For any $n \times d$ matrix X with number of rows n greater than or equal to its number of columns d, can be expressed as the product of an $n \times d$ column-orthogonal matrix S, an $d \times d$ diagonal matrix W and the transpose of an $d \times d$ orthogonal matrix V.

$$X = SWV^T$$

The matrix S is an $n \times d$ orthonormal matrix that is, $S^T S = I_d$. W is an $d \times d$ diagonal matrix with positive diagonal elements $w_1, w_2, ..., w_n$. These elements are so called singular value of X. Since W is diagonal, $W^T = W$. The matrix V is an $d \times d$ orthonormal matrix, then $V^T V = I_d$. The SVD decomposition can also be carried out when $n \leq d$. In this situation the singular values w_j for $j = n + 1, ..., n$ are all zero and the corresponding columns of S are also zero. Our goal is to compute the transformation matrix V. The transformation is illustrated in the figure 4.6.

Now we consider the SVD of X^T and $X^T X$:

$$\begin{aligned} X^T &= (SWV^T)^T = VWS^T \\ X^T X &= (VWS^T)(SWV^T) = VW^2V^T \end{aligned}$$

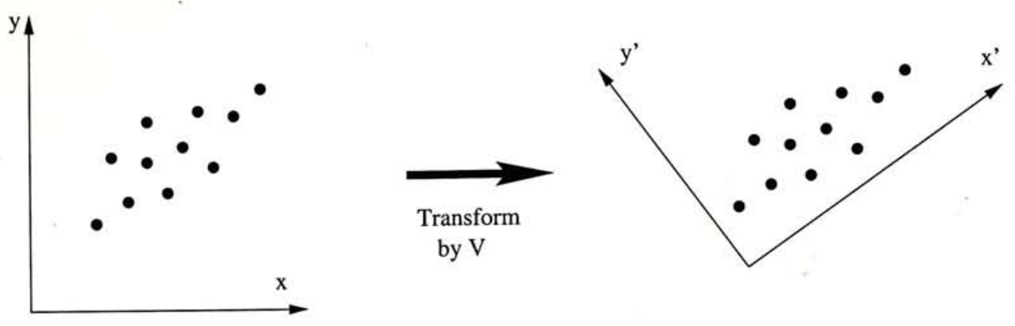


Figure 4.6: Axes change from (x, y) to (x', y') by matrix V.

We can figure out that the matrix V is also the SVD transformation matrix of X^TX. Since the transformation matrix V is the same in either of the case, it takes no difference to adopt which one. In IDAN, we use X^TX instead of X directly. The reason is adopting X^TX is to minimize the computation time and the memory usage in the SVD transformation. The dimensions of X and X^TX are $n \times d$ and $d \times d$ respectively. For databases which are large in size, n should be much larger than d. It is trivial that the memory usage of X^TX is much lower than the one for X. Besides, the computation time of SVD will also be lower for X^TX when comparing with the one for X.

However you may figure out that the cost of calculating X^TX is very high. We agree with you if we have to compute X^TX from X only. The computation of X^TX takes $O(n * d^2)$ time. However if we incrementally update the matrix X^TX, the computation of updating X^TX, when a new record is being inserted, is only $O(d^2)$. Let x_i be a d-dimensional vector representing i th data tuple and X_i be the matrix representing all the first i th data tuples. Then $X_{i+1}^T X_{i+1}$ can be computed by:

$$X_{i+1}^T X_{i+1} = X_i^T X_i + x_{i+1}^T x_{i+1}$$

After calculating the transformation matrix V, the next thing to do is to extract the most significant components. We then sort the rows of matrix V

in the descending order of the corresponding singular values. Since the singular value w_i indicates the variances along the ith dimension, so we extract the first d' dimensions so that they must contain most of the information within the data. We compute the sum of all the singular values in W and then extract the first d' dimensions which contain more the threshold $\theta\%$ of the sum of the singular values. Note that the parameter θ will only affect the performance of the overall algorithm and will not affect the accuracy of the algorithm. Then we build the index structure on the dimensionality reduced dataset. In IDAN, we use R-Tree, the choice of R-Tree is simply based on the simplicity of insertion and searching operations.

The cost of computing SVD transformation matrix is high. When we apply SVD on dynamic database, it is impossible for us to update the transformation matrix V every time when a new record is being inserted. The way we work is to measure the loss in accuracy and update the transformation matrix only when the loss become too large. We will discuss this later.

4.4 IDAN - Algorithm

As mentioned in the previous sections, IDAN clusters multi-dimensional database in an incremental and interactive manner. It keeps the most updated information on the clustering patterns of the database. New records are inserted into the database incrementally and IDAN will update these information dynamically. The insertion procedure is shown in figure 4.7.

IDAN tries to group similar tuples together to form a cluster. When a new tuple is added to the database, we have to identify which data point in the database will merge with the new tuple. Without the help of the dimensionality reduced R-Tree, we have to perform a nearest neighbor search on the A-Tree. It

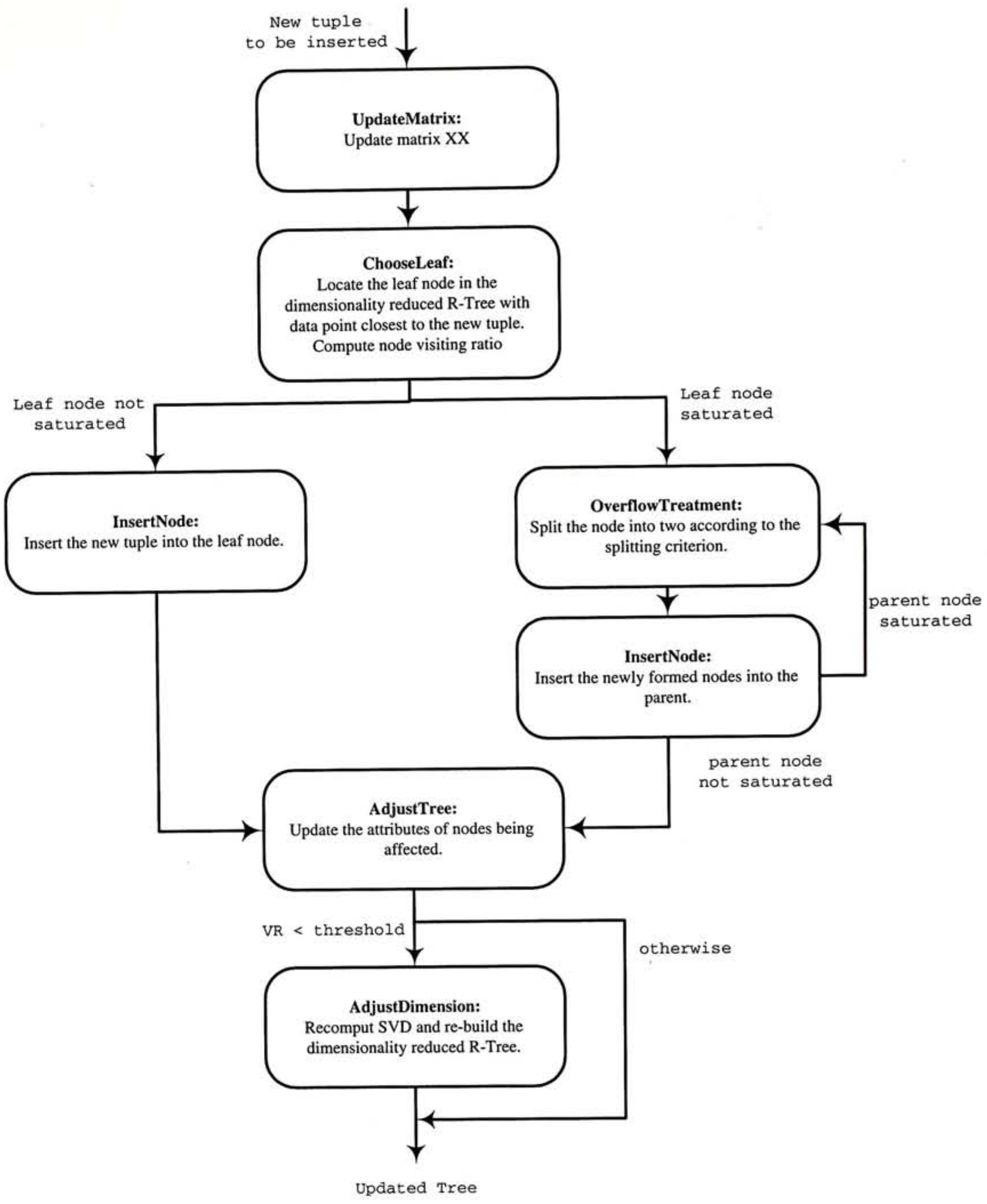

New tuple
to be inserted
UpdateMatrix:
Update matrix XX
ChooseLeaf:
Locate the leaf node in the dimensionality reduced R-Tree with data point closest to the new tuple. Compute node visiting ratio
Leaf node not saturated
Leaf node saturated
InsertNode:
Insert the new tuple into the leaf node.
OverflowTreatment:
Split the node into two according to the splitting criterion.
parent node saturated
InsertNode:
Insert the newly formed nodes into the parent.
parent node not saturated
AdjustTree:
Update the attributes of nodes being affected.
VR < threshold
otherwise
AdjustDimension:
Recomput SVD and re-build the dimensionality reduced R-Tree.
Updated Tree

Figure 4.7: Flow of insertion algorithm

works well for databases which are rather low in dimensionality. However for high dimensional database, the performance of the search will degrade significantly. Even a simple linear search will work better that using any multi-dimensional data structure. In IDAN, we will apply the idea of the dimensionality reduction techniques. It is the reason why we have to use R-Tree together with A-Tree. The dimensionality reduced R-Tree is mainly used in searching of the nearest neighbor, however the A-Tree is used to store the clustering information. We will first search for the dimensionality reduced R-Tree and locate the data point which is nearest to the new tuple. Then with the help of the links between the leaves of R-Tree and A-Tree, we can locate the leaf node of the A-Tree which the new record will be merged into it to form a cluster.

Since the R-Tree index the dimensionality reduced data points, information must be lost. When we are searching the nearest point from the R-Tree, the leaf node being found may not be the correct one in full-dimension scale. In IDAN, we first perform a k-nearest neighbor search on the R-Tree and at the same time, we count and record the number of leaf nodes being visited in the search as r_count. The next step is to compute the full dimensional distance of these neighbors. Then we perform a range query on the R-Tree with the smallest distance ξ being found. We search all the data points in the R-Tree which the distance to the query point is smaller than ξ. Again we count and record the number of leaf nodes being visited in the search as f_count. Since ξ is in full dimensional distance, we will not miss the nearest data point.

