DESIGN AND ANALYSIS OF SUBSPACE CLUSTERING ALGORITHMS AND THEIR APPLICABILITY

THESIS SUBMITTED TO GOA UNIVERSITY

FOR THE DEGREE OF

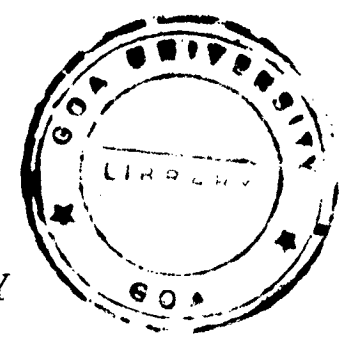
DOCTOR OF PHILOSOPHY

IN

COMPUTER SCIENCE

BY

JYOTI D. PAWAR



GOA UNIVERSITY

TALEIGAO PLATEAU

GOA-403206

INDIA

April 2004

STATEMENT

As required under the University ordinance 0.19.8(iv), I state that the present thesis entitled **"DESIGN AND ANALYSIS OF SUBSPACE CLUSTERING ALGORITHMS AND THEIR APPLICABILITY"** is my original contribution and the same has not been submitted on any previous occasion. To the best of my knowledge the present study is the first comprehensive work of its kind from the area mentioned.

The literature related to the problem investigated has been cited. Due acknowledgements have been made wherever facilities and suggestions have been availed of.

(Jyoti D. Pawar)

CERTIFICATE

This is to certify that the thesis entitled **"DESIGN AND ANALYSIS OF SUBSPACE CLUSTERING ALGORITHMS AND THEIR APPLICABILITY"**, submitted by Smt. Jyoti D. Pawar for the award of the degree of Doctor of Philosophy in Computer Science is based on her original studies carried out under my supervision. The thesis or any part thereof has not been previously submitted for any other degree or diploma in any University or Institute.

Dr. Pralhad. R. Rao
Dept. of Computer Sc., & Tech.,
Goa University,
Taleigao Plateau,
Goa – 403206.

Certified that all corrections suggested by the examiners have been incorporated in the Thesis.

22/10/05
Prof. K. Poulose Jacob
(External Examiner)

22/10/05
Guide

Place: Department of Computer Science & Technology,
Goa University, Goa.

Dated:

"Education is not that amount of information that is put into your brain and runs riot there undigested, all your life. We must have life-building, man-making, character-making, assimilation of ideas. If you have assimilated five ideas – truth, right action, peace, divine love and non-injury and made them your life and character, you have more education than any man who has got by heart a whole library. If education is identical with information, the libraries are the greatest sages in the world, and encyclopaedias are the Rishis. The ideal, therefore, is that we must have the whole education of our country, spiritual and secular, in our own hands, and it must be on national lines, through national methods as far as practical."

——— *Swami Vivekananda (1863-1902)*

Acknowledgement

It would have remained a dream to complete the thesis without the support of many people and many things as well, which proved useful in some or the other way. If I go on mentioning the names of each and every person and thing, there is a possibility that I may miss out unknowingly the names of some people and things. Hence, I have decided to thank God and only one person. For, I know when I thank him it includes all. And that person is none other than my sadguru Param Pujya Sant Shri Asaramji Bappu. When I was completely lost trying to fight against my health and both personal and professional problems, it is he who taught me the art of living a tension free life amidst all favorable, unfavorable, happy, sad and all possible circumstances. If I master what he has taught me and lead my life based on his teachings then, that will be the highest possible invaluable degree for me. And, I know that the knowledge that I get from the degree which my Bapuji, will award me if I master and mould my life based on his teachings will also prove useful to each and every person whom I meet. To practice whatever he has taught me is both *a next to impossible task as well as a very simple task*. Reading this statement, everybody will feel I have gone mad. But it is absolutely true. It becomes very simple when I forget all my ego and have a very clear conscience and see myself in everybody around me. And absolutely impossible when my ego, jealousy, anger, and the feeling of me and mine crops up.

DESIGN AND ANALYSIS OF SUBSPACE CLUSTERING ALGORITHMS AND THEIR APPLICABILITY

BY
JYOTI D. PAWAR

Abstract

Due to the rapid advancement in information technology, it has become very easy to capture data about almost every aspect of ones business or related field. The data captured is stored in various files or databases. Most of the time the mining of knowledge is carried out considering each of the files or databases independently. Hence, we cannot find the patterns or relationships that exist across attributes stored in different files or databases. In Subspace clustering, we try to find all the possible interrelationships that exist between the various data attributes by finding all the clusters that exist in the different subspaces of a very high dimensional dataset. The datasets that we deal with in subspace clustering contain a large number of attributes, are huge in size and most of the times contain many missing values.

In this thesis, based on the properties of very high dimensional huge data sets, and the requirements of the subspace clustering algorithms, an Attribute Oriented Storage Structure (AOSS) for storing very high dimensional huge data sets has been developed. Using the AOSS structure, the complexity of the function, to find the frequency count of the various candidate units in the datasets is reduced considerably. This fact is also proved by the experimental evaluation that has been carried out using

synthetic datasets. An algorithm to reduce the number of passes required over the dataset has been designed by using sampling technique and experimentally shown that it is efficient when we have to deal with huge datasets which cannot be loaded in main memory at one time. In order to efficiently find high-dimensional clusters in very high dimensional huge datasets, a depth-first approach instead of the currently used breadth-first method has been used to find the dense units in the datasets and it is extended to find clusters in datasets with attributes having varying threshold values. And finally, using the AOSS structure with this depth-first approach technique, proposed method to find the various clusters that exist within the clusters identified in the original datasets. This method can be very useful to do a through analysis of datasets in applications like census data analysis.

The AOSS structure along with the depth first method of finding the dense units is found to be very promising to make the design of the subspace clustering algorithms very efficient with respect to the space as well as the time factor to find high dimensional clusters in very high dimensional huge datasets.

Contents

1	**Introduction**	1
	1.1 Background	1
	1.2 Motivation	2
	1.3 Contributions	4
	1.4 Organization of the Thesis	6
2	**Problem Definition and Related Work**	8
	2.1 Subspace Clustering Problem	8
	2.2 **CLIQUE** algorithm	12
	2.3 Improvements over **CLIQUE**	17
3	**AOSS: An Attribute Oriented Storage Structure**	21
	3.1 Limitations of the existing storage techniques.	22
	3.2 Design of Attribute Oriented Storage Structure	24
	3.3 Database operations Using AOSS	28
	3.4 Efficiency obtained using AOSS	34
	3.5 Experimental Results	40
	3.5.1 Synthetic data generation	41
	3.5.2 Synthetic data results	42
	3.6 Summary	45
4	**SAMCLIQ: A SAMpling based CLIQue algorithm**	47
	4.1 Use of Sampling in Data Mining	48

4.1.1 Role played by sampling in data mining 48

4.1.2 Limitations of sampling 49

4.2 Proposed Sampling technique 50

4.2.1 Criteria for a good sample 50

4.2.2 Sampling for finding frequent sets 51

4.2.3 AOSS based sampling technique 52

4.3 Subspace Clustering Using Sampling 58

4.3.1 Algorithm for identification of dense units 59

4.4 Experimental Results 65

4.4.1 Synthetic data generation 65

4.4.2 Synthetic data results 66

4.5 Summary 68

5 MLSCLUS: A Multi Level Subspace CLUStering Algorithm 70

5.1 Use of Maximal Frequent Itemsets in Subspace Clustering 73

5.2 MADUGEN: A Maximal Dense Unit Generation Algorithm Using Multiple Threshold Values 76

5.2.1 Experimental results 83

5.2.2 MADUGENMT: MADUGEN algorithm with multiple threshold values 84

5.3 AOMLSCLUS: An Attribute Oriented Multi Level Subspace CLUStering Algorithm 87

5.3.1 Experimental results 99

5.4 Summary 100

6 Discussion 101

6.1 Characteristics of the AOSS method 101

6.2 Extensions and Applications of Subspace Clustering methods 103

7 Conclusions 106

7.1 Summary of The Thesis 107

7.2 Future Research Directions 108

References 110

List of Figures

2.1 Illustrations of *CLIQUE* definitions 11

2.2 Example of two datasets with equal coverage but different densities 18

3.1 A Sample record layout 23

3.2 A typical very high dimensional huge dataset 23

3.3 A Sample AOSS record layout 25

3.4 Database structure using variable length records 26

3.5 A sample AOSS record table 27

3.6 A Sample AOSS attribute table 27

3.7 Scalability with the number of records(with missing values) 43

3.8 Scalability with the number of records(without missing values) 44

3.9 Scalability with the dimension of the data space 45

4.1 Scalability with the number of records 67

4.2 Scalability with the dimension of the data space 67

4.3 Scalability with the dimensionality of the clusters 68

5.1 Scalability with the dimensionality of the clusters 84

Chapter 1

Introduction

1.1 Background

The amount of raw data and information being captured and stored in computer files and databases in almost every field has been growing at a tremendous pace. In short we can say that we have been flooded with data but we are still starving to get the knowledge from this vast pool of existing data. In today's competitive world, all concerned need to extract as much information as possible from their data sources to help in efficient decision making, so as to compete with their rivals and achieve their goals. Data mining comes into play to help users satisfy such needs. Data mining, which is also referred to as *knowledge discovery* in databases, means a process of nontrivial extraction of implicit, previously unknown and potentially useful information (such as knowledge rules, constraints, regularities) from data in databases [26]. Data mining combines methods and tools from at least three areas namely machine learning, statistics, and databases [19].

Clustering is a data mining technique that helps in identifying clusters within the domain space and has many applications in several fields. As a data mining task, data clustering also referred to as *unsupervised classification* can be thought of as partitioning or segmenting the data into groups that might or might not be disjoint. Data clustering has been studied in statistics [10,20], machine learning [14,15], and spatial

data mining [10, 11, 25] areas with different emphasis. The unsupervised nature of clustering makes it applicable to applications, where the user has limited domain knowledge. Some of the current applications, which use clustering techniques extensively are clustering of web-search results and clustering of spatial databases. Most of the traditional clustering algorithms have been designed to discover clusters in the full dimensional space using various distance functions.

1.2 Motivation

In recent years, there has been an increase in the number of new database applications dealing with very large high dimensional data sets. These applications to name a few include multimedia content-based retrieval, geographic and molecular biology data analysis, text mining, bio-informatics, medical applications, and time-series matching. These applications place special requirements on clustering algorithms: the ability to find good quality clusters embedded in subspaces of high dimensional data preferably without taking any inputs from the user (which requires the user to have good domain knowledge), scalability, end-user comprehensibility of the results, non-presumption of any canonical data distribution, and insensitivity to the order of input records. Clustering algorithms which work on the full dimensional space of the data fail to find clusters in high dimensional datasets due to the following main reasons – the average density of points anywhere in the high dimensional data space is likely to be

low [6]. Secondly, in the high dimensional data there are more chances of having missing values in the data attributes. In order to apply the full dimensional clustering algorithms, these missing values are normally replaced by values taken from a random distribution say X. Here an assumption is made that, the attribute containing missing values, follows that particular X distribution. This assumption need not be true always and thereby affect the quality of the clustering results obtained. Majority of the traditional clustering algorithms are sensitive to the order of input records and require input parameters from the user.

The subspace clustering algorithm *CLIQUE* [1] satisfies some of the above requirements. It identifies the subspace clusters in the high dimensional data by finding all the sets of connected dense units existing in the various subspaces. It presents the cluster descriptions in the form of DNF expressions that are minimized for easy interpretation. It produces identical results irrespective of the order of the input records and does not need to make any assumptions about the data distributions for any attributes to handle any missing values. However, it requires the user to give the inputs, τ (threshold value) and ξ(number of intervals) in order to find the dense units. Hence the accuracy of the results obtained depends on the values input by the user. It uses the level-wise *apriori* [4] algorithm for finding the dense units. Hence suffers from the same problems as the *apriori* algorithm in the following situations:

- If the user inputs a large value for ξ or enters a very low value for τ, the number of candidate and dense units generated will be huge in number. And as a result

the first step of *CLIQUE* to identify the dense units in the different subspaces will be computationally very expensive.

- If the dimensionality of the clusters is large, then the database will have to be scanned a large number of times to find the high dimensional dense units. And, if the size of the database is also very large then it will still add to the time complexity.

As a result of the emerging real life data applications, there is a demand for clustering algorithms, which can efficiently identify good quality clusters from huge, high dimensional data sets. Hence, developing efficient techniques to find clusters in huge, high dimensional data sets has become an important research direction in data mining.

1.3 Contributions

In this thesis, we study the problem of subspace clustering for very high dimensional huge data sets with missing values. In particular, we make the following contributions -

- **Efficient storage structure:** Based on the properties of very high dimensional huge data sets containing missing values, and the requirements of the subspace clustering

algorithms, we have developed an Attribute Oriented Storage Structure (AOSS) for storing very high dimensional huge data sets.

- **Scalability:** With the increasing size of the databases, we need to have subspace clustering algorithms, which can be used for very large data sets. We have used the sampling technique to address this issue. The *SAMCLIQ* algorithm developed using sampling technique gave us very efficient results when compared with the *CLIQUE* algorithm.

- **Efficiency:** To handle this issue, we have used a depth-first approach and the concept of maximal dense units for identifying the subspaces containing the clusters. The subspace clustering algorithms CLIQUE [1] and MAFIA [16] have used the level-wise *apriori* algorithm for identifying the dense units. Again here we used the AOSS method of storage representation and found that it gives very good results for very high dimensional huge datasets with missing value attributes.

- **Applicability:** We extended the AOSS method using the maximal dense unit concept to find clusters in datasets containing attributes with varied threshold requirements. As an application of this technique in applications like census data analysis, we developed a subspace clustering algorithm to allow mining of all the subspace clusters found in the clusters identified in the original dataset.

1.4 Organization of the Thesis

The remainder of the thesis is structured as follows:

- In Chapter 2, we present the subspace clustering problem and an overview of the related work carried out in high-dimensional clustering.