After choosing an appropriate node, the next job is to insert the new tuple into it. If the leaf is not full, the new tuple can be directly inserted to the leaf. However, if the leaf is already saturated, overflow treatment has to be done. In this stage, the overflow treatment procedure will split the saturated node into two. There are many alternatives can be done, such as re-insertion of the overflow nodes [2]. A good overflow treatment procedure can improve the performance of future insertion and the performance of query.

To split a node into two, we first compute the distances between every pairs of children in the overflowing node and store the inter-nodal distances in a $(N\_MAX + 1)$ x $(N\_MAX + 1)$ matrix. Note that $(N\_MAX + 1)$ is the total number of children in an overflowing node. After calculating all the distances, we select a pair of nodes, which are farthest apart as the seeds. Then we group the node, which is closest to the selected seeds, together with the corresponding seed to form a new node. The process will be repeated until all the children in the overflowing node have been assigned. The number of children in the newly formed node will be ranging from $[1, N\_MAX]$. After the splitting, the two newly formed nodes will be inserted to the parent recursively. The parent node may be overflow again. The overflow treatment will be carried on until no further overflow occurs.

Once the new tuple is inserted into the tree, we have to propagate the change from leaf to the root. We have to modify the *mean* and *size* attributes in the A-Tree. These attributes can be directly computed from its children. It is not necessary for us to update all the nodes in the tree. We can simply update the nodes being inserted (the new node and the nodes after splitting) and their ancestors only.

Now we have finished updating the A-Tree, but we still have to update the R-Tree as well. When a new record in inserted in the database, the SVD transformation matrix V will no longer be accurate. Although V is not accurate, it is still a good approximation to the actual V. As more and more records are inserted into the database, the accuracy of V will be degraded. This will greatly affect the performance of searching the nearest point mentioned in the previous step. We have to set up some mechanism to monitor the precision of the SVD transformation matrix V. Remember we have recorded the number of leaf nodes being visited in r_count and f_count. The visiting ratio (VR) $r\_count : f\_count$, in fact, is representing the quality of the SVD transformation matrix V. If VR is smaller than the threshold, we update the SVD transformation matrix V and adjust all the nodes in the R-Tree.

4.5 Visualizing clustering patterns

After integrated with an information visualizer, we can simply display the clustering patterns by parsing the A-Tree. We are interested in the distribution of each cluster in the domain. IDAN is supposed to provide an interactive way of visualizing the clustering patterns. The user can change the number of clusters interactively and the visualizer will response with the correct clustering immediately.

```
procedure MergeCluster(var NodeType  list[],int n)
begin
  var list: array of NodeType;
  level :=0;
  count := 0;
  /* find the level in the tree which the total number of nodes > n */
  while (count < n) do {
     count := CountCluster(root,level);
     level := level + 1;
  }
  /* store the nodes in the level in an array list */
  list := BuildList(root,level);
  while (count > n ) do {
     /* compute internodal-dist in list */
     InterNodalDistance(list,count,inte_dist);
     /* find the pair of clusters which is nearext to each other */
     for(x:=0,y:=1,i:=0;i<count;i++) {
       for(j:=i+1;j<count;j++) {
         if (inter_dist[x][y] > inter_dist[i][j]){
           x := i; y := j;
         }
       }
     }
     list[x] := MergeNode(list[x],list[y]);
     for(i=y;i<count;i++)
       list[i] := list[i+1];
  }
end;
```

Figure 4.8: Procedure for MergeCluster

As mentioned in the previous section, we state that each node in the A-Tree forms a cluster. However the number of nodes in the A-Tree may not be matching with the user's defined number of clusters. To archive our goal, the merging process has to be performed. The merging process is simple. Firstly, we scan the A-Tree level by level until the number of nodes in the same level just exceeds the number of clusters to be found, which is specified by the user. Then we use a list to store all the nodes in that level. At this moment, the merging process can be started. Suppose the number of nodes in the list is K, we first compute the inter-nodal distances between every nodes in the list and store them in a square matrix of K x K. Then we merge the nodes that are closest to each other to form a new node. Now the number of nodes in the list is reduced to $K-1$. After that the inter-nodal distances are computed again and the merging process is repeated until the total number of nodes in the list is just equal to the number of clusters to be found. The details of the procedure is shown in the figure 4.8.

After the merging process, we have found k clusters and the information of each cluster is stored in a node. The next thing is to display the tuples within the same cluster in the visualizer. The thing we have to do is simply to explore the tree rooted at each node being found and to display them is different colors. Figure 4.9 shows an example of datasets being investigated. This dataset is obtained from the homepage http://www.bell-labs.com/project/serendip. The result of the clustering pattern is similar to the one computed by CURE [13], but IDAN can support interactive change in the number of clusters to be discovered. At the same time, IDAN does not require any other parameters while CURE is too sensitive to many parameters. We will discuss the comparison of IDAN with the other successful clustering algorithms in the coming chapter. The results of different number of clusters are demonstrated in figure 4.10 – 4.12.

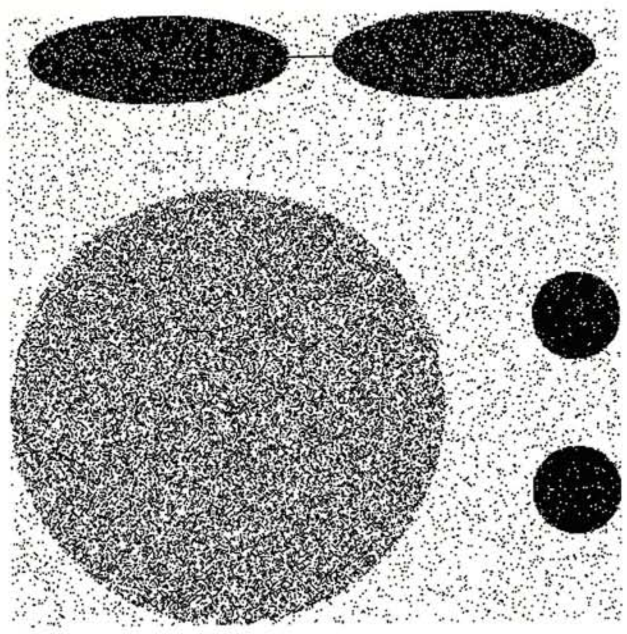


Figure 4.9: Example of dataset

4.6 Comparison

Among the existing clustering algorithms, IDAN is the most similar to BIRCH. Both of the algorithms are using tree structures. IDAN uses A-Tree while BIRCH uses CF-Tree. However IDAN does not suffer from the problem of BIRCH - *mis-clustering*. The reason for BIRCH to mis-cluster the dataset is that BIRCH uses the *centroid* (or *mean*) to represent the whole cluster. It will discard the information like sizes and geometry of the clusters. However in IDAN, we are not using centroid to represent the cluster. Instead, we use an A-Tree node to represent a cluster. All the data points within the same cluster will fall within the same subtree. At the same time, MBR of the node will describe the size and geometry of the cluster. As a result, IDAN can get rid of the problem of mis-clustering. For BIRCH, it uses the minimal bounding sphere to hold the data points within a cluster. However minimal bounding sphere in fact does not represent too much useful geometrical information. At the same time, it requires every data points within the minimal bounding sphere to form a single cluster. Therefore, BIRCH can be only suitable for clustering database which clusters are in the shape of hyper-sphere. Reminded that IDAN is requiring data within the same cluster to fall inside the same sub-tree (MBR), in stead of requiring all data within the MBR of a node to form a cluster. So IDAN is not suffering from same problems as BIRCH, which can only cluster spherical clusters.

For CURE, it is also capable of discovering clusters with different sizes and geometry. However, the algorithm of CURE will be very dependent on the sampling phase. If the qualities of samples are not good, it will greatly affect the clusters being found. However, IDAN does not require sampling to improve the performance since itself is an incremental algorithm, it makes use of the result previously found so that it can achieve high performance. CURE is a good clustering algorithm in the sense that it does not suffer from the problem of mis-clustering. The reason is that CURE tries to shift the data point towards the mean by shrinking the distance between the point and the mean by a fraction of α and it uses multiple representatives to denote a cluster instead of using a single point like centroid or medoid. However, the disadvantage of this mechanism is that the clustering quality is very dependent on the choice of α. It is very difficult for the user to determine the correct α in order to give good clustering result. Besides, the number of representatives is very critical too. If the number of representatives is set to be too large, it will affect the performance of the algorithm. If the number of representatives is too small, the information on cluster size and the geometry will be discarded. It is very difficult to determine the number representatives so that to give good quality of clustering. For IDAN, there is a few number of parameters and all of them will affect the performance of the algorithm only and does not influence the result.

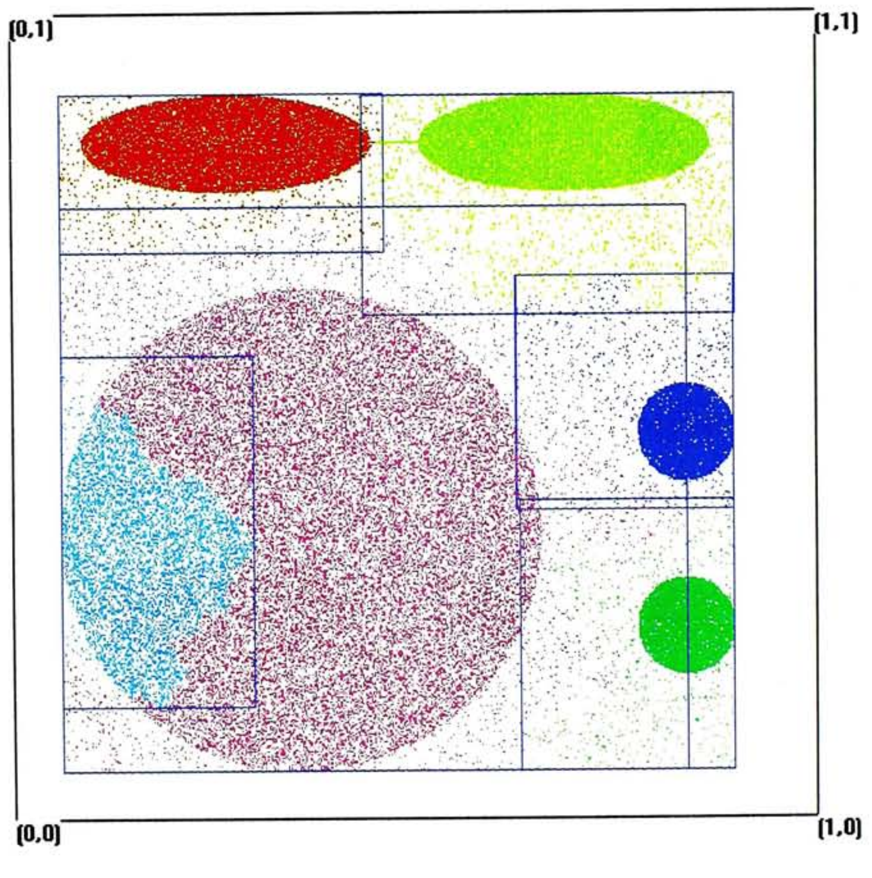


Figure 4.12: Example on clustering (Number of clusters = 6)

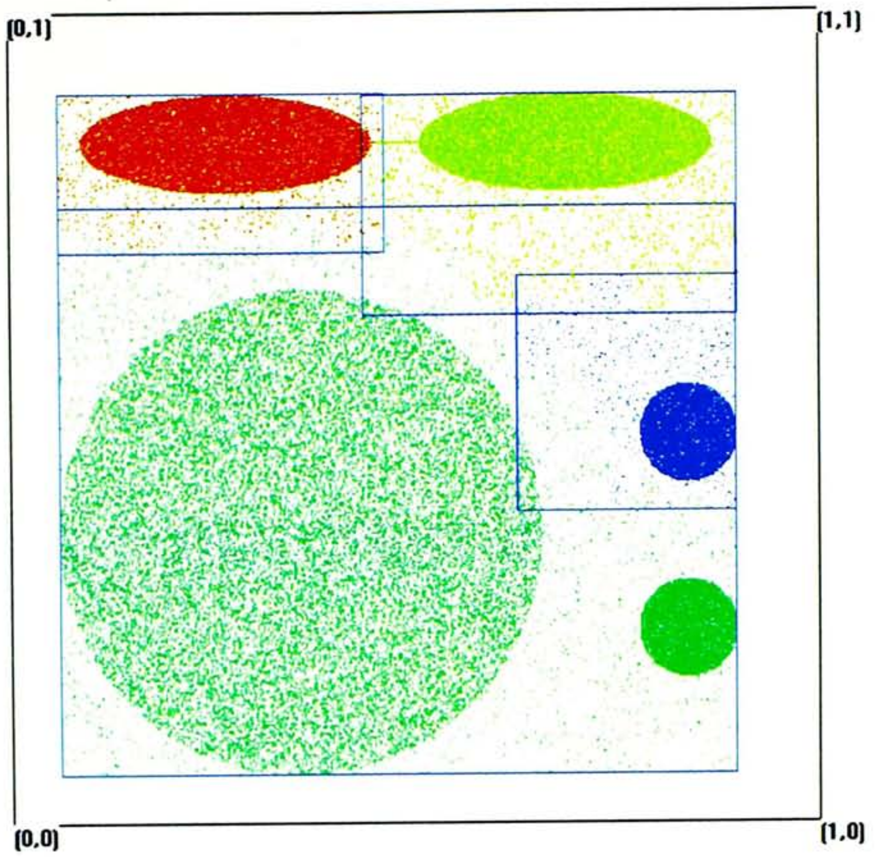


Figure 4.10: Example on clustering (Number of clusters = 4)

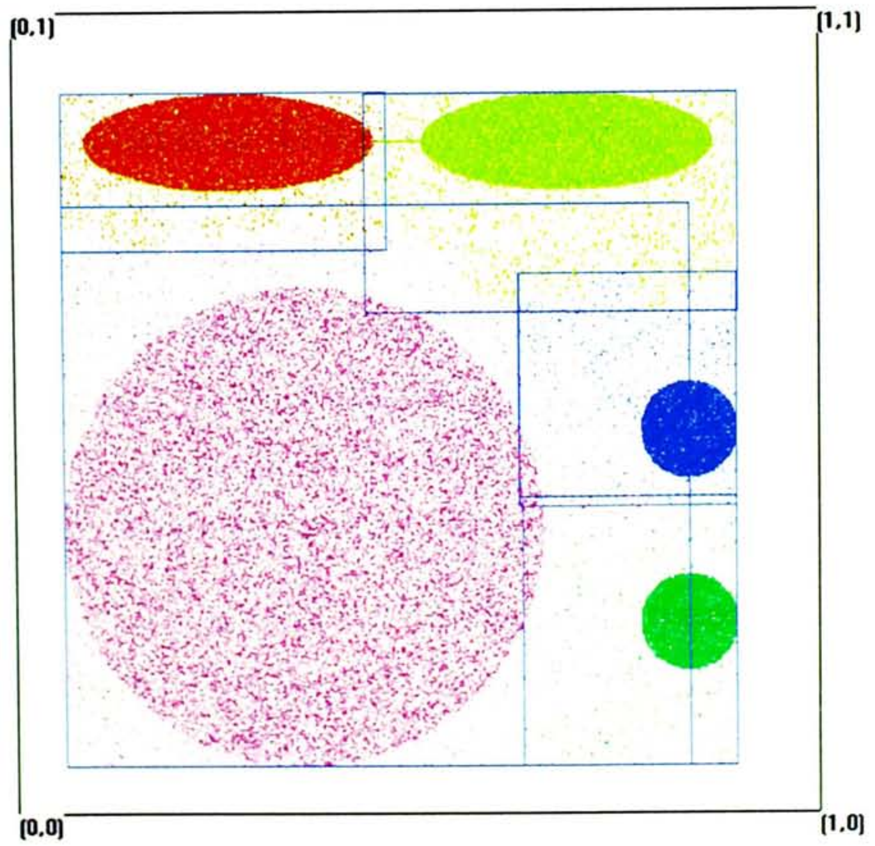


Figure 4.11: Example on clustering (Number of clusters = 5)

Chapter 5

Performance Studies

Up to now, we have described the new algorithm being proposed. In this chapter, we would like to evaluate the performance of IDAN. We will address the result we found in solving the problem of mining quantitative association rules. Then we will talk about the performance of IDAN in solving the clustering problem.

5.1 Association Rules

We implemented the algorithm IDAN in C++. We evaluated the performance on a Ultra Spar 1 machine with 686MB of main memory running Solaris 2.5. As mentioned in chapter 3, the visualization stage involves very little calculations and disk accesses, they are negligible. We are focusing on the first stage: A-tree building stage since most of the computation is taken place in this stage.

First of all, different dimensional data have been tested. We have worked on 3, 5, 10 and 20 dimensional data. For each dimension, we have tried to insert 50,000 records of a synthetic dataset into an A-tree. We recorded the insertion time of each record and compared the average insertion time and the size of dataset. The result is being shown in figure 5.1.

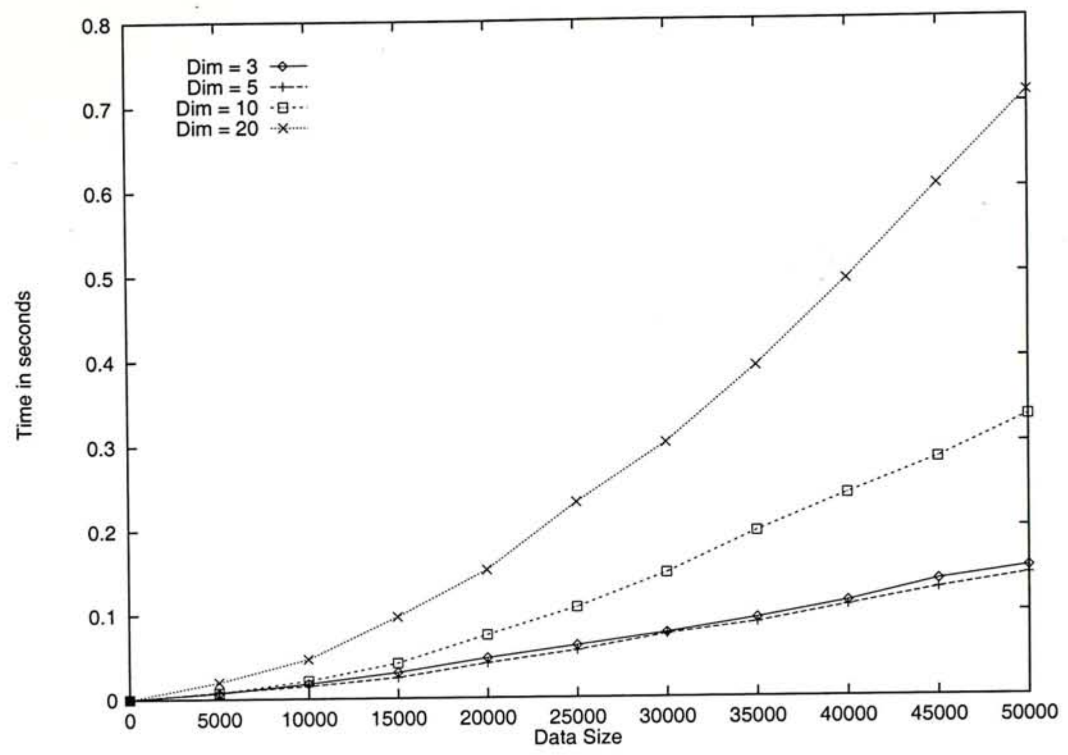


Figure 5.1: Average insertion time

We expect the performance of the algorithm will decrease linearly as the size of dataset is increasing. The result is confirmed and is shown in figure 5.1. For each of the number of dimension being tested in the experiments, all of them behave quite linearly for all the datasets. At the same time, we expect when the number of dimensions increases, the performance will be slow down. From the experiments done, we find that for small dimensional datasets, the performance will not be affected very much. We found that the performance of 3-dimensional dataset and 5-dimensional dataset is almost the same. However for high-dimensional datasets, the performance will be affected quite obviously. In the experiment of 20-dimensional dataset, we found that it halved the performance of the performance in 10-dimensional dataset. The reason is mainly due to the increase of node size and the increase of numeric calculation for high dimensional data. As the number of dimensions increases, the physical size of the node increases directly. So the page size cannot hold as much nodes as before. It will increase the number of disk access and will greatly affect the performance.