- In chapter 3, An Attribute Oriented Storage Structure (AOSS) for storing very high dimensional datasets with many missing values has been developed. The reduction in time complexity using this structure is reported along with the experimental results obtained using synthetic datasets.

- In Chapter 4, a sampling based subspace clustering algorithm *SAMCLIQ* [36] is developed to handle very large data sets. The experimental evaluation and performance study by comparing with CLIQUE has been carried out.

- In Chapter 5, details of algorithms developed using AOSS based structure for finding maximal dense units with uniform threshold value (MADUGEN) and multiple threshold values (MADUGENMT) have been discussed. Using AOSS structure and MADUGENMT a subspace clustering algorithm AOMLSCLUS, has been presented and its application for analyzing census data discussed.

- In Chapter 6, we summarize the characteristics of the AOSS method along with a discussion of some interesting extensions and applications of subspace clustering using AOSS.

- In Chapter 7, we conclude with a few directions for future work.

Chapter 2

Problem Definition and Related Work

In this chapter, we first define the subspace clustering problem, then we discuss the working of the *CLIQUE* [1] algorithm. A few improvements over the *CLIQUE* algorithm are also discussed.

2.1 Subspace Clustering Problem

The Subspace clustering problem was first introduced by R. Agrawal, in [1]. Subspace Clustering is the most informative/systematic approach for clustering high-dimensional data. It is the task of automatically identifying(in general several) subspaces of a high dimensional data space that allow better clustering of the data objects than the original data space [1].

Terminology Used:

Let $A = \{ A_1, A_2, \ldots, A_d \}$ be a set of bounded, totally ordered domains and $S = A_1 \times A_2 \times \ldots \times A_d$ a d-dimensional numerical space. $A_1, \ldots, A_d$ are referred to as the dimensions (attributes) of S.

The input consists of a set of d-dimensional points $V = \{ v_1, v_2, \ldots, v_m \}$ where $v_i = < v_{i1}, v_{i2}, \ldots, v_{id} >$. The jth component of v_i is drawn from domain A_j.

The data space S is partitioned into non-overlapping rectangular units. The units are obtained by partitioning every dimension into ξ intervals of equal length, which is an input parameter.

Each unit u is the intersection of one interval from each attribute. It has the form $\{u_1, \ldots, u_d\}$ where $u_i = [li,hi)$ is a right-open interval in the partitioning of A_i.

A point $v = \{v_1, v_2, \ldots, v_d\}$ is contained in a unit $u = \{u_1, u_2, \ldots, u_d\}$ if $l_i \leq v_i < h_i$ for all u_i.

The *selectivity* of a unit is defined to be the fraction of the total data points contained in the unit. A unit u is called a *dense unit* if selectivity(u) is greater than τ, the *density threshold* which is input by the user.

A *k-dimensional subspace* is a projection of the data set V into $A_{t1} \times A_{t2} \times \ldots \times A_{tk}$, where $k < d$ and $t_i < t_j$ if $i < j$. A *k-dimensional unit* u^k in this subspace is the intersection of an interval from each of the k attributes.

A *cluster* is a maximal set of connected dense units in k-dimensions. Two k-dimensional units u_1, u_2 are *connected* if they have a common face or if there exists

another k-dimensional unit u_3 such that u_1 is connected to u_3 and u_2 is connected to u_3.

Units $u^k_1 = \{ r_{t1}, \ldots, r_{tk}\}$ and $u^k_2 = \{ r'_{t1}, \ldots, r'_{tk}\}$ have a common face if there are k-1 dimensions, assume dimensions $A_{t1}, \ldots, A_{tk-1}$, such that $r_{tj} = r_{tj}'$ for j = 1 to k-1 and either $h_{tk} = l'_{tk}$ or $h'_{tk} = l_{tk}$.

A *region* in k dimensions is an axis–parallel rectangular k-dimensional set. Regions are considered as unions of units. Region R is said to be contained in a cluster C if $R \cap C = R$.

A region R contained in a cluster C is said to be *maximal* if no proper superset of R is contained in C.

A *minimal description* of a cluster is a non-redundant covering of the cluster with maximal regions. That is, a minimal description of a cluster C is a set R of maximal regions such that their union equals C but the union of any proper subset of R does not equal C.

The Problem: Given a set of data points and the input parameters ξ and τ, find clusters in all subspaces of the original data space and present a minimal description of each cluster in the form of a DNF expression.

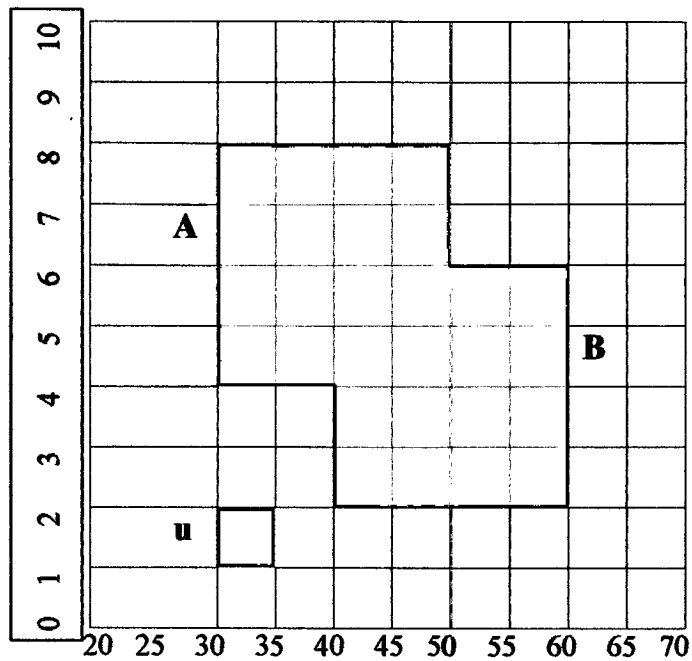


Figure 2.1: Illustration of *CLIQUE* definitions.

Example 2.1:

In Figure 2.1, the two dimensional space (age, salary) has been partitioned by a 10 X 10 grid.

A *unit* is the intersection of intervals; above an example of a 2-dimensional unit u = $(30 \leq \text{age} < 35) \wedge (1 \leq \text{salary} < 2)$.

A *region* is a rectangular union of units. A and B are both regions: A = $(30 \leq \text{age} < 50) \wedge (4 \leq \text{salary} < 8)$ and B = $(40 \leq \text{age} < 60) \wedge (2 \leq \text{salary} < 6)$.

The *minimal description* for the cluster $(A \cup B)$ is the DNF expression:

$A = (30 \leq \text{age} < 50) \wedge (4 \leq \text{salary} < 8) \vee (40 \leq \text{age} < 60) \wedge (2 \leq \text{salary} < 6)$.

2.2 *CLIQUE* Algorithm

The *CLIQUE* algorithm consists of the following three steps:

1. Identification of subspaces that contain clusters.
2. Identification of clusters.
3. Generation of minimal description for the clusters.

The main part of step 1 consists of finding the dense units in different subspaces. The dense units are identified using a bottom-up algorithm that exploits the monotonicity of the clustering criterion with respect to dimensionality to prune the search space. This algorithm is similar to the apriori algorithm for mining association rules [4]

Example 2.2 Let the transaction database, TDB, be Table 2.1 consisting of a total of 10 transactions, with 6 numeric attributes each and the user input values of ξ and τ be 5 and 0.2 respectively. The *missing values* for the attributes are represented by a '?' symbol. Assume for the sake of simplicity that all the attribute values range from 1 to 100.

Tid	A	B	C	D	E	F
10	1	21	41	?	4	18
20	4	24	44	9	9	49
30	6	26	46	45	23	83
40	9	29	?	57	?	5
50	2	8	25	58	78	30
60	53	?	92	59	?	52
70	19	8	89	58	78	57
80	82	2	?	52	72	12
90	89	78	10	25	?	38
100	?	68	75	?	62	13

Table 2.1: A transaction database TDB.

CLIQUE finds the dense units for identification of the subspaces containing clusters as follows –

1. Each attribute is split into ξ intervals to form ξ 1-dimensional candidate units for each attribute namely A1, ...A5, B1,...,B5, C1,...,C5, D1,...,D5 and so on till F1,..., F5.

   Hence in this example we will have a total of 5 * 6 = 30 1-dimensional candidate units.

2. By doing a first pass over the above dataset, the frequency count of all these 1-dimensional candidate units is found. Selectivity of a unit is equal to the frequency count of that unit divided by the total number of transactions. Those units whose selectivity is greater than 0.2 are identified as 1-dimensional dense units D_1. D_1 in this example is {A1, A9, B1, B3, C5, D6, E1, E8, F2, F6}
3. The 2-dimensional candidate units C_2, are generated by forming all possible pairs of the 1-dimensional dense units D_1. Some candidate units are pruned. Only those candidate units are retained which have all its subset units dense. A 2-dimensional candidate unit $u^1_i u^1_j \in C_2$ if and only if $u^1_i , u^1_j \in D_1$. In this example, C_2 consists of { A1B1, A1B3, ..., E8F2, E8F6}
4. A second pass is made through the dataset to find the selectivity of all the two dimensional candidate units $u^2_i \in C_2$ for i = 1 to n, n representing the total number of 2-dimensional candidate units. Thus we get, D_2 consisting of {A1B3, B3C5}
5. For $k \geq 3$, the candidate units generation procedure and procedure used for pruning the generated candidate units is as given below -

   The candidate generation procedure used for generating C_k from D_{k-1} is as under –

   insert into C_k

   select $u_1.[l_1 , h_1), u_1.[l_2 , h_2), \ldots, u_1.[l_{k-1} , h_{k-1}), u_2.[l_{k-1} , h_{k-1})$

   from $D_{k-1}\, u_1$, $D_{k-1}\, u_2$

   where $u_1.a_1 = u_2.a_1 , u_1.l_1 = u_2.l_1 , u_1.h_1 = u_2.h_1,$

$u_1.a_2 = u_2.a_2$, $u_1.l_2 = u_2.l_2$, $u_1.h_2 = u_2.h_2$, ...,

$u_1.a_{k-2} = u_2.a_{k-2}$, $u_1.l_{k-2} = u_2.l_{k-2}$, $u_1.h_{k-2} = u_2.h_{k-2}$,

$u_1.a_{k-1} < u_2.a_{k-1}$

In the above pseudo-code for the join operation, $u.a_i$ represents the ith dimension or attribute of unit u and $u.[l_i , h_i)$, represents its interval in the ith dimension.

Prunning procedure used for k-dimensional candidate units C_k - All those C_k units which do not have all its (k-1) dimensional subsets in the set of (k-1) dimensional dense units are discarded from the set of C_k units generated above.

Then the k^{th} pass is done to find selectivity of all C_k units and obtain the D_k units.

This process is continued till no candidate units can be derived or no candidate is dense. In this manner all the dense units belonging to the different subspaces are found. These units form the input for the second step of *CLIQUE*.

Time complexity: If k is the highest dimensionality of any dense unit and m is the number of the input points, the above algorithm will make k passes over the database. If a dense unit exists in k dimensions, then all of its projections in a subset of the k dimensions that is, $O(2^k)$ different combinations will also be dense. Hence, the time complexity of this algorithm is $O(c^k + mk)$ for a constant c.

The **second step** of *CLIQUE* takes as input the set of dense units D, all in the same k-dimensional space S and outputs a partition of D into P_1 , ..., P_q , such that all

units in P_i are connected and no two units $u_i \in P_i$, $u_j \in P_j$ with $i \neq j$ are connected. All these partitions represent the clusters found in the k-dimensional space S. It finds the partitions by using a depth-first search algorithm to find the connected components in the graph formed by representing the dense units as the vertices of the graph. An edge exists between those vertices whose corresponding dense units have a common face.

The **step three** takes as input the clusters identified in step two and generates a concise description for it. For this purpose it first uses a greedy growth method to cover the clusters by a number of maximal rectangles(regions), and then discards the redundant rectangles to generate a minimal cover.

Some of the *drawbacks of the CLIQUE* algorithm are as under-

- It does not provide any support to the user for selecting the values for the input parameters ξ and τ. The cluster boundaries generated are totally dependant on the value of ξ and the value of τ decides the quality of the clusters that will be generated. If the value of τ is set too low then we will get a large number of dense units, and some of the clusters that we get from these dense units will be redundant. Similarly, if the value is too high then we will miss to capture some significant clusters.

- It is tedious to make repeated passes over the database to find the selectivity of the large number of candidate units generated. This condition worsens when the dimensionality of the subspace clusters found in the database increases. As the dimensionality of the subspace clusters increases, there is an explosion in the number of dense and the candidate units generated. *CLIQUE* uses a *MDL-based pruning* technique. In this the dense units in the subspaces with low *coverage* are pruned so as to reduce the number of dense and candidate units generated. The *coverage* of a subspace is the fraction of the database that is covered by the dense units. This is believed to make the algorithm faster but it may lead to missing out of some important clusters.

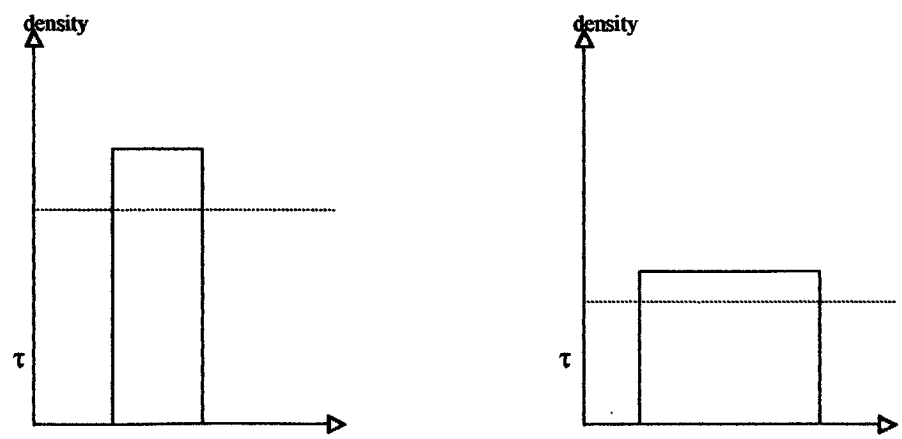
- If the size of the dataset is very large both with respect to the number of records and the number of attributes (data dimensionality), the time taken for each database pass to find the selectivity of the candidate units will increase substantially.