After that we started another series of experiments. This time we are interested on how the number of antecedents affect the performance of IDAN. We fixed the total dimensions of the datasets at 10. Then we vary the number of antecedents and see whether the performance will be affected. We have worked on 3, 5, 7 and 9 antecedents. The result on average insertion time against the size of dataset is plotted in the figure 5.2

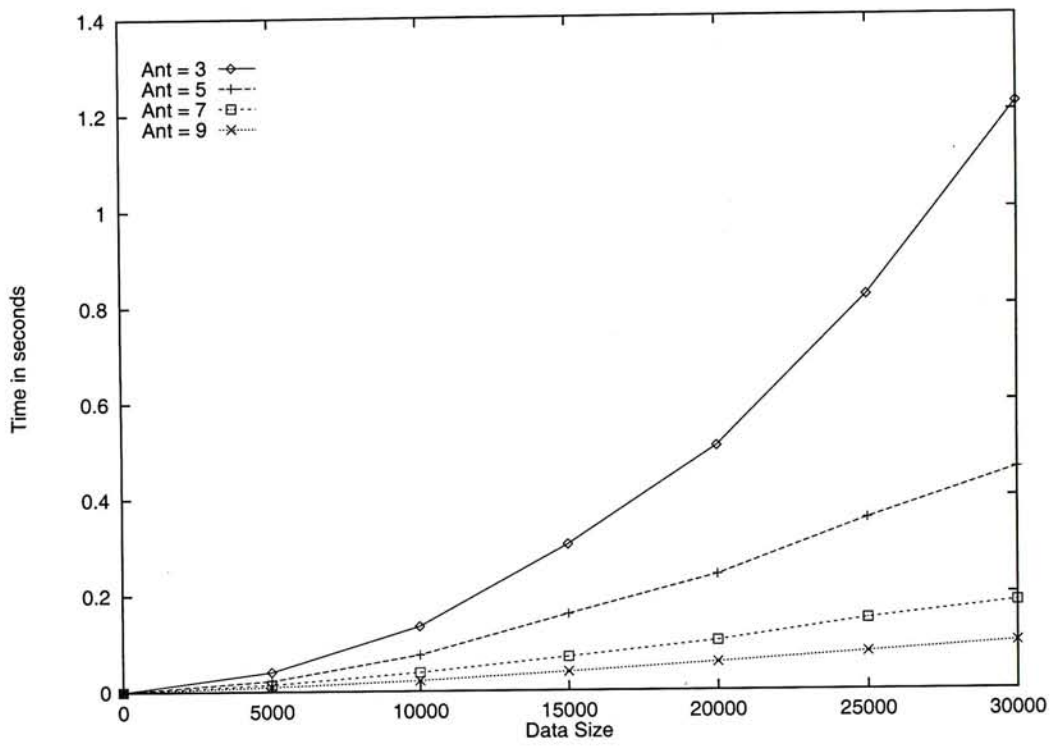


Figure 5.2: Dimension of antecedent

From the result of the experiments, we found that the performance with large antecedent will be better. The average insertion time of antecedent of 3 is double the one for antecedent of 5. The gain in performance of increase in antecedent is almost linear. This is reasonable since we are discovering rules from more dimensions of antecedent part. This means that we are mining rules from more information given. As a result, the computation time should be less than the others with smaller number of antecedents.

5.2 Clustering

The proposed algorithm has been written in C++. We perform the evaluation on a Ultra Sparc 5 machine with 192MB of main memory running Solaris 2.5. Again we are focusing on the first stage: A-tree building stage, since most of the computation is taken place in this stage and the computation cost in visualization stage is negligible.

We have done several series of experiments to test the performance of IDAN. First of all, we would like to investigate how dimensionality reduction technique help us on clustering high dimensional database. We prepared datasets of 50,000 data points with number of dimensions varying from 5 to 20. We recorded the average insertion time of each record and did the experiments on both with and without dimensionality reduction transform. The result is plotted in figure 5.2.

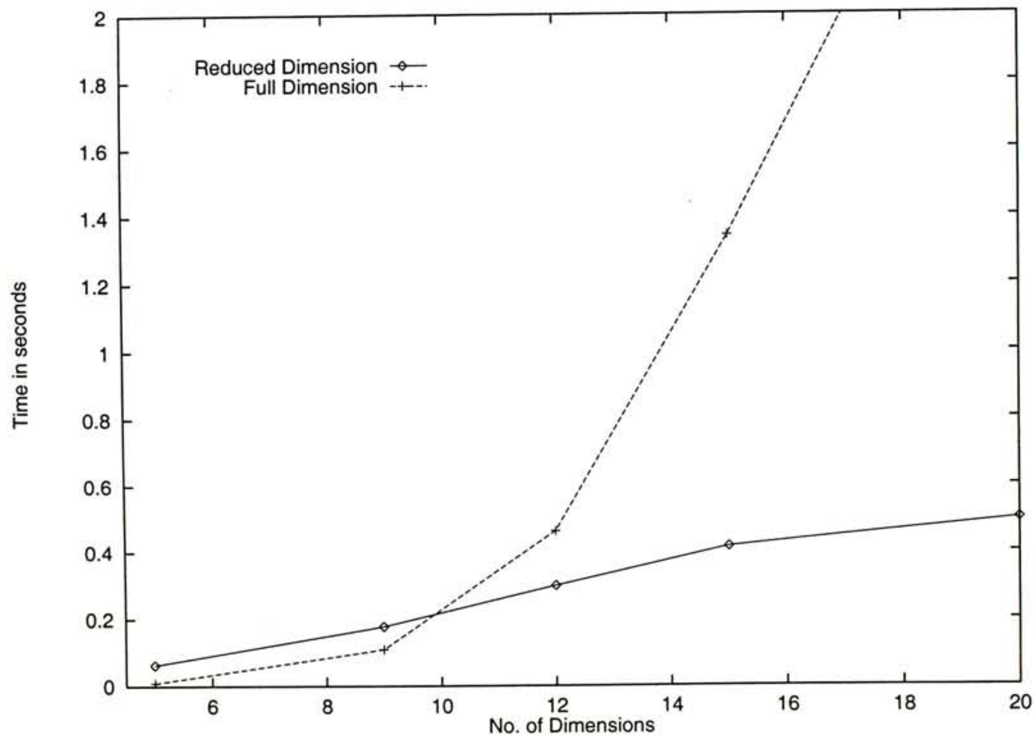


Figure 5.3: Performance with and without dimensionality reduction transform.

From the figure we found that without dimensionality reduction, IDAN works better for datasets with dimensionality below 10. It is reasonable since the gain

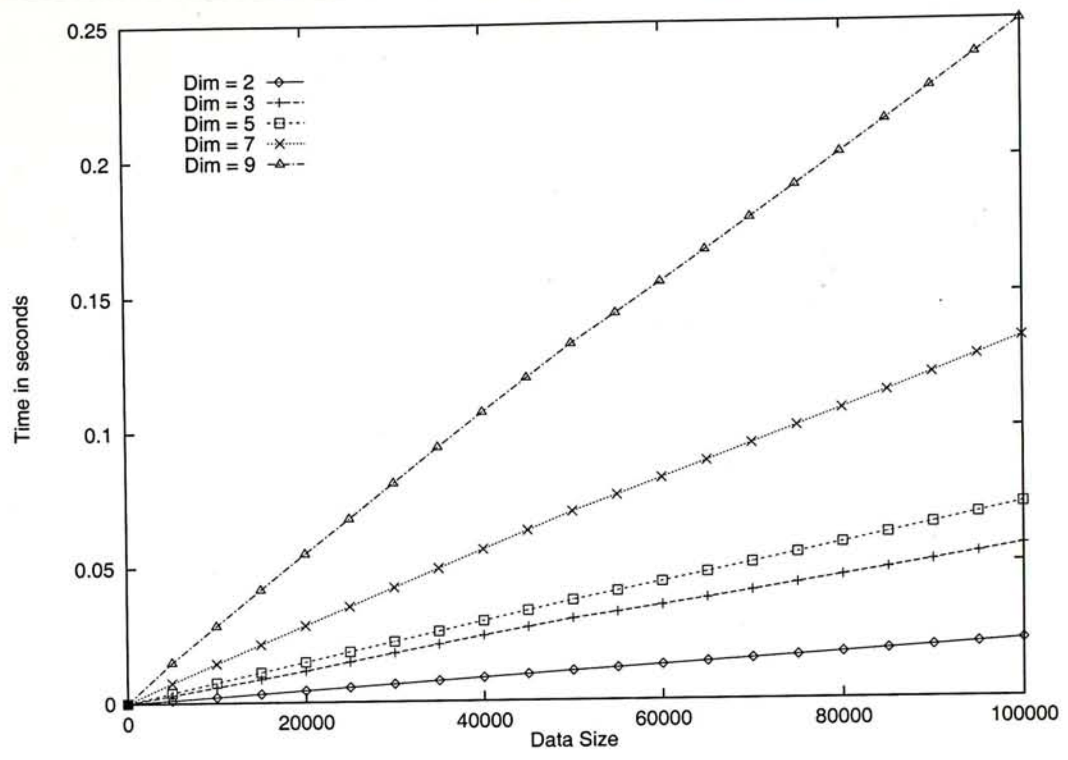


Figure 5.4: Average insertion time

in performance of searching along the R-Tree cannot overcome the the cost on computation for dimension reduction on low-dimensional data. However as the number of dimensions increases, the performance degrades sharply and becomes unacceptable. If we apply dimensionality reduction technique on IDAN, we find that the insertion time of new record increases gently as the number of dimensions increases. The result shows that with the help of dimensionality reduction techniques, IDAN can cluster high-dimensional databases very efficiently.

Next, we try to see how IDAN work on low dimensional datasets. We have worked on 2, 3, 5, 7 and 9 dimensional data. For each dimension, we have tried to insert 100,000 records of a synthetic dataset into an A-tree. We recorded the insertion time of each record and compared the average insertion time and the size of dataset. The result is shown in figure 5.2.

We expect that the performance of the algorithm will decrease linearly as the size of dataset increases. This is confirmed from the result shown in figure 5.2. For each of the number of dimension being tested in the experiments, all of them behave quite linearly for all the datasets. At the same time, we expect that as the number of dimensions increases, the performance will be slow down. From the experiments done, we find that for small dimensional datasets, the performance will not be affected very much. We found that the performance of 3-dimensional dataset and 5-dimensional dataset is almost the same. However for high-dimensional datasets, the performance will be affected quite obviously. In the experiment of 9-dimensional dataset, we found that it halved the performance of the performance in 5-dimensional dataset. The reason is mainly due to the increase of node size and the increase of numeric calculation for high dimensional data. As the number of dimensions increases, the physical size of the node increases directly. So the page size cannot hold as much nodes as before. It will increase the number of disk access and will greatly affect the performance.

After that we started another series of experiments. This time we are interested in how the number of clusters in the dataset affects the performance of IDAN. We fixed the total dimensions of the datasets at 2. Then we vary the number of clusters and see whether the performance will be affected. We have synthesized different datasets with different number of clusters for the experiments. We have worked on datasets with 2, 4, 5, 8 and 10 clusters and each of them is consisting of 50,000 data tuples. The result on average insertion time against the size of dataset is plotted in figure 5.2. From the result of the experiments, we found that the performance of the algorithm is not affected by the data distribution. We can observe from figure 5.2 that almost all the curves lie nearly to each others. Some of the hierarchical clustering algorithms are very sensitive on the distribution of the datasets. Some distribution of data points will have very poor performance. From this set of experiments we can conclude that IDAN is immune from the distribution of datasets.

Finally, we have worked on a set of experiments, which is aimed to investigate the effect of N_MAX on the performance of the algorithm. We synthesized a

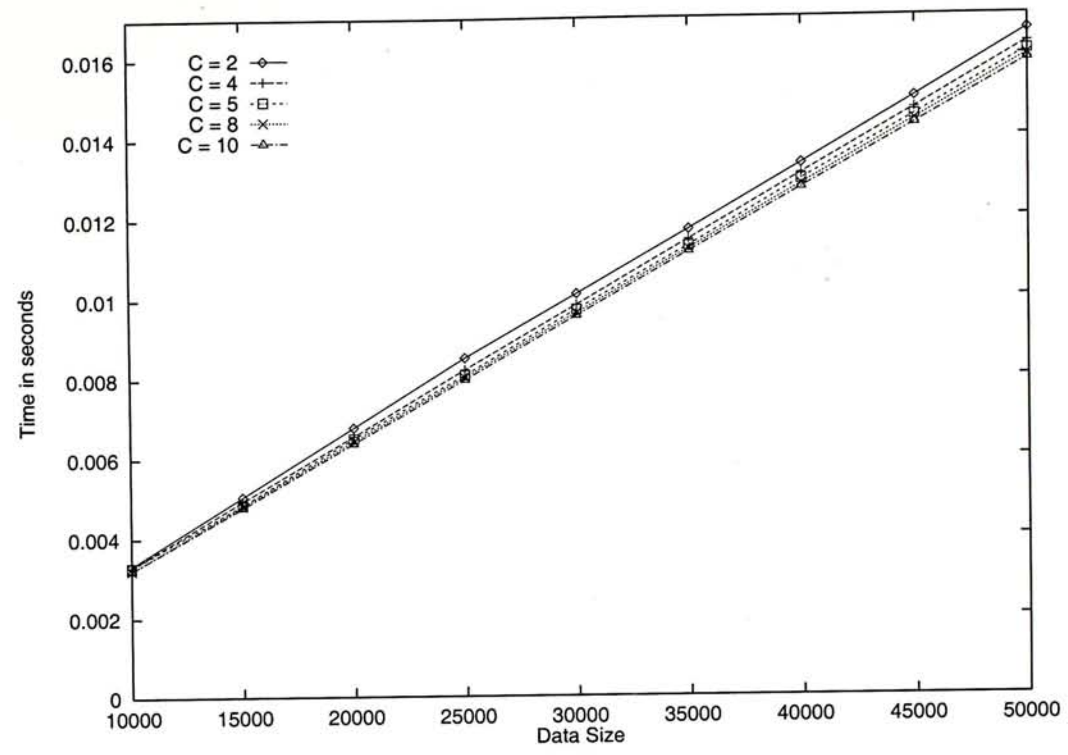


Figure 5.5: Effect on different number of clusters

dataset of dimension 2 with 50,000 records. Then we use different values of N_MAX and use our algorithm to cluster the same dataset. We have tried to use 5, 10 , 15, 20 and 25 as examples of N_MAX to see whether the performance of the algorithm will be affected. The result is plotted in figure 5.2. We expect the performance of the algorithm will increase as the value of N_MAX increases. As mentioned in the previous chapters, insertion in IDAN requires parsing the tree from the root to a leaf once only. So the performance greatly depends on the height of the tree. If we set the parameter of N_MAX to a smaller value, the tree will be "thinner" and "taller". We have to spend more time to parse the tree. On the other hand, if we set a larger value of N_MAX, the tree will be flat and as a result, the height of the tree will be reduced. This will improve the computation time on parsing the tree from the root to a leaf. From figure 5.2, we can figure out that the performance of small N_MAX will be affected more than that of large N_MAX. The curve for $N\_MAX = 5$ is higher than the others in a large amount. However, from the curves of $N\_MAX = 15$, 20 and

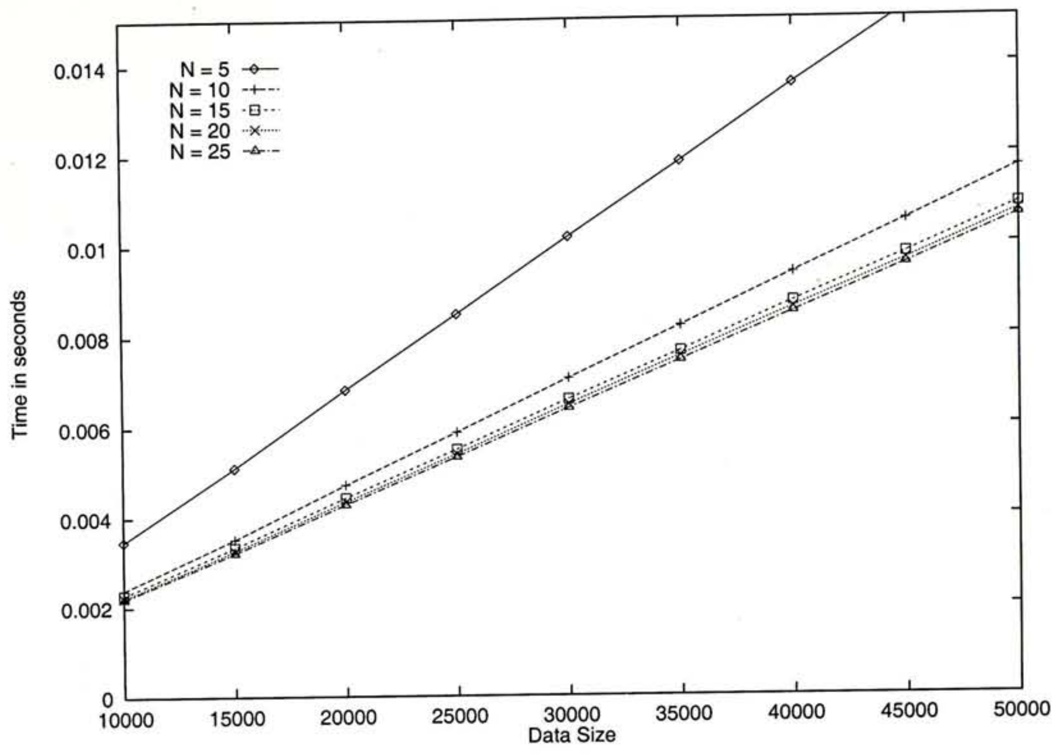


Figure 5.6: Scalability wrt. increasing N_MAX

25, we find that the performance are very similar tp each other. The reason is that the height of the tree is $O(log_{N\_MAX}(n))$, so the height of the tree will not reduce too much for large N_MAX.

Chapter 6

Survey on data visualization techniques

Due to great improvement in data collection techniques, now people are much more easy to obtain useful data. The amount of data is increasing in an unbelievable rate. For example, satellites going around the Earth are transmitting large amount of data all the time. The amount of data is measured in the unit of gigabyte per hour. Traditional database management system, which are usually text based, are capable to store the data and most of them work very well in data retrieval. However, it is very difficult for the user to explore the data in the traditional database management systems.

Data visualization techniques is a right solution to the problem. The basic idea of data visualization techniques is to represent as many data as possible on the screen by mapping every data records to some graphical representations which users are able to have an overview on the dataset.

One of the most important visualization techniques is scatter-plot. Each data value is mapped to a single point in the visualization domain. As we are living in 3-dimensional space, scatter-plot is a perfect way to represent data visually for data with dimensionality below 3. However, for multidimensional data, scatter-plot cannot provide good result. In this chapter, we would like to introduce some important visualization techniques for multidimensional data. We try to classify the techniques into 4 groups: Geometric Projection, Icon-based, Pixel-oriented and Hierarchical techniques. In fact, there are many other techniques, but due to limited time and space, we just address the most common and important ones here.

6.1 Geometric Projection Techniques

The key idea of geometric projection techniques is to construct interesting projections on multidimensional data. To apply the techniques, we have to find out the interesting projections first. There are many ways such as principal component analysis, factor analysis and multidimensional scaling, which is aimed to find out these projections. We classify these methods as the field of projection pursuit [15].

6.1.1 Scatter-plot Matrix

The first technique we introduce here is scatter-plot matrices technique [6]. A scatter-plot matrix consists of a number of 2-dimensional scatter-plots with different projections of the multidimensional dataset. Suppose the original dataset is in k-dimensional. The scatter-plot matrix will consist of $k^2/2 - k$ different scatter-plots. The advantage of the scatter-plot matrix is human readability since people are very familiar with 2-D scatter-plots. On the other hand, scatter-plot-matrix cannot indicate the relationship more than 2 attributes since each scatter-plot in the matrix is in 2-dimensional space only.