2.3 Improvements over *CLIQUE*

In the past few years, some subspace clustering algorithms have been proposed to overcome some of the problems of the *CLIQUE* algorithm.

The *ENCLUS [8],* a ENtropy-based subspace CLUStering algorithm was proposed to handle the large number of subspaces with clusters within them. In *CLIQUE,* the MDL-based pruning technique was used to prune some subspaces with low coverage to make the algorithm faster. However, it had the trade-off of missing out some significant dense units found in subspaces with low coverage. The *ENCLUS [8]* algorithm has made the following contributions to the subspace clustering problem –

- It has identified the following additional criteria for determining subspaces with good clustering:
  a) Criterion of High Coverage
  b) Criterion of high density and
  c) Correlation of dimensions



(a) (b)

Figure 2.2: Examples of two data sets with equal coverage but different densities. The area within the rectangles is the value of the coverage.

In the figure 2.2 cases (a) and (b) have the same coverage, however the points in (a) are more closely packed and is a better candidate to qualify as a cluster.

- *ENCLUS [8]* uses the entropy metric to measure all the above three criteria simultaneously to find subspaces with good clustering. It is also a grid based method and takes the inputs for the threshold from the user. In order to calculate the entropy, it also divides each dimension into equal width intervals to form a grid. Hence the size selected for the intervals, affects the quality of the final clustering obtained.

- *MAFIA [16]* (Merging of Adaptive Finite Intervals) is another subspace clustering algorithm, which uses adaptive interval sizes to partition the dimension depending on the distribution of data in the dimension. Using adaptive grid sizes, *MAFIA* attempted to reduce the computation and improve the clustering quality by concentrating on the portion of the data space which have more data points and thus more likelihood of having clusters.

- *PROCLUS [2]* uses the concept of PROjected CLUStering for finding clusters in a multi-dimensional data space. *PROCLUS* also discovers interesting correlations among the data in various subspaces of the original high dimensional space, but it differs from *CLIQUE* in the output produced. It outputs a partition of the data points into clusters, together with the sets of

dimensions on which points in each cluster are correlated. The clusters output by *PROCLUS* are useful in applications like classification and trend analysis where it is required to partition the data points into disjoint partitions. It fails to detect any overlapping clusters existing in the data set. ORCLUS [3] is also an example of a projected clustering algorithm.

Chapter 3

AOSS: An Attribute Oriented Storage Structure

We discussed the details of the first subspace clustering algorithm *CLIQUE* in Section 2.2. We observed that, one of the major drawbacks of the algorithm is the repeated number of database passes required during step one to find the selectivity of the large number of candidate units that are generated. In step one of *CLIQUE*, which is based on the *apriori [4]* algorithm we observe that the entire database is scanned in order to find the selectivity of each candidate unit. Hence, as the number of candidate units increases the time taken for each database pass increases proportionately. In reality, all the database records need not be accessed to find the selectivity of each candidate unit. Similarly, in the case when the dimensionality of the data space is very large it is not required to access the entire data record to find the selectivity of all the units. In order to find the selectivity of a unit u it needs to access only the values of those attributes, which are a part of the unit u. In short, if we can tackle the above two problems we can significantly improve the performance of step one of *CLIQUE*.

Can we cut short on the number of data records and the number of attribute values of each record that are accessed by each unit to find its selectivity? To handle these concerns, we develop in this chapter an efficient method for storing the

very high dimensional huge datasets. Because of the ease with which data can be captured today from almost every aspect of the problem domain, all the datasets consist of a huge number of data records with a very large number of data attributes. First, we discuss the limitations of the existing data storage techniques to support the clustering of very high dimensional huge datasets in Section 3.1. Then in Section 3.2, we propose the Attribute Oriented Storage Structure (AOSS) method for storing the datasets. In Section 3.3, we discuss how to perform the various database operations on the data stored using the proposed AOSS method, in section 3.4 we report the experimental results and conclude with a summary in Section 3.5.

3.1 Limitations of the Existing Storage Techniques

In subspace clustering, we try to find all the possible interrelationships that exist between the various data attributes. If we apply subspace clustering, on individual files then we will only find the subspace clusters existing within each of them independently. However, to get all the possible knowledge or patterns from the available data, we will have to consider all the data at one place in a database. Each record in such a database should consist of all the attributes of ones business. The resulting database that we get will represent a very high dimensional huge dataset. Such a database is likely to contain many missing values.

A Sample layout of a record consisting of 100 attributes will be as under –

a_1	a_2	a_3	?	a_5	?	a_7	. . .	a_{99}	a_{100}

Figure 3.1: A Sample record layout

The a_i value, represents the value of the ith attribute A_i, for i = 1 to 100 and a '?' represents a missing value for the corresponding attribute. For example in the above layout, values of attribute A_5 and A_6 are missing. A typical database layout, consisting of ten lakhs of data records storing information about 100 attributes will be as under.

	A_1	A_2	.....	A_i	.....	A_{100}
Rec1	a_1	a_2	?	?	a_i	?	?	a_{100}
Rec2		a'_2	?	a'_{i-1}		a'_{i+1}		?	a'_{100}
								?	
Rec999999						?			
Rec1000000	a''_1		a''_j			a''_{i+1}			a''_{100}

Figure 3.2: A typical very high dimensional huge dataset.

Some of the drawbacks of the representation shown in figure 3.2 are as under:

- Since, there will be many missing data values, a lot of storage space will be wasted.
- Each time you access any record, you will be reading all the attribute values of the record. Majority of the cases you do not need to access all the attributes at the same time.
- This representation is not suitable for subspace clustering algorithms. In order to find the selectivity of the different candidate units, we need not access all the attributes of the data and all the data records need not be accessed for all the candidate units.

Hence, we propose the following Attribute Oriented Storage Structure (AOSS) for storing such a very high dimensional huge dataset.

3.2 Attribute Oriented Storage Structure(AOSS)

In the Attribute Oriented Storage Structure (AOSS), we store the information in such a way that all the records from the database are not accessed to find the selectivity of the various candidate units. And at the same time we access only the attribute information of the attributes, which are present in that particular candidate unit while finding its selectivity.

We explain below the various steps carried out to arrive at such a structure.

- *Step 1:* In Figure 3.2, we observe that there are a lot of missing values in the various records. We could save a lot on space and time to process such missing value attributes if we store only those attributes having valid values. By doing so we will get records with varying number of attributes. And the database so constructed will lead to a substantial reduction in the space required. The record layout shown in figure 3.1 and the database structure of figure 3.2 will appear as under using the new AOSS structure-

(A_1,a_1)	(A_2,a_2)	(A_3,a_3)	(A_5,a_5)	(A_7,a_7)	. . .	(A_{99},a_{99})	(A_{100},a_{100})

Figure 3.3: A Sample AOSS record layout

In the above figure 3.3, (A_i, a_i) represents the ith attribute value pair for those attributes having valid values. We do not show the details of the 4^{th}, 6^{th}, and other attributes which contain missing values.

After the step one the database shown in figure 3.2 will appear as shown in figure 3.4 with records containing varying number of attribute value pairs.

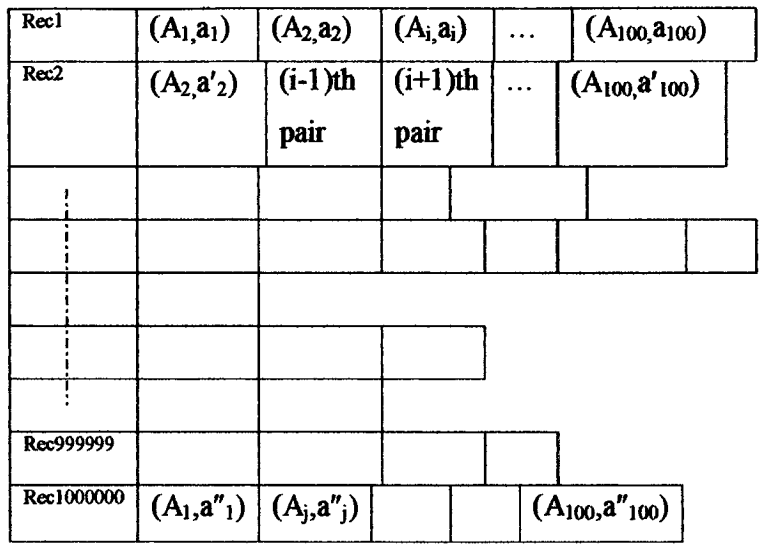


Figure 3.4: Database structure using variable length records

Step 2: The above structure reduces the space required by eliminating the information of the missing value attributes from each record. Hence it results in shortening the length of the data records to be processed during each database pass to find the selectivity of the units. While finding the selectivity of a unit u, to avoid the processing of those attributes, which are not part of the candidate unit u, the above structure shown in figure 3.4 is split into two levels. At the top level, we have the *AOSS record table*. In this table, for each record we keep along with the record identifier the attribute information details, which include the address of the table storing the attribute values and the position where the value is stored in it. And at the next level we have independent *AOSS attribute tables* for storing the values of each attribute along with its record identifier from the AOSS record table. Using the AOSS method, the database structure from figure 3.4 will be represented with the

help of a AOSS record table and the AOSS attribute tables. A Sample of the structure of a AOSS record table and a AOSS attribute table is shown below.

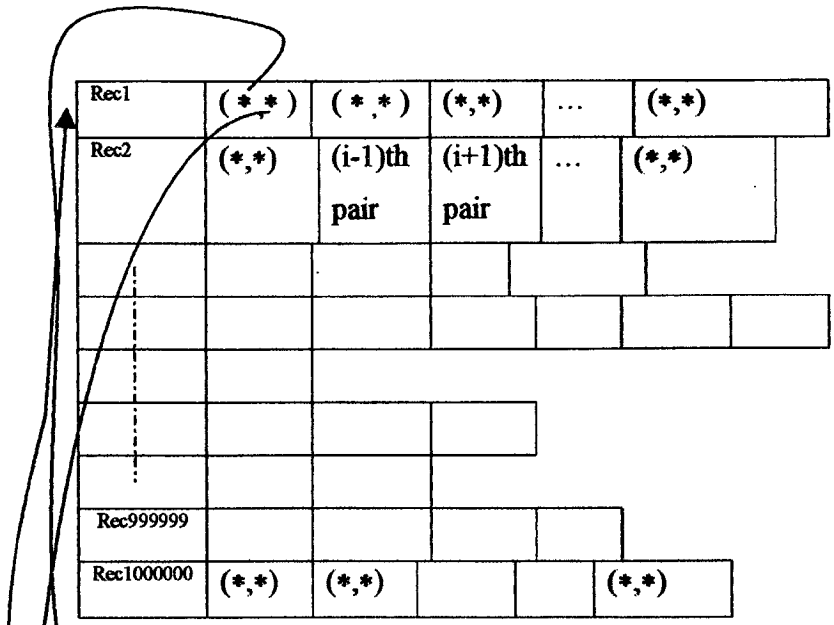


Figure 3.5: A sample AOSS record table

A_1 attribute table

Attribute Rec-id	Rec-id from AOSS record table	Attribute value
Arec-1		a_1
Arec-2		a_1'

Figure 3.6: A Sample AOSS attribute table

In figure 3.5 and figure 3.6, we have shown a sample of the AOSS record table and a sample of the AOSS attribute table respectively. There will be a separate AOSS attribute table for each attribute. Hence, for our database of figure 3.2 we will have 100 such tables. Another advantage of this is that when we have a large number of candidate units we can split them into units with disjoint sets of attributes and process them in parallel by using the independent sets of attribute tables.

3.3 Database Operations Using AOSS

Currently, we have all database operations defined with records as the base unit. That is to say we have operations for creating, reading, deleting, and updating records. Each record is considered as consisting of a fixed set of data attributes. When we deal with very high dimensional data, and which is most of the time sparse in nature, it no longer makes sense to still continue with record as a base unit for carrying out the database operations. In subspace clustering, we are interested in capturing all the possible interrelationships that exist between the various data attributes. Therefore, it is sensible to consider an attribute as the base unit and define all the database operations in terms of attributes. Hence, we have defined operations for attribute creation, attribute reading, attribute insertion, attribute deletion, and attribute updation. Since, the AOSS design is mainly developed to make the subspace clustering process efficient, we assume that the most frequently performed operation will be Attribute reading and the other operations will be very infrequently

carried out. The details of all these operations considering the AOSS method for representing the data are discussed below-

Attribute Creation: This operation involves creating a new AOSS attribute table and storing the address of the table in the *Attribute-details table.* The *Attribute-details table* stores the addresses of all the attribute tables along with the attribute-ids and their descriptions. In figure 3.5 and figure 3.6 we have not shown the Attribute-details table so as not to show the low-level implementation details and make the figure complicated.

Algorithm details:

1) *Read the attribute description.*
2) *Check if it exists in the Attribute-details table.*
3) *If it exists report "Attribute already exists" and go to 6*
4) *Generate Attribute-id and store Attribute-id and description in Attribute-details table.*
5) *Allocate space for attribute-table and store its address in Attribute-details table. The attribute table could be stored as a separate file and its address will mean here its filename.*
6) *Stop*

Attribute Insertion: This involves entering the attribute values of some specific attributes for specific records. The record entries may already be existing in the

AOSS record table or may not be existing. The details of the insertion operation are as under-

Algorithm details:

1) *Read Attribute-id/ Attribute description*
2) *Check if it exists in the Attribute-details table*
3) *If does not exist report" Attribute not found " go to 11*
4) *Read record-id*
5) *Check if record-id exists in AOSS record table*
6) *If not found report and then create after receiving confirmation from user.*
7) *Check if attribute information for the record exists in the AOSS record table.*
8) *If found, then display existing value from AOSS Attribute table and allow to update after receiving user's confirmation and go to 11 .*
9) *Add the attribute-details including its id, description and record-id to the AOSS attribute table for that attribute.*
10) *Add the Attribute-id and the Attribute rec-id to the AOSS record table entry for this record.*
11) *Stop.*

Attribute Deletion: Attribute deletion takes as input the attribute-id or attribute description and the record-id of record whose details need to be deleted and removes

the entry from the AOSS attribute table and also from the AOSS record table's entry for that record-id.