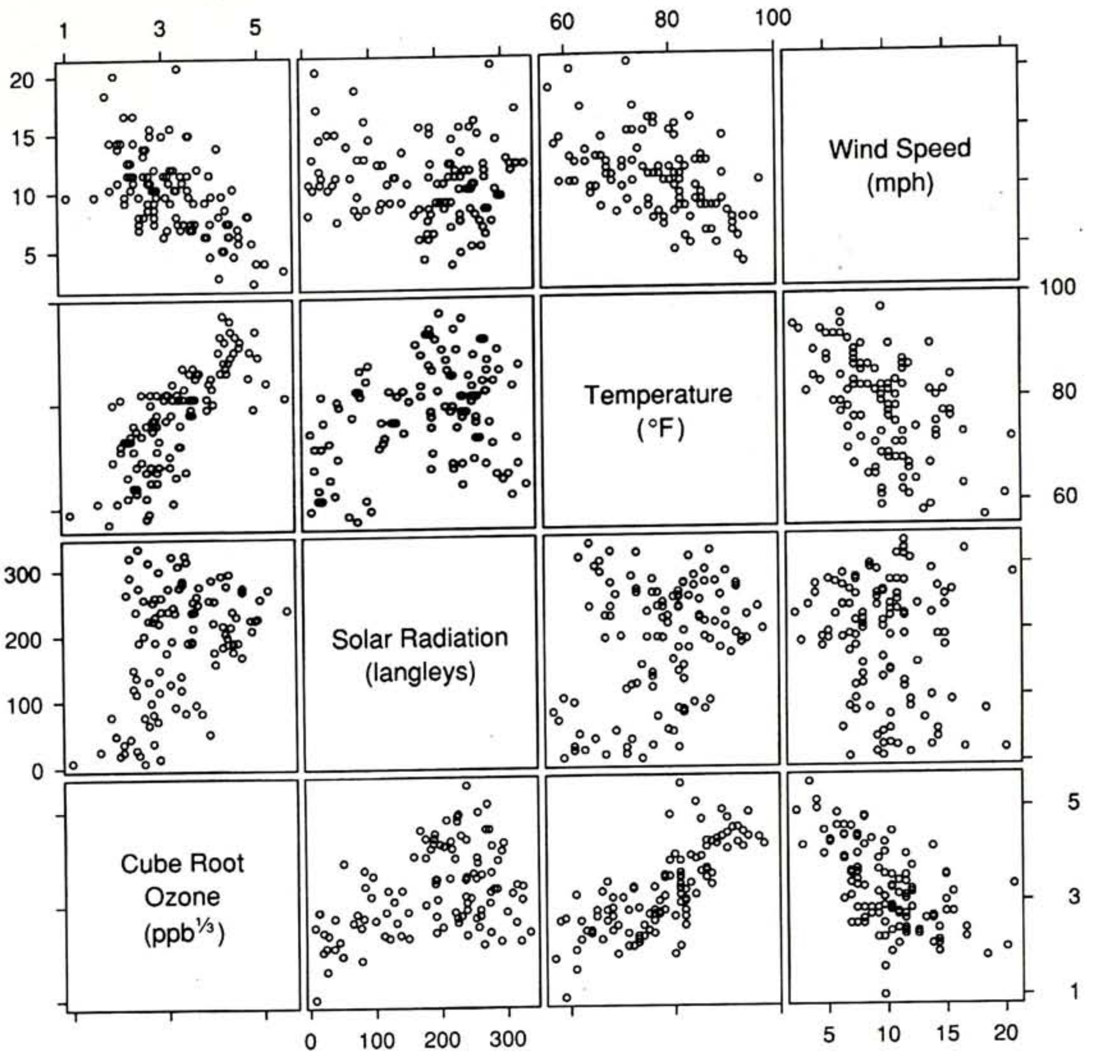


Figure 6.1: A scatter-plot matrix graphs hypervariate data: measurements of solar radiation, temperature, wind speed, and cube root ozone concentration on 111 days at different places in the New York metropolitan region

6.1.2 Parallel Coordinates

Another geometric projection technique is the parallel coordinates technique [16, 17]. Parallel coordinates technique tries to transform the k-dimensional data records into 2-dimensional visual space. There are k equidistant axes in the visual space and these axes are arranged to be parallel to each other. Each axe is corresponding to one of the dimension in the dataset and the axe is linearly scaled from the minimum to the maximum value of corresponding dimension.

Unlike scatter-plot matrix, data values are mapped into polygonal lines in-

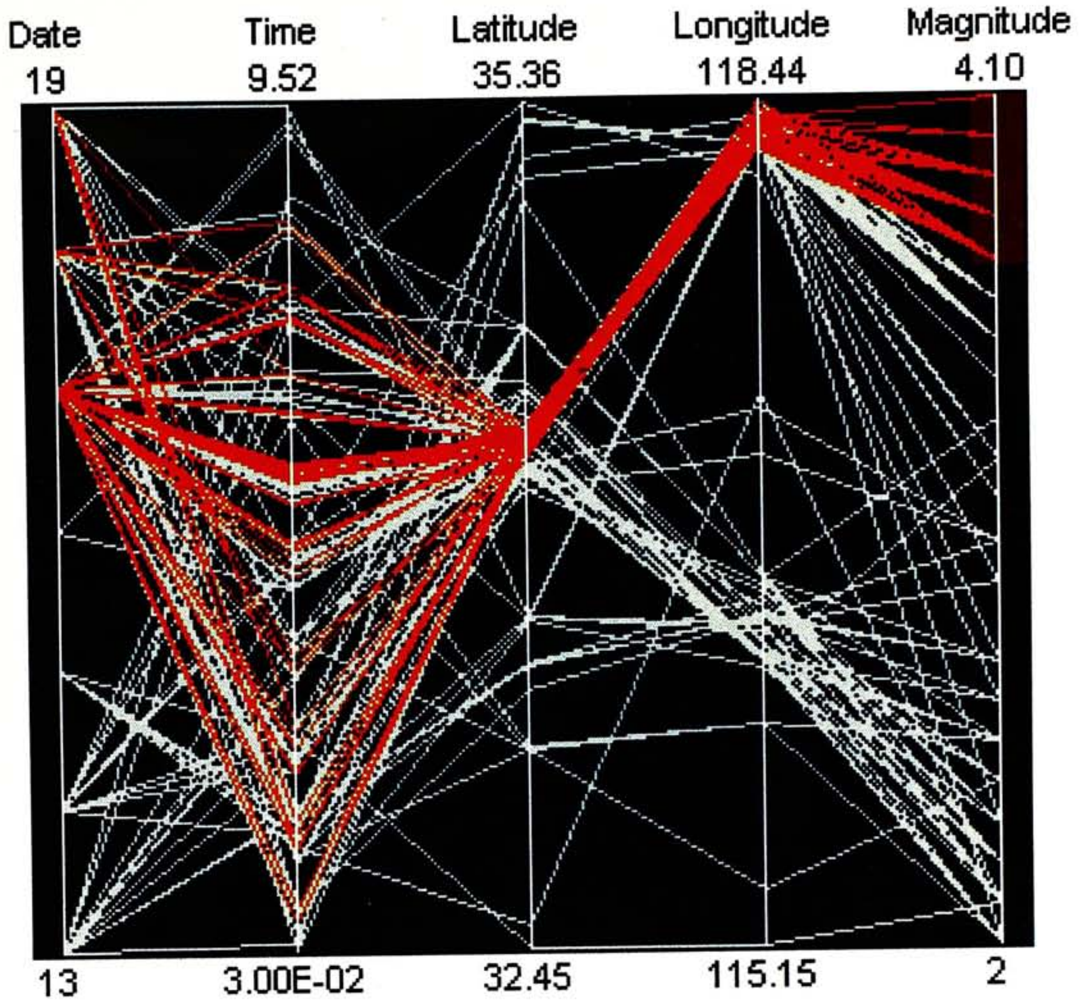


Figure 6.2: Example of parallel coordinates: This plot reveals that high-magnitude earthquakes (highlighted in red) occurred at the same longitude and latitude and on three particular days.

stead of points. The polygonal lines are intersecting each of the axes at the right place which is corresponding to the value of the data in that dimension. The advantage of the parallel coordinates techniques is that it can give us a wide range of characteristics such as the distribution of the data and the functional dependencies. However, since the polygonal lines may overlap to each other, so it is very difficult for people to read the graph for large dataset. Sometimes, we will use different color to represent each polygonal line and it can increase the readability of the graph.

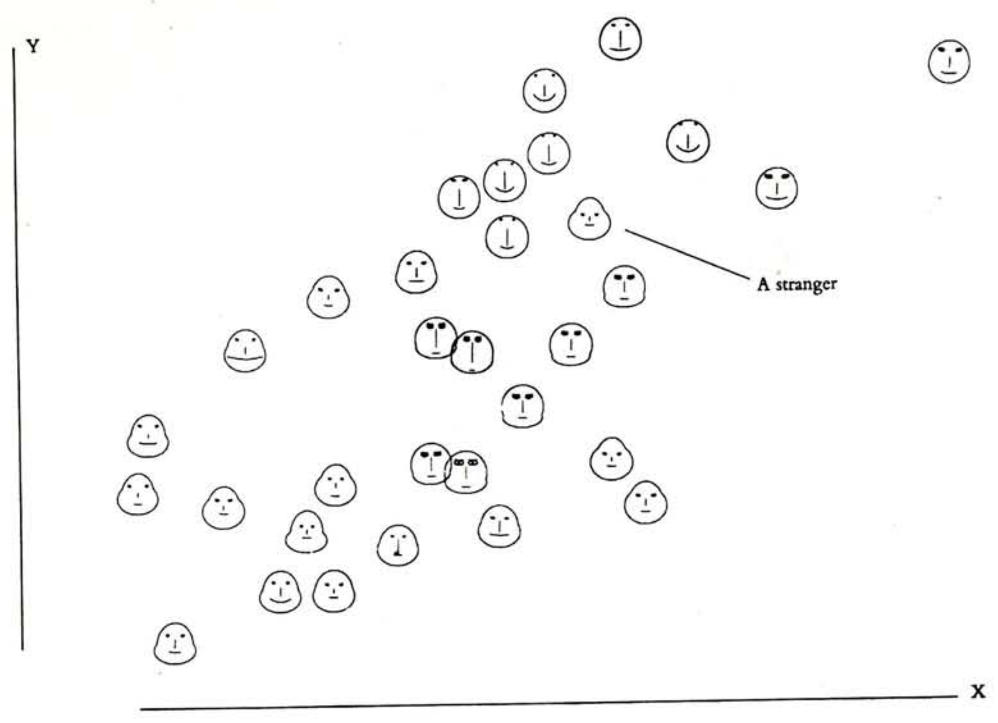


Figure 6.3: Example of Chernoff face representation

6.2 Icon-based Techniques

Another class of visualization techniques are the icon-based-techniques. Some people call them as iconic display techniques. The key idea of these techniques is to construct different icons to represent data records of different values.