Algorithm details:

1) *Read Attribute-id/ Attribute description*
2) *Check if it exists in the Attribute-details table*
3) *If does not exist report" Attribute not found " go to 11*
4) *Read record-id*
5) *Check if record-id exists in AOSS record table*
6) *If not found report and go to 11*
7) *Check if attribute information for the record exists in the AOSS record table.*
8) *If not found report error go to 11*
9) *If found, then display existing value from AOSS Attribute table and delete after receiving user's confirmation*
10) *Delete the corresponding entry for that record-id from the AOSS record table.*
11) *Stop.*

Attribute Updation: Attribute Updation takes as input the attribute-id or attribute description and the record-id of record whose details need to be updated.

Algorithm details:

1) *Read Attribute-id/ Attribute description*
2) *Check if it exists in the Attribute-details table*

3) If does not exist report" Attribute not found " go to 12

4) Read record-id

5) Check if record-id exists in AOSS record table

6) If not found report and go to 12

7) Check if attribute information for the record exists in the AOSS record table.

8) If not found report error go to 12

9) If found, then display existing value from AOSS Attribute table

10) Read new value for the attribute

11) Write new value into the AOSS Attribute table.

12) *Stop.*

Attribute Reading: This operation allows you to read the attribute value of a particular attribute for a particular record-id.

Algorithm details:

1) Read Attribute-id/ Attribute description

2) Check if it exists in the Attribute-details table

3) If does not exist report" Attribute not found " go to 10

4) Read record-id

5) Check if record-id exists in AOSS record table

6) If not found report and go to 10

7) Check if attribute information for the record exists in the AOSS record table.

8) If not found report error go to 10

9) If found, then display existing value from AOSS Attribute table

10) *Stop.*

Load Attribute-values: In subspace clustering, most of the times all the attribute values of a particular attribute-id need to be accessed at one time irrespective of their record-id's to find the selectivity of the candidate units. This is very efficient in the AOSS method as all the values of a particular attribute are stored in its AOSS Attribute table.

Algorithm details:

1) Read Attribute-id/ Attribute description

2) Check if it exists in the Attribute-details table

3) If does not exist report" Attribute not found " go to 5

4) While not end of AOSS Attribute table of Attribute-id attribute

Read attribute values;

5) Stop

3.4 Efficiency obtained using AOSS

In this section we discuss the time efficiency achieved using the AOSS method. This method of representation has mainly helped in making the algorithm used to find the frequency count of candidate units efficient. We present the details of the algorithm used in our implementation to find the frequency count of a candidate unit using the old Record Oriented Structure (ROS) as well as the AOSS representation and demonstrate the working with the help of the dataset given in table 2.1 in chapter 2.

Algorithm used for finding frequency count of a k-dimensional unit u using AOSS

Inputs

- unit u – consisting of k attribute-id and unit-id pairs, along with *startrec* and *endrec* of each pair. *startrec* and *endrec* denote the first and last occurrence positions of the unit in the dataset respectively.
- AttTable - Attribute table containing record-id details of attributes present in the unit u

Output

- frequency count of unit u

Processing

1. startpos = greatest of startrec values of the attribute-units forming the unit u
2. endpos = smallest of endrec values of the attribute-units forming the unit u
3. if (endpos <= startpos)
4. count = 01
5. go to step 38
6. for d = 1 to k
7. initialize curpos[d] to 1 // to keep track of record-ids in AttTable of various units contained in unit u
8. curpos1 = curpos[1]
9. att1 = attribute-id of first pair of u
10. unit1 = unit-id of first pair of u
11. // skip all rec-ids less than startrec of the first attribute Arecarray
12. while(AttTable[att1][unit1].Arecarray[curpos1] < startpos)
13. increment curpos1
14. startpos = AttTable[att1][unit1].Arecarray[curpos1]
15. m = curpos1 + 1
16. while(startpos < = endpos and m < AttTable[att1][unit1].*reccnt*)

// *reccnt* in above step represents number of record-ids stored in Arecarray of AttTable for that att-id and unit-id

17. match = TRUE;
18. for d = 1 to k
19. att = attribute-id of d th pair of u

```
20.  uid = unit-id of d th pair of u
21.  curposd = curpos[d]
22.  for ( p =  curposd; p < AttTable[att][uid].Reccnt; p++)
23.       if ( AttTable[att][uid].Arecarray[p]  < startpos )
24.            continue        // i.e move to next iteration of for p step 22
25.       else
26.            break           // i.e out of for p loop
27.  if ( AttTable[att][uid]. Arecarray[p]  equal to startpos )
28.       curpos[l] = p+1
29.       continue // move to next iteration of for d  loop step 18
30.  else
31.       match = FALSE
32.       curpos[l] = p;
33.       break; // i.e out of for d loop
34.  if (match is equal to TRUE )
35.       increment count
36.  startpos = AttTable[att1][unit1].Arecarray[m];
37.  increment m and if m < AttTable[att1][unit1].reccnt  goto 16//end while loop
38. stop
```

Algorithm used for finding frequency count of a k-dimensional unit u using ROS

Inputs

- unit *u* – consisting of k attribute-id and unit-id pairs
- Region - table containing unit-ids of all records of all attributes present in the dataset

Output

- frequency count of unit u

Processing

```
1. for p = 1 to number of records in dataset // complexity more due to this step
2.      match = TRUE
3.      for d = 1 to k //  number of attribute-id unit-id pairs in unit u
4.              attid = attribute-id of d th pair of unit u
5.              unitid = unit-id of d th pair of unit u
6.              if( Region[attid][attid]  not equal to unit-id)
7.                      match = FALSE
8.                      goto step 11
9.      if (match  equal to TRUE)
10.             increment count
11. stop
```

For the sake of continuity, we reproduce table 2.1 here again, with minor modifications with respect to notations used – here we use Rec-id, -1 instead of Tid

and ? to represent record-id and missing values respectively and the attributes are named as A_1, A_2, A_3, A_4, A_5, A_6 instead of A, B, C, D, E, F.

Rec-id	A_1	A_2	A_3	A_4	A_5	A_6
10	1	21	41	-1	4	18
20	4	24	44	9	9	49
30	6	26	46	45	23	83
40	9	29	-1	57	-1	5
50	2	8	25	58	78	30
60	53	-1	92	59	-1	52
70	19	8	89	58	78	57
80	82	2	-1	52	72	12
90	89	78	10	25	-1	38
100	-1	68	75	-1	62	13

In this example, considering the value of ξ to be equal to 5, we obtain the region table shown in table 5.1 storing the unit-ids 0 to 5 for the respective attribute values, 0 is used to represent missing values or values out of range, 1 for values from 1 to 20, 2 from 21 to 40, and so on ... The range of values above is from 1 to 100.

Rec-id	A_1	A_2	A_3	A_4	A_5	A_6
10	1	2	3	0	1	1
20	1	2	3	1	1	3
30	1	2	3	3	2	5
40	1	2	0	3	0	1
50	1	1	2	3	4	2
60	3	0	5	3	0	3
70	1	1	5	3	4	3
80	5	1	0	3	4	1
90	5	4	1	2	0	2
100	0	4	4	0	4	1

Table 3.1: Region table used in ROS

Consider the 2-dimensional unit u = {(A_1 , 1) (A_4 , 3)}. Using ROS, it will scan through the entire Region table consisting of the 10 entries and obtain the frequency count as 4 for this unit as there are 4 records which have for attributes A_1 and A_4 their values falling in unit 1 and unit 3 respectively.

Using AOSS, it will access only the AttTable entries of attributes A_1 and A_4 for unit 1 and 3 respectively. AttTable[A_1][1].Arecarray = {10, 20, 30, 40, 50, 70} and that of AttTable[A_4][3].Arecarray = {30, 40, 50, 60, 70, 80}. The startrec for attributes A_1 and A_4 are 10 and 30 respectively and the endrec values are 70 and 80. In AOSS, the startpos value is equal to the value of the largest startrec and the endpos is equal to the value of the lowest endrec of all the attribute-unit pairs occurring in unit u. In this case the startpos is 30 and endpos is 70. Hence it just starts scanning from 30 in AttTable[A_1][1].Arecarray , finds if 30 is found in

AttTable[A_4][3].Arecarray, since it is found it increments the count. Next moves to 40 in AttTable[A_1][1].Arecarray and this time AttTable[A_4][3].Arecarray curpos[4] would be pointing to 40, since they match it increments count moves on to next in both tables till AttTable[A_1][1].Arecarray[curpos[1]] becomes greater than endpos. When curpos of A_1 moves to 70 that time curpos of A_4 will be pointing to 60, since they do not match, curpos of A_4 is moved to the next position which now points to 70 since they match count is incremented and both curpos are incremented. At this point it so happens in this example that both termination conditions are true – that is it has reached the end of AttTable[A_1][1].Arecarray and also curpos of AttTable[A_4][3].Arecarray is greater than endpos i.e 70. Using this method it greatly reduces on the number of records accessed and also we can load only the attribute tables of those attributes, which are present in the unit u.

3.5 Experimental Results

In this section we present an empirical evaluation of the CLIQUE algorithm using the AOSS file structure using synthetic datasets. The objective of the experiments was to compare the time efficiency of the CLIQUE algorithm when implemented using the old record based file structure and the proposed attribute oriented file structure for storing very high dimensional huge data sets. We compared the performance by varying the size of the database, dimension of the data

space and dimension of the clusters. The experiments were run on a 3.00GHz Pentium 4 processor running linux.

3.5.1 Synthetic data generation

The synthetic data generation method described in [1] has been used for the data generation. The data generator takes as input the number of records to be generated, the number of attributes and the range of values for each attribute. The range of values was set to [1,100] for all attributes. The clusters are hyper-rectangles in a subset of dimensions such that the average density of points inside the hyper-rectangle is much larger than the average density in the subspace. The cluster description details provided by the user include the number of clusters, the maximum dimensionality of the clusters, and the cluster descriptions which specify the subspaces of each hyper-rectangle and the range of each attribute in the subspace. The attribute values for a data point assigned to a cluster are generated as follows. For those attributes that define the subspace in which the cluster is embedded, the value is drawn independently at random from the uniform distribution within the range of the hyper-rectangle. For the remaining attributes, the value is drawn independently at random from the uniform distribution over the entire range of the attribute. We add 90% of the specified number of points equally among the specified clusters, and the remaining 10% points are added as random noise. Values for all the attributes of these points are drawn independently at random from the uniform distribution over the entire range of the attribute.

3.5.2 Synthetic data results

We studied the performance of ROSCLIQUE(CLIQUE implemented using Record Oriented Storage structure) v/s AOSSCLIQUE(CLIQUE implemented using Attribute Oriented Storage Structure) algorithm by varying the number of records, the dimension of the data space(total number of attributes) and the dimension of the clusters. The values for ξ and τ, were set to 10 and 0.15 respectively.

Database size: Figure 3.7 shows the results of the experiments carried out by varying the number of records from 50,000 to 1,50,000. The dimension of the data space was selected as 100. The number of missing values contained in any record, have been generated randomly between 20 and 40. The dimension of the 3 clusters generated was 9.

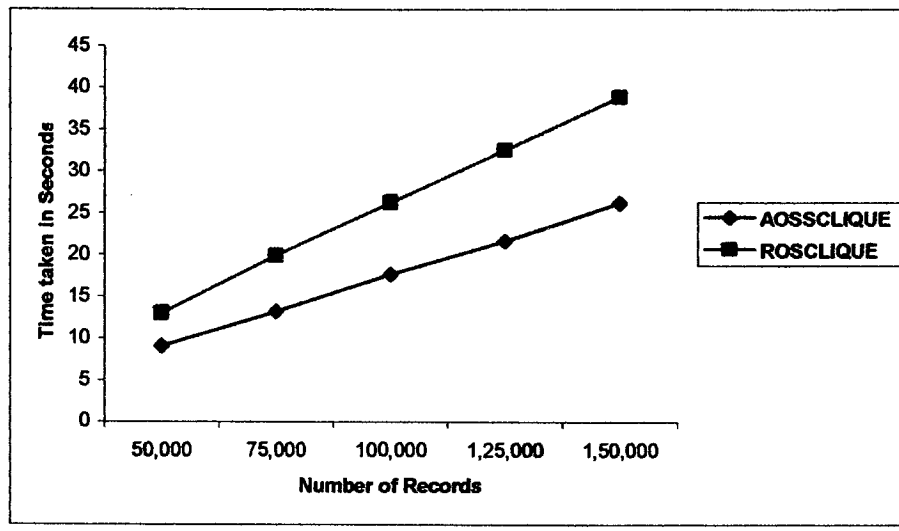


Figure 3.7: Scalability with the number of records(with missing values).

It can be observed from figure 3.7 that there is a significant improvement in the time taken for CLIQUE, when we used the AOSS method to store the data. The gain will be much more if the process of finding the 1-dimensional dense units of all the attributes is carried out in parallel during the first pass.

The same experiment was again repeated for the same set of data records this time containing no missing values. The only difference in the observations as expected was a proportionate increase in the time taken by both ROSCLIQUE and AOSSCLIQUE. The results are shown in figure 3.8 .