6.2.1 Chernoff Face

Chernoff face [5] visualization technique is the most well known one among all the icon-based techniques. Suppose the dimensionality of the dataset is k, which $k > 2$. Two of the dimension of the dataset is mapped into the X and Y axes. The remaining dimensions are mapped to the properties of different parts on the face such as the appearance of the face, ears, eyes and mouth.

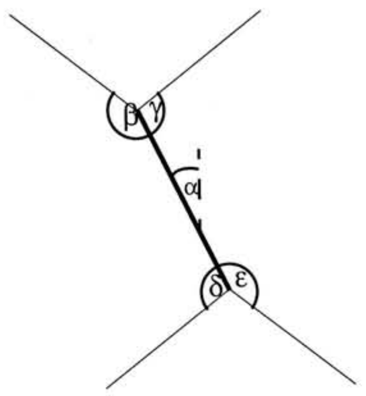
Chernoff face visualization uses different appearances of human faces to represent different data values. It is an intelligent way since people are very sensitive to the appearance of human faces and facial features. By using this technique, we can easily find out data which are in the same cluster. Outlying faces and those inconsistent with others in the neighbor are treated as 'strangers' and can be eliminated immediately. On the other hand, Chernoff face cannot work very well in large databases since the space that the faces occupying is quite large and it is very difficult to visualize a large number of data items. Besides, the number of the dimensions that Chernoff face technique can handle is quite limited

6.2.2 Stick Figures

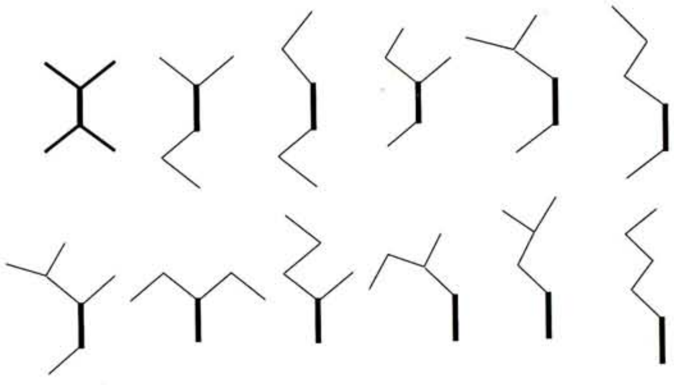
Stick figures [12] is another icon-based visualization technique. Again, two of the dimensions of the dataset are represented in the X and Y axes. For the other dimensions, this time we use different shape of the 'sticks' to represent the different data values instead of using the appearance of the facial features. Two of the attributes of the data are mapped to the display axes and the remaining attributes are mapped to the angle and/or length of the limbs.

For example, if the dimensionality of the dataset is 7, we map two of the dimensions of the data to the display axes. Then the remaining 5 dimensions of the data are mapped to the angles of the limbs of a 5 segments stick. By observing the texture patterns in the visualization, we can find out the data characteristics.

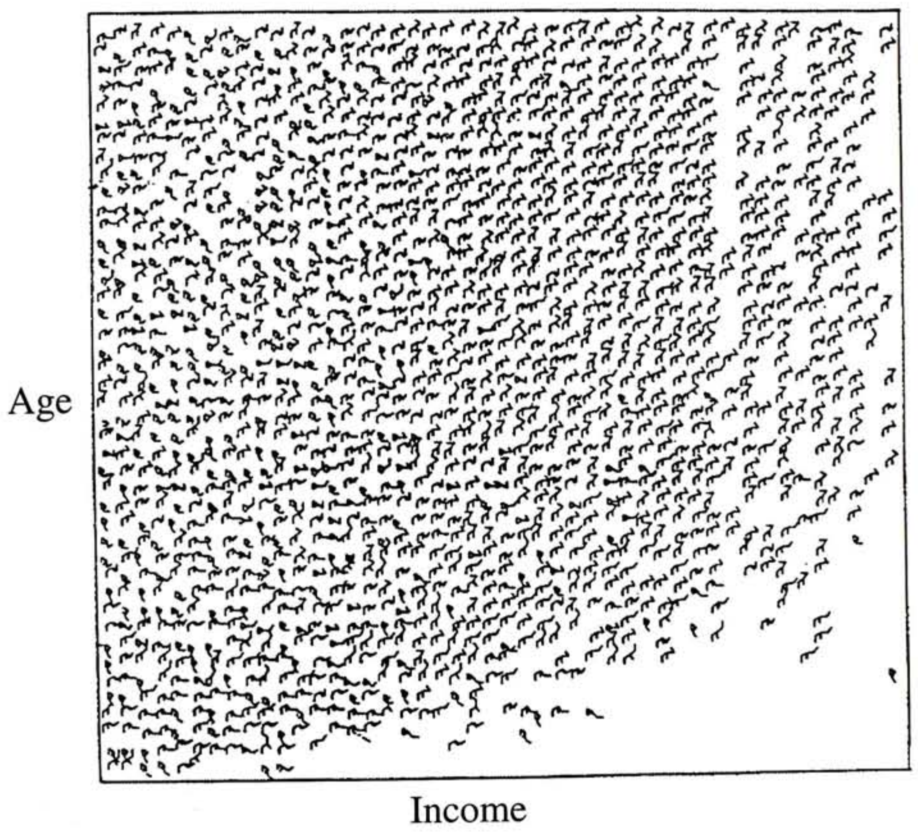
Stick figures can represent more data items when compare with Chernoff face technique. Again the number of the dimensions represented, cannot be too large since people cannot observe the difference between sticks if the number of segments is too large.



(a) Stick figure icon



(b) Family of stick figures



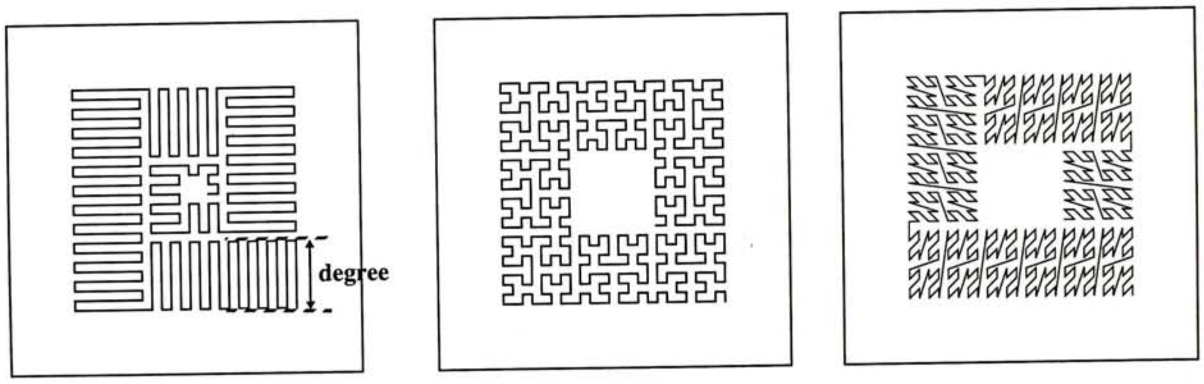
(c) Example of stick figures representation: census data showing age, income, sex, education, etc.

Figure 6.4: Stick figures visualization technique

6.3 Pixel-oriented Techniques

Pixel-oriented techniques [21, 22]use different sub-windows to represent different attributes of the dataset. The basic idea of the techniques is to map each data item to a single pixel in each of the sub-windows. The advantage of using this kind of techniques is that we can visualize a large amount of data since each data only requires a single pixel to represent itself. Data are at first sorted according to some attributes and then we fill the windows pixel by pixel.

There are many variations of pixel-oriented techniques. We can obtain different results by using different arrangements of pixels. Line by line, snake spiral, Peano Hilbert spiral and Morton spiral are some of the most commonly used arrangement.



(a) snake spiral (b) Peano Hilbert spiral (c) Morton spiral

Figure 6.5: General spiral arrangement

Line by line arrangement, in general, cannot produce the useful visual patterns. The reason is that the way we look at the visual patterns is not line by line. We usually concentrate on a particular area of the pattern. So it is better for us to arrange items which are close together in the area nearby. So spiral arrangement like snake, Peano Hilbert and Morton can produce much useful patterns.

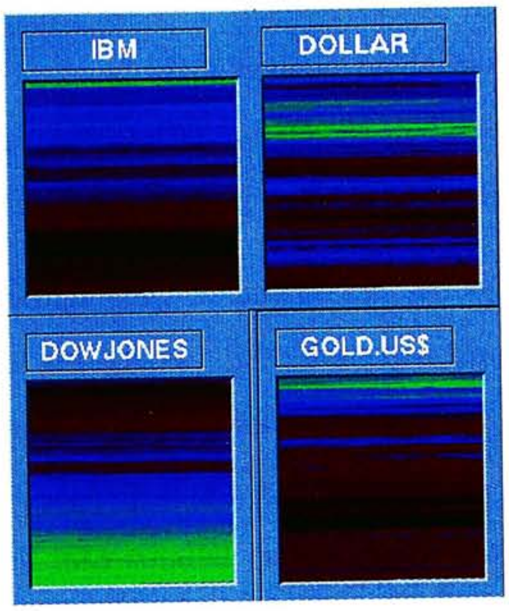


Figure 6.6: Example of pixel-oriented techniques: The graphs show datasets of time series of financial data. The graphs are arranged line-by-line