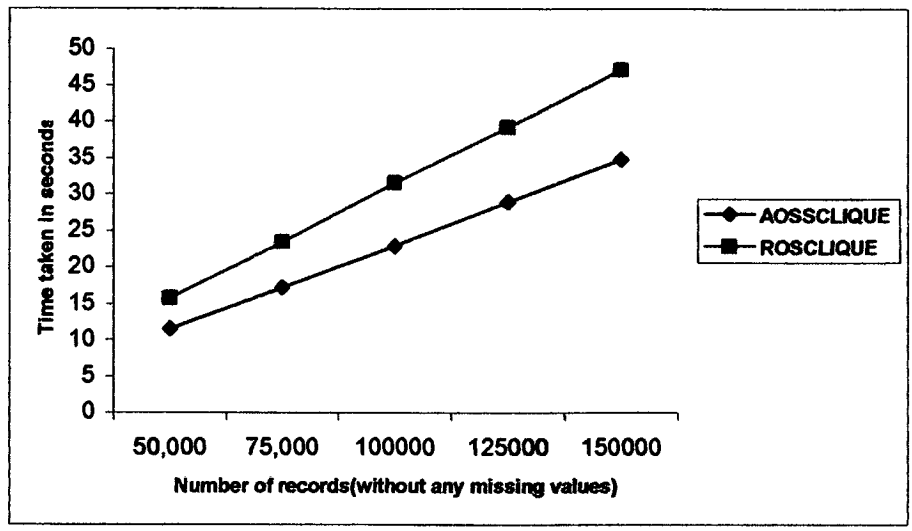


Figure 3.8: Scalability with number of records(without missing values)

Dimensionality of the data space: The next set of observations were taken by varying the total number of attributes(dimension of data space) from 50 to 150. The total number of records was selected as 50,000. Again the dimensionality of the 3 clusters chosen was taken as 9. And the number of missing values in any record was taken as a random number between 20 and 40. The results are shown in figure 3.9.

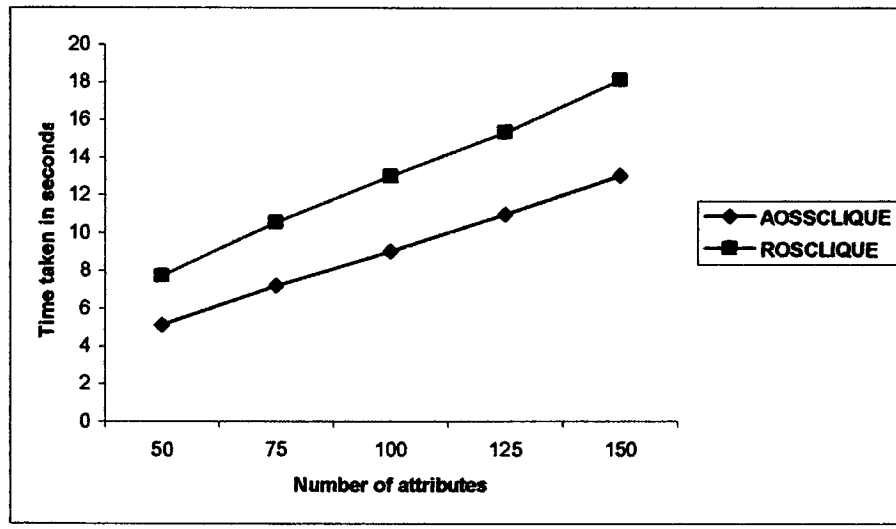


Figure 3.9: Scalability with the dimension of the data space.

Dimensionality of the clusters: Both the methods suffer when the dimensionality of the clusters increases, due to the inherent nature of the apriori algorithm which suffers from the curse of dimensionality. However, using the depth-first method to find the maximal dense units, to help in finding the subspace clusters with the AOSS structure has showed very good results, details of this are reported in chapter 5.

3.6 Summary

Although, a tremendous amount of research work has been carried out in clustering by the data mining community, it has been found that all these algorithms fail when it comes to finding clusters in very high dimensional huge datasets. It is

here that we realize the importance of subspace clustering algorithms. Although a few subspace clustering algorithms like *CLIQUE* [1], *MAFIA* [16], *ENCLUS* [8] have been designed, most of them suffer due to the way the data is stored. In this chapter, we have discussed a new framework for storing the data keeping in mind the requirements of the subspace clustering algorithms. In subspace clustering, we place more importance on the attributes of the data and do not want to miss out on any useful patterns that may exist across attributes. Hence we have developed the Attribute Oriented Storage Structure (AOSS) for storing the very high dimensional huge datasets and performed an experimental study, which showed the performance gain obtained using this method. The performance gain is mainly due to the different approach used to find the selectivity of the different units, which became possible due to AOSS. This approach helps in reducing the number of records actually accessed to find the selectivity of the different units. However, since CLIQUE is based on the apriori algorithm to find the dense units, AOSSCLIQUE also does not show improvement as the dimensionality of the clusters increases and the capability of this structure is not utilized to its full extent. Hence in chapter 5, we have demonstrated the efficiency of this structure using a depth-first method unlike the breadth-first method of apriori for high dimensional clusters.

Chapter 4

SAMCLIQ: A SAMpling based CLIQue Algorithm

We discussed the details of the first subspace clustering algorithm *CLIQUE[1]* in Section 2.2. We observed that, one of the major drawbacks of the algorithm is the repeated number of database passes required during step one to find the selectivity of the large number of candidate units that are generated. For very large databases, when the entire data cannot be loaded into the main memory at one time this step will require a tremendous amount of I/O to be done. Hence, if for example 10% of the data fits in the available main memory at a time then for one pass through the entire database, the data will have to be loaded in parts 10 times from the disk. And for k passes over the data, 10 * k loads will be required.

Can we improve the efficiency of the first step, to handle very large databases ? To address this problem, we developed an algorithm *SAMCLIQ* which uses a sampling based approach to find the dense units existing in the various subspaces of the data space. In this chapter, we first discuss in Section 4.1 the Use of sampling technique in data mining. In Section 4.2, we propose a sampling technique to get the sample of records from the original data space for finding the initial set of dense units. The details of the SAMCLIQ algorithm are presented in Section 4.3 and Section 4.4 reports the experimental results.

4.1 Use of Sampling in Data Mining

Sampling has played a very important role in data mining and has been mainly used to reduce the I/O activity required for knowledge discovery in large databases. In section 4.1.1, we explain the important role played by *sampling* in data mining. Section 4.1.2 explains some limitations of sampling and certain solutions to overcome them.

4.1.1 Role played by Sampling in Data mining

The application of sampling for mining association rules has been suggested in [21], and its effectiveness for mining association rules has been evaluated in [31]. It has been noted in [29] that samples of reasonable size provide good approximations for frequent sets. In [18], a general analysis on the relationship between the logical form of the discovered knowledge and the approximate sample sizes needed for discovering the knowledge has been studied. The role played by sampling in data mining has been well explained in [22] also. In the experimental evaluation carried out in [31], it has been shown that samples of reasonable size which fit in the main memory can be used with a reasonably high level of accuracy, to find the data patterns that exist in the database with high confidence.

4.1.2 Limitations of Sampling

To reduce the I/O activity, a random sample from the original database, small enough to be handled totally in main memory is drawn and the approximate regularities that exist in the original database are found out. These approximate results are then used to adjust parameters for a more complete knowledge discovery phase. Choosing sample sizes depending on the available main memory, *approximate results* can be obtained about the original database. However, we cannot be very sure that we have not missed out any data patterns that exist in the original database. And at the same time, if we do not include the right set of records in the sample we may get some patterns in the sample which actually do not exist in the original database.

Hence, in order to obtain the best results from the sample drawn it is important that we select a proper size for the sample and at the same time ensure that we select those records from the original database which help us in identifying in majority of the cases, all the patterns which exist in the original database. For this purpose we have developed a sampling technique for extracting a sample of data records from very high dimensional huge datasets, which is based on the AOSS method used for storing data. We discuss this method of sampling in Section 4.2.

In business and various other applications, where important decisions have to be taken based on the data patterns that exist in the databases, one cannot rely totally on the results obtained from sampling. Hence, as a tool for further analysis, the concept of negative border has been applied in many applications. The negative

border information has been used in [27], [13] and [23] to achieve efficiency in the incremental mining of association rules. In [23], Manilla and Toivonen have shown that the evaluation of the negative border units ensures that no frequent patterns are missed out. We have adopted the use of the negative border concept to ensure that we do not miss out any of the dense units, which were not present in the sampled records, but are actually found in the original database. More details about the negative border units have been discussed in section 4.3.

4.2 Proposed Sampling technique

In this section, we first discuss some criteria for a good sample under subsection 4.2.1 followed by subsection 4.2.2 which presents a brief discussion on the sampling method used in [29]. Section 4.2.3 discusses the AOSS based sampling technique.

4.2.1 Criteria for a good sample

The efficiency and the accuracy of the results obtained by using sampling, depends on the following two factors-

- *Sample size* – If the sample size selected is too small, compared to the size of the original database then there is a more chance of missing out the patterns found in the original data. And at the same time if the sample

size selected is very large, then we may not get any missed units but the actual purpose of sampling is lost. The size selected should be able to have a balance between the number of missed patterns generated and the extra time that we spent in processing the sample records.

- *Selection of good records* – The selection of a proper sample size is important, but choosing the right set of records for the sample is more important than this. Even if we choose a big sample size, but if most of the records selected are either outliers or noise points then we will fail to identify the correct patterns from the database.

Given a sample size n, we have designed a sampling algorithm which gets the best set of n points to help in identifying all the possible patterns from the original database of size N in majority of the cases. For this purpose, we assume that the original data has a very large number of attributes, and is very large in size such that the entire data does not fit in main memory at one time. The details of the sampling technique is explained in section 4.2.3.

4.2.2 Sampling for finding frequent sets

Till date, many algorithms have been designed for sampling but none of them address in specific, the issue of drawing a sample from a very high dimensional huge dataset. Most of them randomly pick up the points, without giving much importance to the quality of the points that are selected. A lot of the

algorithms have attempted to get the sample sizes for a required level of accuracy. In [29], Toivonen has used sampling for reducing the number of database passes required to find the frequent item sets to be used for finding the association rules from large databases. The performance study in [29] shows that after mining the sample, the sampling algorithm needs only one scan of the original database to find all frequent patterns. However, this algorithm does not focus on the selection of the points for the sample, but uses Chernoff bounds to determine the sample size required for a desired level of accuracy. This process of finding the sample size does not take into account the size N of the original database, hence many times if the accuracy level required is very high it may give a large sample size. Besides the algorithm has not paid much attention to picking the right set of points for the sample, since they were not dealing with very high dimensional data sets.

4.2.3 AOSS based sampling technique

In our proposed sampling technique we have focused on the selection of the points for the sample from those, which contribute to the formation of the various 1-dimensional dense units. The various steps are as under-

1) Using the user-input value for ξ, form the various *one dimensional candidate units* by splitting the range of all attributes into ξ intervals.

2) Using the AOSS attribute tables for all the attributes, the selectivity of all units is found out. The process of finding the *selectivity* for the candidate units of the various attributes can be carried out in parallel.

3) The 1-dimensional dense units are obtained by choosing those candidate units whose selectivity is larger than the user specified threshold value τ.

4) We retain only the record-id information of the 1-dimensional dense units. We call this set of record-ids the sample pool. Naturally this sample pool will be much smaller in size compared to the total data size. We select the points for the sample from this pool.

5) Sample selection –

   The details of the sample selection are discussed after example 4.1.

Example 4.1

Consider the following transaction table 4.1 consisting of 10 records. Each record has 6 attributes namely A, B, C, D, E and F. The values of all these attributes, are in the range of 1 to 100.

TID	A	B	C	D	E	F
T1	1	21	41	0	4	18
T2	4	24	44	9	9	49
T3	6	26	46	45	23	83
T4	9	29	49	57	56	5
T5	2	8	25	58	78	30
T6	53	9	92	59	79	52

T7	19	8	89	58	78	57
T8	82	2	98	52	72	12
T9	89	78	10	25	2	38
T10	68	68	75	1	62	13

Table 4.1: A sample of 10 transactions consisting of 6 attributes

Consider a threshold value of 0.4 and the number of intervals equal to 10. The various *units* will have the following range values-

Unit 1: 1-10 ,

Unit 2: 11-20,

⋮

Unit 10: 91-100

The various 1-dim candidate units formed are as under –

A1, A2, ... , A10,

B1, B2, ... ,B10,

⋮ ⋮ ⋮

F1,F2, ... , F10.

After pass one through the transactions in the table 4.1, we get the following dense units –

unit	*freq. count*	*TID lists*
A1	5	T1, T2, T3, T4, T5
B1	4	T5, T6, T7, T8
B3	4	T1, T2, T3, T4
C5	4	T1, T2, T3, T4
D6	5	T4, T5, T6, T7, T8
E8	4	T5, T6, T7, T8

From the 1-dimensional dense units, the following 2-dimensional candidate units will be generated -

(A1 B1),

(A1 B3),

... ,

(A1 E8),

(B1 C5),

... ,

(B1 E8),

... ,

(D6 E8).

After second pass we get the following 2-dim dense units -

(A1 B3) : 4,

(A1 C5) : 4,

(B3 C5) : 4,

(B1 D6) : 4,

(B1 E8) : 4,

(D6 E8) : 4.

After third pass we get the following 3-dim dense units -

(A1 B3 C5) : 4

(B1 D6 E8) : 4

Given above are the various steps carried out in step one of the *CLIQUE algorithm.*

The **Sample Selection procedure** is as follows -.

1) find all one-dimensional dense units and their tid-lists(record-ids)
2) group transactions(record-ids) based on number of such 1-dim dense units they are contained in and have these groups sorted in descending order of the record-id counts.
3) Choose a proportionate number $f / t_f * S$, of record-ids randomly from each group in the sorted order. f represents the number of dense-units, t_f the

count i.e the number of record-ids present in that group and S the desired sample size.

Grouping of record-ids based on number of 1-dim dense units they are present in, for the data in example 4.1 above this will be as follows –

No. of units	Record-ids	count
3	T1, T2, T3, T6, T7, T8	6
4	T4, T5	2
5	-	0
6	-	0

Assume sample size = 4. Randomly select any 4 record-ids from group with number of unit equal to 3. If number of record-ids in unit 3 is less than 4 then select from remaining units i.e 4 in this case. This process ensures that we get good set of records for the sample i.e records containing dense units.

4.3 Subspace Clustering Using Sampling

The first step of the *CLIQUE[1]* algorithm is quite complex for huge datasets having high dimensional subspace clusters. If k is the highest dimensionality of any unit that is found than it will require as many database passes over the data as equal to the highest dimensionality of any dense unit in the data. Hence in order to reduce the number of database passes and the I/O required, we have developed the SAMCLIQ algorithm. The *SAMCLIQ* algorithm basically tries to improve the performance by using an efficient sampling technique for identifying the dense units in step one of *CLIQUE.* After selecting the sample using the method discussed in section 4.2.3, we find all the dense units in the sample using the method discussed in section 4.3.1. After getting the dense units from the sample, we use the concept of negative border units to make sure that we have not missed out on any units, which are present in the original data space.

The Negative Border N, consists of all the candidate units generated in the level-wise algorithm that were not dense units. In other words if C is the set of all the candidate units generated, D is the set of dense units then $C = D \cup N$. After obtaining the results using sampling, we want to make sure that we have not missed out any units, which are dense in the original database but were not detected in the sample. Obviously the subsets of all such likely missed units will be found in the negative border N of the sample.

4.3.1 Algorithm for identification of dense units

The Algorithm we propose requires an initial pass, for selecting the sample during which it generates the 1-dimensional dense units also. After selecting the sample we apply the level wise algorithm used in step one of CLIQUE to get all the candidate and dense units present in the sample. Then a first pass over the original database (O. D) is carried out to find if any units are missed out, by using sampling. If any units are missed then an additional pass is made over the O.D. This work was carried out prior to the development of the AOSS method. Hence, in this chapter we have not used it as such for the main algorithm. The details of the algorithms are as under

Sampling for identification of dense units – The accuracy of the results obtained using sampling, to a large extent depends on the size of the sample and the method used to select the sample points from the database. Since we are considering very large databases and we want our sample to fit in main memory, we choose sample size s such that it is neither too large and nor too low by using the technique discussed in section 4.2.3. We know that, with increasing sample sizes the probability of finding the dense units identified in the sample, in the original database also are high and thereby the possibility of occurrence of false dense units and of missed units are almost negligible. Hence, an extra pass over the database will not be required, but if the sample size is too large, then the time taken to process the sample is very large compared to the gain in performance achieved by reducing

the number of passes. In place of CLIQUE, other subspace clustering algorithms like MAFIA [16] can also be used. Depending on the quality of sample selected, there are three cases that we can encounter -

a) There may be some units, which were dense in the sample but are not dense in the original database. In such cases we have unnecessarily counted such units, we will call such units as false dense units. These false dense units get discarded after the first pass over the original database and do not affect the accuracy of the final results obtained for the original database.

b) There may be some units, which were not dense in the sample but are dense in the original database. In such a case, we say that there has been a miss i.e., we have missed to capture these units and some higher-level units of these in the sample. There are two types of misses that we may come across first type is where we fail to capture some dense units in some subspaces and second wherein some subspaces containing dense units were fully missed. Whenever there are such missed units say M, then some higher level candidate units say C1 generated using M may be dense in the original database. But this set C1 would not be generated by the sample, hence there is a need to generate higher level candidate units of such missed units and evaluate them i.e find their counts in the original database by doing an additional pass over the database.

c) The units which were dense in the sample are dense in the original database also and vice-versa. This is an ideal case and gives the best performance if the sample size is selected properly i.e it is not too large, but at the same

time very closely resembles the original database. In this case the results will be obtained by doing a single pass over the original database, besides the initial pass required to draw a sample and find 1-dimensional candidate and dense units for the original database.

From above we observe that it is case (a) and case (b) that needs to be handled properly. To handle case (b) one method that is discussed in [29] is to lower the density threshold value, while generating the candidate units for the sample. This will definitely reduce the chances of a miss, but will lead to an increase in the number of false dense units. The aim to avoid the misses, is to achieve the results in just one pass. If there are missed units, then two complete passes will be required over the original database. Another method to reduce the number of passes to less then two complete passes is to adopt the technique used in [7]. Instead of waiting for the end of the first pass to find the missed units, we check for missed units after every M records have been processed and generate the higher level candidate units for such missed units and start counting the occurrence of these units from that point onwards. If all the missed units were detected, after x number of transactions were processed during the first pass, then we will need one complete pass and scan only the un-scanned x transactions during the second pass for the missed units in C1. We will need two complete passes only when we have found missed units towards the end of the first pass. This will normally happen if the data is very correlated. The value for M should be selected carefully, in such a way that there is not much of processing overhead.

Algorithm for Generating the dense units in the Sample drawn :

Inputs:

- The Sample points of size s from the original database (O.D),
- number of attributes(dim),
- density threshold τ,
- set C[1] of 1- dimensional candidate units
- set D[1] of 1-dimensional dense units obtained from the sample.

Outputs:

- set C of candidate units in the sample,
- set D of dense units obtained from the sample data records
- the 1-dimensional dense units of O.D.

Processing:

1. Use D[1] to find C[2] set of 2-dimensional candidate units; // this avoids the chances of a 1-dimensional missed unit
2. While more candidates are generated

```
{
find selectivity of C[k] in the sample ;   // for k >= 2 and <= dim
find D[k] = dense units in sample from C[k];
generate C[k+1] from D[k];      // the candidate generation procedure
                                   used in CLIQUE is used.
```

```
++k;
}
```

Algorithm for the first_pass *// to find if any missed units are found*

Inputs:

- set C and D obtained from the sample,
- the Original Database O.D,
- the threshold value τ.

Outputs:

- set D1 set of dense units in the O.D
- set C1 set of missed units.

Processing:

1. Num_parts = N/BUFFSIZE;
2. Initialize counts of all units in C to 0;
3. for (n = 1; n<= Num_parts; n++)
   { read BUFFSIZE records into main memory buffer[BUFFSIZE];
   update counts of units in C using buffer[BUFFSIZE];
   }
4. find D1 = set of dense units in O.D ; // those units from C whose count is >= N* τ

5. find M - missed units by comparing D and D1;

6. if M is empty

goto step 9; //stop

else

find C1 = set of all candidate units formed from M and D units ;

// C1 is the candidate units missed in the sample , which may be dense in the O.D

7. Do a second pass through the O. D ; // required only if there are missed units .

8. The set D1 consists of all the dense units in the Original Database (O.D).

9. Stop

This forms the input to the second step of the subspace clustering algorithm.

Algorithm for the second pass

Inputs:

- set C1 // the set of all candidate units formed from M and D units obtained from first pass,
- O.D,
- the threshold value τ.

Output:

- the final set D1 of O.D // all dense units of O.D

Processing:

1. Find the counts of all units in set C1 in O.D.
2. Find additional dense units obtained from C1 and add to set D1;
3. Stop.

4.4 Experimental Results

In this section we present an empirical evaluation of the above algorithm, which we call SAMCLIQ (SAMpling based CLIQue) algorithm using synthetic datasets. The goal of the experiments was to compare the performance of step one of CLIQUE with the step one of SAMCLIQ. The MDL pruning used in step one of CLIQUE is not used in our implementation of CLIQUE. We compared the performance by varying the size of the database, dimension of the data space and dimension of the Clusters. The experiments were run on a 800 MHz Intel Pentium III processor running linux.

4.4.1 Synthetic data generation

The synthetic data generation method described in [1] has been used for the data generation. The data generator takes as input the number of records to be generated, the number of attributes and the range of values for each attribute. The range of values was set to [0,100] for all attributes. The clusters are hyper-rectangles

in a subset of dimensions such that the average density of points inside the hyper-rectangle is much larger than the average density in the subspace. The cluster description details provided by the user include the number of clusters, the maximum dimensionality of the clusters, and the cluster descriptions which specify the subspaces of each hyper-rectangle and the range of each attribute in the subspace. The attribute values for a data point assigned to a cluster are generated as follows. For those attributes that define the subspace in which the cluster is embedded, the value is drawn independently at random from the uniform distribution within the range of the hyper-rectangle. For the remaining attributes, the value is drawn independently at random from the uniform distribution over the entire range of the attribute. We add 90% of the specified number of points equally among the specified clusters, and the remaining 10% points are added as random noise. Values for all the attributes of these points are drawn independently at random from the uniform distribution over the entire range of the attribute.

4.4.2 Synthetic data results

We studied the performance of CLIQUE v/s SAMCLIQ algorithm, by varying the following parameters the database size, the dimension of the data space and the dimension of the clusters. The values for ξ and τ, were set to 10 and 0.15 respectively.

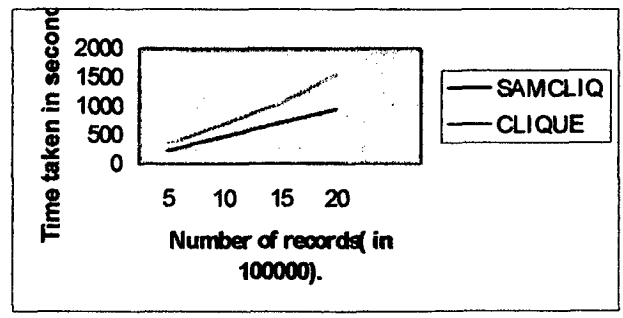


Figure 4.1: Scalability with the number of records.

Database size: Figure 4.1 shows the results of the experiments carried out by varying the database size from 5,00,000 records to 20,00,000 records. The sample size was selected as 1% of the database size and the main memory buffer size was taken equal to the space required to load 50,000 records. The data space had 50 dimensions. We found that the difference between the time taken by CLIQUE and SAMCLIQ increases significantly with the increase in the database sizes.

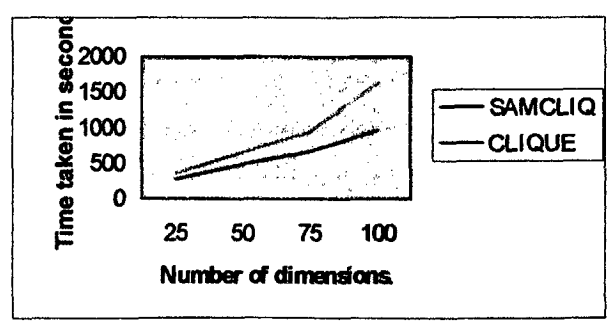


Figure 4.2: Scalability with the dimension of the data space.

Dimensionality of the data space: Figure 4.2 shows the scalability as the dimensionality of the dataspace is increased from 25 to 100. The experiments were carried out with a database containing 10,00,000 records. There were 5 clusters each in a different 7 dimensional subspace. The sample size selected for SAMCLIQ was 5% of the database size.

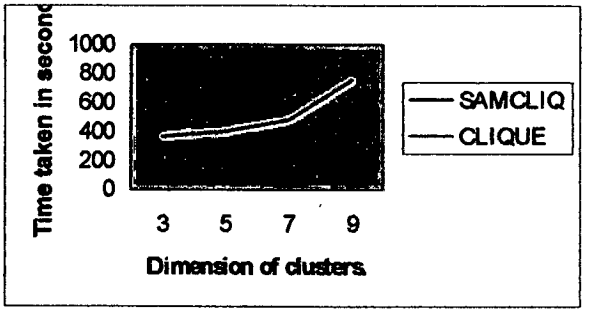


Figure 4.3: Scalability with the dimensionality of the clusters.

Dimensionality of the clusters: Figure 4.3 shows the scalability as the highest dimensionality of the clusters embedded in the different subspaces is increased from 3 to 9. Again the database size was selected to be equal to 10,00,000 records and the sample size was taken equal to 5% of the database size.

4.5 Summary

We have used the CLIQUE algorithm as the base on which the SAMCLIQ algorithm has been developed. But this can also be used in other subspace clustering

algorithms like MAFIA to further boost the performance. Our algorithm can easily be implemented as a parallel algorithm. After the first pass through the data base and the generation of candidate units using the sample, the data base can be split into n parts and the counts of all the candidate units can be computed in parallel in all the subparts. Then the counts in all n parts can be summed up and if there are missed units then again the counts for the missed units can be found in parallel during the second pass in SAMCLIQ.

We compared the performance of SAMCLIQ with CLIQUE by varying the database size, the dimension of the data space, and the dimensionality of the clusters. We found that there is a significant gain in performance when we use SAMCLIQ for large databases and higher dimensional data spaces. As we vary the dimensionality of the clusters, also the performance of SAMCLIQ is much better compared to CLIQUE but for very high dimensional clusters, the performance of SAMCLIQ also suffers because of very large number of candidate units produced. Hence, there is a need to use a different technique to find the high dimensional dense units in such cases.

The different techniques that can be used for this purpose are to make use of the concepts of g-closed itemsets, use FP-tree for generating dense units, or use of maximal frequent itemsets for finding the subspaces. The performance can be significantly improved by using the AOSS method for storing the high dimensional huge datasets, which gives the freedom to access only those attributes or records, which are needed during the process of finding the selectivity of the various candidate units.

Chapter 5

MLSCLUS: A Multi Level Subspace CLUStering Algorithm

In Chapter 3, we developed AOSS method to store very high dimensional huge datasets to be used mainly for subspace clustering. Although AOSS method is more efficient than the old record based method of storage, to handle the very high dimensional huge datasets with many missing values, it will fail to produce the best results in some cases, as illustrated below.

- *Not many missing values are found.* AOSS method stores the data in various AOSS Attribute tables by splitting the original record-based database to save on the space required by the many missing value attributes, that are most common in the very high dimensional datasets. We know that in each of the entries of the AOSS Attribute tables, we store the record-ids along with the respective attribute values. If missing values are not common, then in the AOSS method the additional space required to store the record-ids along with the attribute values in all the AOSS attribute tables will be significantly large.
- *Real datasets contain a combination of all types of attributes.* Majority of the existing subspace clustering algorithms use a uniform threshold value for all the

attributes. By using a single threshold value for all attributes, we tend to give equal importance to all the attributes. In reality, the threshold should not be uniform. Certain attributes are exceptional and need to be handled in a different manner. These exceptional attributes either have a much lower threshold requirement, or need a very high threshold value depending on whether they are rare or very frequent. Thus a uniform threshold for all attributes might lead to either generation of uninteresting subspace clusters for frequently occurring attributes and at the same time miss out some interesting subspace clusters of rare attributes.