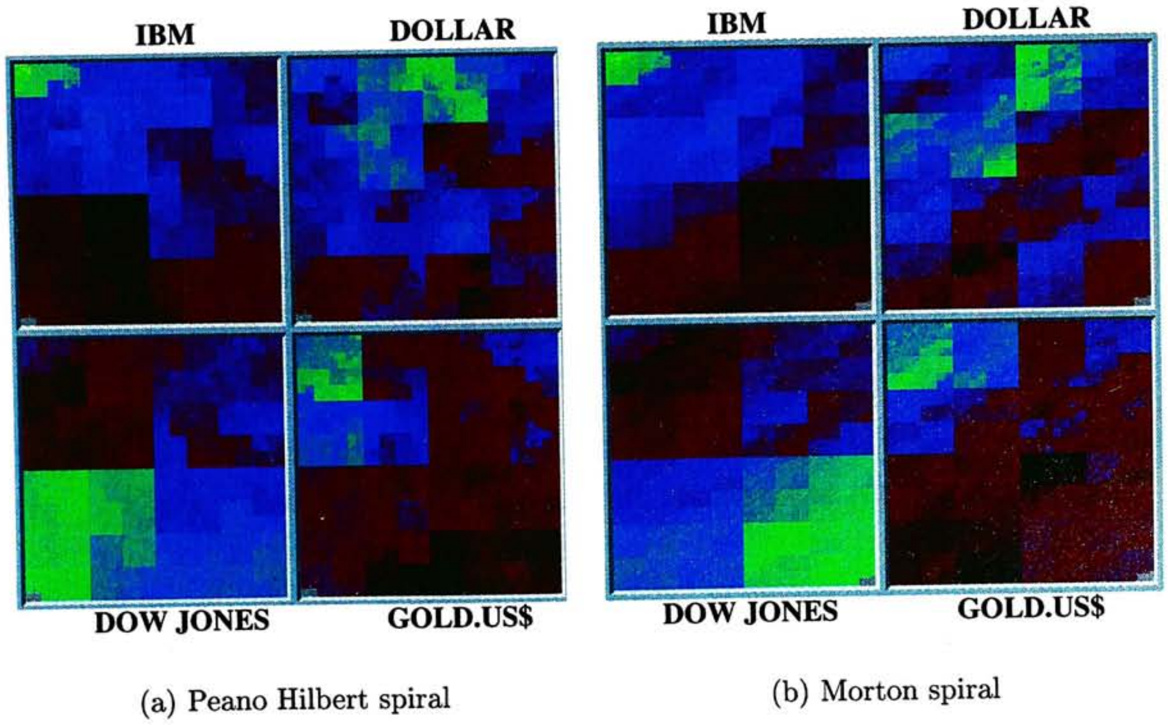


Figure 6.7: Examples of different arrangement of time series of financial data

6.4 Hierarchical Techniques

The key concept of hierarchical visualization techniques is to subdivide the k-dimensional space and represent the subspaces in hierarchical fashion. Dimensional stacking is one of the well-known techniques belonging to this class.

For dimensional stacking [24], we subdivide the k-dimensional space into a number of 2-dimensional subspaces. The first two dimensions span the X and Y axes. We obtain a number of grid cells by dividing the domain along X and Y axes into a number of equally sized intervals. Then for each of the grid cell, we further divide the cell into smaller cells along X and Y axes. This time the X and Y axes with the cell represent the other two dimensions. The dividing process will be carried on until all the k dimensions have been embedded. The advantage of hierarchical visualization techniques is that they can handle high dimensional datasets.

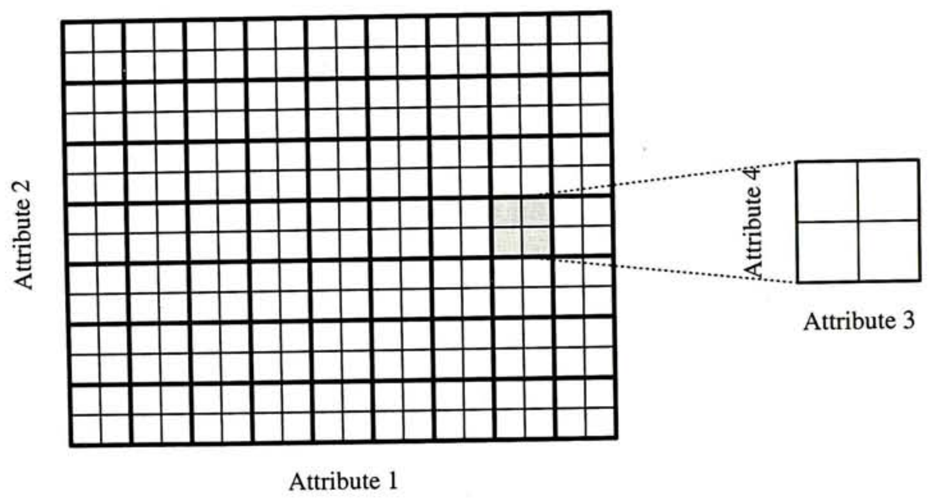


Figure 6.8: Principle of dimensional stacking representation

Chapter 7

Conclusion

In this paper, we have addressed the problems of traditional algorithms on discovering clustering patterns in large databases. The traditional algorithms are suffering from: (i) lack of user interaction and exploration, (ii) problem of mis-clustering. The whole process is just like running in a black box and the user takes no interaction except setting up the parameters for the algorithms. Once the parameter has been changed, the whole process has to be repeated from the very beginning. User is not able to preview the distribution of data so that the user cannot set up the most appropriate parameters for the algorithms. We proposed a new algorithm called IDAN, which is aimed at providing fast and user friendly way to discover knowledge from numerical data. The followings are the features of IDAN:

Hierarchical Algorithm IDAN makes use of a hierarchical index structure called *A-tree*. This provides good performance to the algorithm. IDAN will explore the tree node by node and efficient pruning mechanism can be employed such that a large number of nodes can be ignored when doing a particular task.

Incremental Algorithm IDAN supports incremental update on the dataset. New tuples can be inserted efficiently and the mining process will only process on the newly inserted data. It needs not to start from the very beginning.

Interactive Algorithm IDAN is divided into two phases: *tree-building* and *visualization* phases. The tree-building phase studies the distribution of the data and does not require any user parameters. The user can change their parameters in the visualization phase and IDAN can response immediately.

From the experiments we have done, we proved the effectiveness and efficiency of IDAN. The experiments have demonstrated that IDAN not only provides good performance and at the same time it also provides good quality on the knowledge being mined.

Bibliography

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In Jorgeesh Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *20th International Conference on Very Large Data Bases, September 12–15, 1994, Santiago, Chile proceedings*, pages 487–499, Los Altos, CA 94022, USA, 1994.

[2] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. R*-tree. an efficient and robust access method for points and rectangles. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 19(2):322–331, June 1990.

[3] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In T. M. Vijayaraman et al., editors, *Proceedings of the twenty-second international Conference on Very Large Data Bases, September 3–6, 1996, Mumbai (Bombay), India*, pages 28–39, Los Altos, CA 94022, USA, 1996.

[4] P. Cheeseman and J. Stutz. Bayesian classification (AutoClass): theory and results. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Knowledge Discovery in Data Bases II*, chapter 6, pages 153–180. AAAI Press / The MIT Press, Menlo Park, CA, 1995.

[5] H. Chernoff. The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association*, 68:361–368, 1973.

[6] William S. Cleveland. *Visualizing Data.* Hobart Press, 1994.

[7] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, New York, 1974.

[8] Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Evangelos Simoudis, Jia Wei Han, and Usama Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, page 226. AAAI Press, 1996.

[9] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhr Smyth, and Ramasamy Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining.* M.I.T. Press, March 1996.

[10] Takeshi Fukuda, Yasuhido Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Mining optimized association rules for numeric attributes. In ACM, editor, *PODS '96. Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 1996, Montréal, Canada, June 3–5, 1996*, volume 15, pages 182–191, New York, NY 10036, USA, 1996. ACM Press.

[11] K. Fukunaga. *Introduction to Statistical Pattern Recognition, Second Edition.* Academic Press, Boston, MA, 1990.

[12] Grinstein G. G. and Pickett R. M. *Iconographic displays for visualizing multi-dimensional data.* 1988.

[13] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: An efficient clustering algorithm for large databases. In *Proceedings of the ACM SIGMOD*

International Conference on Management of Data (SIGMOD-98), volume 27,2 of *ACM SIGMOD Record*, pages 73–84, New York, June1–4 1998. ACM Press.

[14] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 14(2):47–57, 1984.

[15] P. J. Huber. Projection pursuit (with discussion). *Annals of Statistics*, 13:435–525, 1985.

[16] A. Inselberg and B. Dimsdale. Parallel coordinates: a tool for visualizing multidimensional geometry. In *IEEE Visualization '90 Proceedings*, pages 361–378. IEEE Computer Society, October 1990.

[17] Alfred Inselberg. A survey of parallel coordinates. In Hans-Christian Hege and Konrad Polthier, editors, *Mathematical Visualization*, pages 167–179. Springer Verlag, Heidelberg, 1998.

[18] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice-Hall, Englewood Cliffs, NJ, 1988.

[19] Norio Katayama and Shin'ichi Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, volume 26,2 of *SIGMOD Record*, pages 369–380, New York, May13–15 1997. ACM Press.

[20] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data. An Introduction to Cluster Analysis.* Wiley, New York, 1990.

[21] D. A. Keim. Pixel-oriented database visualizations. *SIGMOD RECORD*, December 1996.

[22] D. A. Keim, H.-P. Kriegel, and M. Ankerst. Recursive pattern: A technique for visualizing very large amounts of data. In *Proc. Visualization'95*, pages 297–286, 1995.

[23] A. Kumar. A study of spatial clustering techniques. *Lecture Notes in Computer Science*, 856:57–??, 1994.

[24] J. LeBlanc, M. O. Ward, and N. Wittels. Exploring N-dimensional databases. In *IEEE Visualization '90 Proceedings*, pages 230–239. IEEE Computer Society, October 1990.

[25] King-Ip Lin, H. V. Jagadish, and Christos Faloutsos. The TV-tree — an index structure for high-dimensional data. *VLDB J.: Special Issue on Spatial Database Systems*, 3(4):517–542, October 1994.

[26] R. S. Michalski and R. Stepp. Learning from observation: conceptual clustering. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach.* Springer-Verlag, Berlin, 1984.

[27] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In Jorgeesh Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *20th International Conference on Very Large Data Bases, September 12–15, 1994, Santiago, Chile proceedings*, pages 144–155, Los Altos, CA 94022, USA, 1994. Morgan Kaufmann Publishers.

[28] Jong Soo Park, Ming-Syan Chen, and P. S. Yu. An effective hash-based algorithm for mining association rules. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 24(2):175–186, June 1995.

[29] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, Cambridge, 2 edition, 1992.

[30] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, June 1984.

[31] Ramakrishnan Srikant and Rakesh Agrawal. Mining quantitative association rules in large relational tables. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 25(2):1–12, 1996.

[32] D. A. White and R. Jain. Similarity indexing with the SS-tree. In *Proceedings of the 12th International Conference on Data Engineering*, pages 516–523, Washington - Brussels - Tokyo, February 1996. IEEE Computer Society.

[33] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: an efficient data clustering method for very large databases. In *Proceedings of theACM SIGMOD International Conference on Management of Data*, volume 25, 2 of *ACM SIGMOD Record*, pages 103–114, New York, June4–6 1996. ACM Press.