- Applications with Huge datasets need more scalability. Existing methods are not efficient when the dataset is very large. This problem was addressed in chapter 4 and the *SAMCLIQ [36]* algorithm designed for this purpose. The *SAMCLIQ* algorithm succeeded in reducing the number of passes, but since it used the old record based method of storage and the level wise apriori based method for generation of dense units it fails just like the other methods when the dimensionality of the subspace clusters is high. This was also observed from the experimental results of *SAMCLIQ*, in section 4.4

In this chapter, we present algorithms to address some of the above issues. They differ from the CLIQUE, SAMCLIQ and AOSSCLIQUE methods addressed in the previous chapters of this thesis in the following respects -

- Based on the AOSS method presented in chapter 3,
- Use different threshold values for the different attributes,
- Use the concept of maximal dense units for identification of subspace clusters,
- Mine the knowledge about the subspace clusters at different levels.

All these features are required in order to do a thorough analysis of data in any application area like census data analysis, and classification of web documents involving huge datasets with a very large number of attributes.

The remainder of the chapter is organized as follows. In Section 5.1, we discuss the use of the maximal frequent itemsets concept for finding all the dense units in first step of subspace clustering algorithm. Section 5.2 presents the details of the algorithm MADUGEN designed to find all the maximal dense units in a given dataset using uniform threshold for all attributes. In subsection 5.2.1 of this section we present the experimental results obtained using MADUGEN. In section 5.2.2 we present an algorithm MADUGENMT(MAximal Dense Unit GENeration with Multiple Thresholds), to find the maximal dense units using different threshold values for different attributes. We present in section 5.3 *AOMLSCLUS*, the Attribute Oriented Multi Level Subspace CLUStering algorithm, which uses concept of maximal dense units to identify sub subspace clusters in very high dimensional huge datasets,

consisting of attributes with varied threshold requirements and uses a variable number of intervals instead of using the same value for all attributes. The experimental results of AOMLSCLUS and AOMADUGENMT are reported in section 5.3.1. AOMADUGENMT is again a variation of CLIQUE implemented using MADUGENMT in step 1 of CLIQUE. Due to the use of variable intervals for different attributes, AOMLSCLUS can be enhanced further even for processing categorical attributes. In case of categorical attributes, the number of intervals will be equal to the number of unique categorical values of that attribute.

5.1 Use of Maximal Frequent Itemsets in Subspace Clustering

The apriori algorithm used for finding the frequent itemsets has the following main drawback –

- It employs a bottom-up search that enumerates every single itemset. Hence, in order to produce a frequent itemset of length k, it must produce all 2^k of its subsets since they too must be frequent. This exponential complexity of the algorithm restricts it to discovering only short patterns in medium sized datasets. To address this problem, the concept of maximal frequent itemsets [33] [34] [35] was introduced.

Definition 5.1: A *frequent itemset* is a set of items appearing together in a number of database records meeting a user-specified *threshold.* For example, if X is a k-itemset (an itemset consisting of k items), then X is frequent iff all the items found in X occur in atleast minsupport number of records, where minsupport is equal to *threshold* multiplied by *total number of records* in the dataset.

Definition 5.2: If X is a frequent itemset and no superset of X is frequent, then we say that X is a maximal frequent itemset.

Any frequent itemset Y which is not a maximal frequent itemset, will be a subset of some maximal frequent itemset X of the dataset. Hence the set of all maximal frequent itemsets present in a dataset, concisely represents all the frequent itemsets present in that dataset.

The first step of the subspace clustering algorithm, needs to find all the dense units in order to identify the subspaces containing clusters. CLIQUE [1] uses a level wise apriori based algorithm to generate the dense units and suffers from the same drawback as the apriori algorithm. The efficiency of this step can be significantly improved if we apply the maximal frequent itemset concept to the dense units. The concept of dense units has been explained in chapter 2, but we redefine them again

considering our AOSS representation and then we proceed to define maximal dense unit.

Definition 5.3 A k-dimensional unit u^k is defined as the collection of the units from each of k distinct attributes. It has the form $u^k = \{u_1, u_2, \ldots, u_k\}$ where u_i is the $<A_i, I_i>$ pair of the i^{th} attribute A_i present in u^k, I_i is a integer value representing the interval to which the attribute belongs to.

Definition 5.4 The frequency count of a unit u^k in the original database DB is equal to the number of record-ids common to all the AOSS attribute tables of the k attribute units present in u^k.

Definition 5.5 The minimum support value *msv* of a unit u^k in the original database DB *msv* (u^k, DB) = N * min { τ_i of $A_i \in u^k$, i = 1 to k }where N is the number of records in DB and τ_i is the density threshold value of the i^{th} attribute of u^k, expressed as a percentage of records expected in each unit of the attribute for it to be dense.

Definition 5.6 A k-dimensional unit u^k is said to be a dense unit in the original database DB, if the frequency count of this unit in the original database DB is greater than or equal to the minimum support value *msv* (u^k, DB) of the unit in DB.

Definition 5.7 If X is a k-dimensional dense unit and no m-dimensional superset of X where m > k, is dense, then we say that X is a *maximal dense unit*.

In order to improve the efficiency of the first step we can find all the maximal dense units and use them to find the subspace clusters present in the dataset. In the next section we present an algorithm for the same.

5.2 MADUGEN: A MAximal Dense Unit GENeration algorithm.

In this section, we present the MADUGEN algorithm to find all the maximal dense units present in a dataset containing attributes with a uniform threshold value. We have designed this algorithm using the AOSS method of representing the data. It is based on the GenMax [34] algorithm used to find the maximal frequent itemsets for association rule mining.

Notations and Terminology used –

unit - is a pair of integers representing the attribute-id and unit-id respectively.

iset - it is a collection of attribute-id and unit-id pairs(units) belonging to the one dimensional dense units. An iset of length k consists of k such units.

len – used to keep track of number of units in iset.

pset(u^k) - pset of a k-dimensional unit u^k denotes the possible set of u^k and consists of all the units from the one-dimensional dense units D[1] , which are candidate units for forming higher dimensional dense units with u^k as the base unit. The units in D[1] are sorted in ascending order based on attribute-id as primary key and unit-id as secondary key.

plen - is used to keep track of the number of units in *pset(u^k)*.

cset(u^k) - cset of a unit u^k denotes the combine set of u^k and is a subset of pset(u^k) consisting of only those units from pset which form k+1 dimensional dense unit when combined with u^k . This helps in pruning those units from pset which are not candidates for forming higher dimensional dense units.

ccnt - ccnt used to keep track of the number of units in the *cset(u^k)*.

mduset - consists of all the maximal dense units from the given data set.

mducnt – used to keep track of the number of maximal dense units obtained in the data set.

threshold value and *one-dimensional dense units* have the same interpretation as used in chapter 2.

MADUGEN Algorithm

Input:

- D[1] – details of one dimensional dense units obtained in the dataset,
- threshold value,
- record count (total number of data records in the dataset),

Output:

- mduset – set of maximal dense units found,
- mducnt – number of maximal dense units found in the dataset.

Processing method:

1. Start
2. initialize mduset to empty and mducnt to 0
3. for each subspace unit u in D[1]
   a. initialize iset to u
   b. call findpset(D[1], u) // findpset used to find the pset of unit u.
   c. call findmdu(mduset, iset, len, pset, plen, threshold)
   // findmdu is a recursive function to find maximal dense units.
4. // endfor
5. Stop // end of MADUGEN algorithm

The details of the algorithm used for findpset and findmdu are described in the following pages.

Algorithm for findpset

Inputs

- D[1] – one dimensional dense units
- unit u – consisting of its attribute-id and unit-id

Output

- pset of u – set of all units in D[1], whose attribute-id is greater than attribute-id of unit u.
- plen - number of units in pset.

Processing method

1. for each unit u1 in D[1]
   if attribute-id of u1 > attribute-id of u
   add unit u1 to the pset
2. return pset

Algorithm for findmdu

Inputs

- iset
- len
- pset
- plen
- threshold

- record-count

Output

- mduset – set of all maximal dense units.

Processing method

```
1.  if pset is not empty
2.    begin
3.          call findcset(iset, len, pset, plen, mduset, threshold, record-count)
// finds the combine set - cset, detailed algorithm for findcset described after this
algorithm.
4.      if cset is empty
5.          call addiset(mduset, iset, len)    // detailed algorithm for addiset
described after algorithm for findcset.
6.      else
7.        begin
8.          for all the units cu in cset
                with attribute-id = attribute-id of first unit in cset do
9.              begin   //for
10.             // form new isets by extending iset with cu.
11.                 iset = iset + cu
12.                 cset = cset – cu
13.             end   //for
14.         len = len +1  // length of iset increased by 1
15.         for all the newisets obtained in step 8 do
16.           begin  //for
```

17. call findmdu(mduset, iset, len, cset, ccnt, threshold, record-count)
// pset = cset in above step 17.
18. end // for
19. end // else of if cset empty
20. end // if pset not empty
21. call addiset(mduset, iset, len)
22. Stop // end of findmdu.

Algorithm for findcset

Inputs

- iset
- len
- pset
- plen
- mduset
- threshold
- record-count

Output

- cset

- ccnt

Processing method:

1. for each unit u in pset
2. newiset = iset + u // extend iset by adding u
3. check if newiset exists in mduset
4. if exists goto step 1 // continue with next unit u from pset
5. if not find frequency count of newset in the dataset
6. if frequency count > threshold * record-count
7. add u to cset
8. increment ccnt
9. stop

Algorithm for addiset

Inputs

- iset
- len
- mduset

Output

- mduset

- mducnt

Processing method

1. add iset to mduset.
2. increment mducnt
3. stop

5.2.1 Experimental results

In this subsection we present the comparison of time requirements of the ROSCLIQUE, AOSSCLIQUE and AOMADUCLIQUE. AOMADUCLIQUE is the implementation which uses MADUGEN algorithm in step one of CLIQUE to find the maximal dense units and uses only the maximal dense units to find the subspace clusters, instead of using all the dense units. We compared the performance by varying the dimension of the clusters from 5 to 12 using dataset of size 50,000 with 100 attributes. The values for the threshold and number of intervals, was set to 0.15 and 10 respectively. For generating synthetic data the method discussed in section 3.4.1 was used. The experiments were run on a 3.00GHz Pentium 4 processor running linux. The results obtained are shown in figure 5.1.

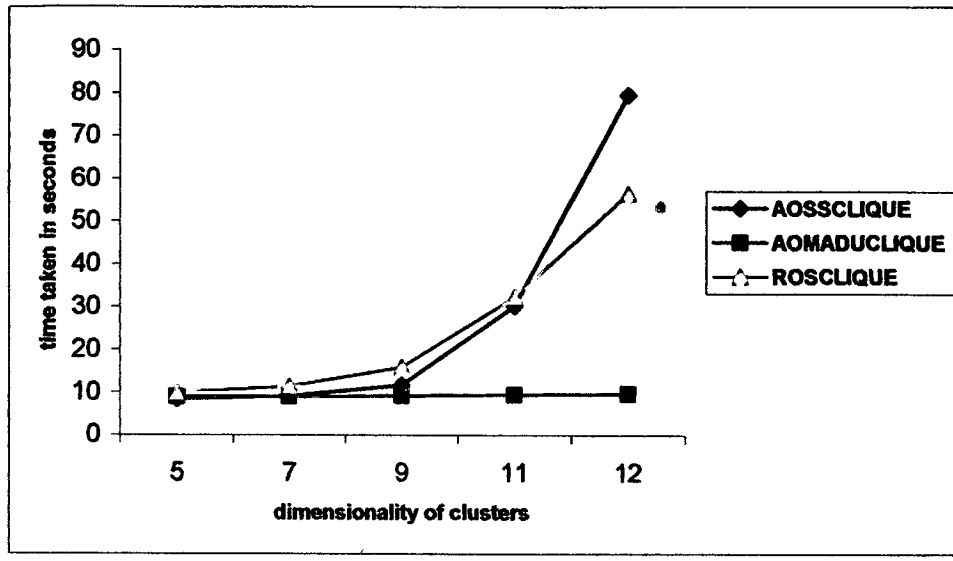


Figure 5.1: Scalability with the dimension of the clusters.

5.2.2 MADUGENMT: MADUGEN algorithm with multiple threshold values.

In this section we discuss, how we have modified MADUGEN algorithm to handle attributes with varying threshold values and also discuss the experimental results obtained using MADUGENMT.

In all the discussions earlier we have seen that the downward closure property is satisfied by all the dense units i.e if a particular k-dimensional unit u^k is dense then all its subsets are also dense. This property is no longer satisfied when the attributes have different threshold values. In MADUGEN, D_t is used to generate the maximal dense

units, but here if we use only D_1 we fail to generate all the maximal dense units, that is all those units possible with the low threshold value attributes. Hence we use the seed set SS, which is generated using D_1 and C_1. The seed set SS is generated as follows-

Seed Set Generation -

1. sort all the 1-dimensional candidate units C_1 based on the threshold values of the attributes in ascending order.
2. D_1 in this multiple threshold value case consists of all the units whose frequency count is greater than or equal to the minimum support value calculated using the threshold value of the attribute representing the respective units. Again in D_1 all the units are kept sorted in ascending value of the threshold values of the attributes.
3. Using the sorted order of attributes in C_1, find the first attribute unit fu in C_1 which belongs to D_1 and insert it into seedset SS.
4. for each subsequent attribute unit su in C_1 which comes after fu, whose frequency count is greater than or equal to the msv(fu, DB), insert su into SS.

All the maximal dense units generated using SS ensure that these dense units satisfy the sorted closure property. The sorted closure property ensures that we do not miss out any dense subsets of the low threshold attributes. Another variation that is required is that in step 6 of findcset function, in MADUGEN we had just one threshold value to decide whether to add it to cset, but in this case we will be having a maximum

of k different threshold values if iset is of length k. From these k different values, we use the threshold value of the first unit in iset as that will be the lowest threshold value. In MADUGEN, findpset function uses D[1] the set of all one-dimensional dense units D_1 to find the units in pset, but in MADUGENMT we use the seed set SS obtained as explained above and pset is obtained as follows-

Processing method // for pset(u^k)of MADUGENMT

1. for each unit u1 in SS
   if attribute-id of u1 is not contained in any unit of u^k
   add unit u1 to the pset
2. return pset

MADUGENMT Algorithm

Input:

- C[1] – details of one dimensional candidate units of all attributes after finding their frequency counts
- D[1] – details of one dimensional dense units obtained in the dataset
- thresholdarray // storing threshold values of all attributes
- record count (total number of data records in the dataset)

Output:

- mduset – set of maximal dense units found,
- mducnt – number of maximal dense units found in the dataset.

Processing method:

1. Start
2. initialize mduset to empty and mducnt to 0
3. find seed set SS as explained earlier in this section under *Seed Set Generation*
4. sort D[1] in ascending order of threshold values of attributes
5. for each subspace unit u in D[1]
   a. initialize iset to u
   b. call findpset(SS, u) // findpset used to find the pset of unit u.
   c. call findmdu(mduset, iset, len, pset, plen, thresholdarray, record-count)
   // findmdu is a recursive function to find maximal dense units.
6. // endfor
7. Stop // end of MADUGENMT algorithm

5.3 AOMLSCLUS: An Attribute Oriented Multi Level Subspace CLUStering Algorithm.

The subspace clustering algorithms that we have discussed so far find only the subspace clusters, which are found in the original dataset. In this section, we define the subspace clustering problem for attributes with different threshold and interval values, which also identifies all the sub subspace clusters found in the subspace clusters of the original dataset. We also report some experimental results obtained.

Problem statement:

Given a set of say n records $R_1, R_2, \ldots, R_n$ each record R_j, j = 1 to n having d attributes where $R_j = \{ <A_i, V_i>, i = 1 \text{ to } d\}$ where A_i is the ith attribute and V_i is the value of the ith attribute and given user input interval values ξ_i for i = 1 to d, for the different d attributes the problem is to find the clusters in all the subspaces of the original data space and also identify those sub clusters which are not found in the original data space by the one dimensional dense units, but are found in the subspace clusters identified in the original data space.

The algorithm for this problem has been named as AOMLSCLUS, and is implemented using the AOSS data representation, the MADUGENMT algorithm and the findthreshold algorithm which we have designed to find the threshold values of the attributes in the dataset. The details of the findthreshold algorithm are discussed in the next subsection. We used the AOSS method discussed in chapter 3 to store the data records. All the record details are stored in the AOSS record table. The record details include the attribute-id and the attribute-record id pairs for all attributes of the record containing non-missing values. The attribute values along with the record id's are stored in independent AOSS attribute tables. The Attribute-details table stores the information about all the attributes of the dataset. The Attribute-details table includes following information about each attribute –

- Attribute-id
- Attribute-name

- Attribute-filename
- Attribute-type(Numeric/Categorical)
- Attribute-Num-Intervals(number of intervals for the attribute)

The number of intervals is input by user for each attribute. Our implementation of the algorithm splits the attribute values into equal sized intervals and works for numeric attributes. In order to provide for variable interval sizes, the lower and upper range for each of the intervals can also be stored in case of numeric attributes and all distinct values of the categorical attributes can be stored to handle the categorical attributes.

This problem can be decomposed into the following three main parts –

1. Identification of the maximal dense units in the original dataset DB.
2. Identification of the maximal sub dense units in the various maximal dense units identified in DB.
3. Presentation of the details regarding the maximal dense and maximal sub dense units to the user.

The details of the above three steps are explained in the following subsections-

Identification of the maximal dense units in the original dataset DB The maximal dense units found in the original dataset are found using the MADUGENMT algorithm. The different threshold values for all the attributes are found using the following algorithm-

findthreshold algorithm

Inputs

- frequency counts of each attribute in all its units

// number of units for each attribute is equal to the number of intervals for that attribute.

Output

- threshold value for each attribute

Processing

1. for each attribute repeat following steps -
2. find the average frequency count, *avgfcount*

// avgfcount = sum of frequency counts in all its units divided by the number of units for that attribute.

3. find new average frequency count, *navgfcount* by considering only those units whose frequency counts are greater than the *avgfcount*
4. find the standard deviation, *stddev*

// stddev = squareroot of (sumofsquares(difference between navgfcount and frequency count) of each unit)

5. threshold of the attribute = (navgcount – stddev) / sum of frequency count in all units of that attribute.
6. stop

It has been observed that, those attributes, which do not contain any significant clusters do not show much variation in their frequency count values in the various units and as a result have a low value for the standard deviation. But in the other case, where there are clusters there is a wide variation in the frequency count values of the different units, and have a much higher value for the standard deviation. By finding the new average frequency count in step 3 of findthreshold and using it to find the threshold value, we try to eliminate the low frequency count units of that attribute by setting a higher threshold value for that attribute. If we just use the average frequency count obtained in step 2, we get more or less the same threshold value for all attributes having the same number of intervals. Using the threshold values obtained as above, we find all maximal dense units and use these to find the subspace clusters in DB. The user can also be given the freedom to input their own threshold values for the attributes at the time of entering the attribute details.

We define below certain terms, which are used in the next subsection.

Definition 5.8 The minimum support value msv of unit u^p in a dense unit u^k is defined as

$$msv(u^p, u^k) = M * \min\{ \tau_i \text{ of } A_i \in u^p, i = 1 \text{ to } p \}$$

where M is the frequency count of the u^p unit in unit u^k and τ_i is the density threshold value of the i^{th} attribute of u^p, expressed as a percentage of records expected in each unit of the attribute for it to be dense.

Definition 5.9 We define the frequency count of a unit u^p in another unit u^k as equal to the number of record-ids common to all the AOSS attribute tables of the (k+p) attribute units. This is used to find the sub dense units

Definition 5.10 We define a *sub dense unit* as one which is not dense in the original dataset DB but is dense within the maximal dense unit identified in DB. A unit u^p is dense in u^k if frequency count of u^p in u^k is greater than or equal to $msv(u^p, u^k)$.

Definition 5.11 If X is a k-dimensional sub dense unit in unit u^k and no m-dimensional superset of X where m > k, is sub dense in u^k, then we say that X is a *maximal sub dense unit* of u^k.

Identification of the maximal sub dense units in the various maximal dense units identified in DB.

In order to find the sub subspace clusters in the subspace clusters identified above, we need to first find all maximal sub dense units found in the maximal dense units identified in above step. We have designed algorithm findsubmdu which uses the findpset and findmdu functions of MADUGENMT algorithm for this purpose. The details are as follows –

findsubmdu algorithm

Inputs

- mduset // output of step 1 obtained using MADUGENMT
- mducnt

Output

- mdusubset // maximal dense units found in the mduset of step 1
- mdusubcnt // number of sub dense units identified

Processing

1. for each unit mdu belonging to mduset
2. record-count = frequency count of mdu
3. initialize iset to mdu

4. call findpset(SS, mdu) // findpset used to find the pset of unit mdu.
5. call findmdu(mduset, iset, len, pset, plen, thresholdarray, record-count)
   // findmdu is a recursive function to find maximal dense units.
6. endfor
7. stop // end of findsubmdu

Presentation of the details regarding the maximal dense and maximal sub dense units to the user

We display the details of the maximal dense units found in the original database DB along with the maximal sub dense units found in them both in descending order of their dimensionality. In order to obtain a concise description for the cluster and sub clusters represented by the maximal dense units and the maximal sub dense units respectively the logic used in step 2 and step 3 of CLIQUE discussed in section 2.2 of chapter 2 can be used.

Clusters obtained from k-dimensional maximal dense units-

u_l^k -

sub clusters details obtained from maximal sub dense units found in u_l^k

(d-k)-dimensional sub dense units –

.

.

1-dimensional sub dense units –

u_2^k -

sub clusters details obtained from maximal sub dense units found in u_2^k

(d-k)-dimensional sub dense units –

.

.

.

1-dimensional sub dense units –

and so on ...

Cluster details obtained from (k-1)-dimensional maximal dense units-

u_1^{k-1} -

sub cluster details obtained from maximal sub dense units found in u_1^{k-1}

(d-(k-1))-dimensional sub dense units –

.

.

.

1-dimensional sub dense units –

u_2^{k-1} -

sub cluster details obtained from maximal sub dense units found in u_2^{k-1}

(d-(k-1))-dimensional sub dense units –

.

.

.

1-dimensional sub dense units –

and so on till the one dimensional dense units.

Clusters obtained from one-dimensional maximal dense units-

u_1^1 -

sub clusters details obtained from maximal sub dense units found in u_1^1

(d-1)-dimensional sub dense units –

.

.

.

1-dimensional sub dense units –

u_2^1 -

sub clusters details obtained from maximal sub dense units found in u_2^1

(d-1)-dimensional sub dense units –

.

.

.

1-dimensional sub dense units –

and so on for each of the one-dimensional dense units.

Example 5.3: Consider a census database having four attributes namely age as a numeric attribute and sex, educational_qualifications and marital_status as categorical attributes.

Each of these attributes have say, the following number of intervals –

for age it will be equal to $\lceil (110 - 1)/10 \rceil = 11$,

for sex it will be 2 having values male and female

for educational_qualifications it will be 6 with values non-SSC, SSC, HSSC, GRADUATE, POST_GRADUATE, DOCTORATE and for marital_status it is equal to 2 having values married and unmarried.

Assume that k = 2, for this database and the threshold values input for age, sex, marital status and educational qualifications are 0.2, 0.6, 0.7 and 0.15 respectively.

The presentation of the dense units and their sub units for a sample of data will be as shown below -

2-dimensional cluster details

< edu_qual, GRADUATE> <Age, 21 –30> -

<sex, female>

<edu_qual, POSTGRADUATE> <Age, 31 – 40> -

<sex, male>

<marital_status, married>

1-dimensional cluster details

<Age, 41-50> -

<sex, male> <edu_qual, DOCTORATE>

<marital_status, married>

5.3.1 Experimental results

We present here the details of the clusters identified using ROSCLIQUE, AOSSCLIQUE, AOMADUCLIQUE, AOMADUMTCLIQUE and AOMLSCLUS using a synthetic dataset. AOMADUCLIQUE and AOMADUMTCLIQUE are the implementations, which use MADUGEN and MADUGENMT algorithm in step one of CLIQUE respectively to find the maximal dense units and use only the maximal dense units to find the subspace clusters, instead of using all the dense units. We compared the results obtained using each of them using a synthetic dataset of size 50,000 with 100 attributes containing three 9-dimensional clusters. The value for the number of intervals was set to 10. For generating synthetic data the method discussed in section 3.4.1 was used. The experiments were run on a 3.00GHz Pentium 4 processor running linux. The results obtained are reported below.

Method Used	Threshold values used	No. of correct clusters found
ROSCLIQUE	0.13	One 9-dimensional (out of 3)
AOSSCLIQUE	0.13	Three 9-dimensional (all 3)
AOMADUCLIQUE	0.13	Two 9-dimensional (out of 3)
AOMADUMTCLIQUE	0.13, 0.16, 0.17, 0.20	Three 9-dimensional (all 3)
AOMLSCLUS	0.13, 0.16, 0.17, 0.20	Three 9-dimensional (all 3)

Besides the three 9-dimensional main clusters, AOMLSCLUS also reported the sub clusters found within the dataset. In case of AOMADUMTCLIQUE and AOMLSCLUS the threshold values obtained using ***findthreshold*** algorithm discussed in section 5.3 were used.

5.4 Summary

In this Chapter, we proposed the subspace clustering problem for mixed data types for finding the subspace clusters in the database by fixing different values of threshold for the different attributes. We also extended the algorithm to find the significant subspace clusters found in the various subspace clusters existing in the database at various levels. This problem can be used to study the patterns found in a typical census data, university/college enrollment data, grading data which contains information about marks, grade awarded, course name, course credits etc.,

Chapter 6

Discussion

We have developed an Attribute Oriented Storage Structure (AOSS), for effective and efficient subspace clustering of very high dimensional huge datasets. In this chapter, we first summarize the major characteristics of the very high dimensional huge datasets. We then explain the efficiency of the AOSS method for use in subspace clustering and give some extensions and applications of subspace clustering using the AOSS method.

6.1 Characteristics of the AOSS method

We have developed the new class of AOSS method for effective and efficient subspace clustering of very high dimensional huge datasets. We, summarize the major characteristics of AOSS based methods here.

- AOSS based methods adopt a divide-and-conquer methodology and partition the data sets consisting of various records from the high dimensional data space into independent AOSS Attribute tables for each attribute belonging to the data set. In general, subspace clustering has to search a very huge very high dimensional data

space. Divide-and-conquer methodology enables the subspace clustering algorithms to focus on reduced subsets of records within each AOSS Attribute table in the first pass and then again a much smaller subset of records belonging to the various dense units by applying the divide-and-conquer strategy on the AOSS Attribute tables. This process automatically focuses on only the relevant set of attributes and the relevant set of records belonging to the various units while finding their selectivity. Hence it eliminates the processing of irrelevant records as well as irrelevant attributes of a particular unit, thereby saving a lot of processing time. It also helps in processing some set of units, which are independent with respect to attributes and units in parallel.

- AOSS based methods also save on the main memory space requirements as those records and attributes not contributing to any dense units are automatically pruned and are not loaded in memory in subsequent processing of the high-dimensional candidate units for determining their selectivity.

- AOSS based-methods MADUGEN and MADUGENMT for finding the dense units use a depth-first search algorithm and eliminate the requirement to find the selectivity of all the 2^{k-1} subsets of a k-dimensional dense unit before finding the selectivity of the k-dimensional unit, a feature which is most common in the level-wise (apriori based) algorithms.

6.2 Extensions and Applications of AOSS based methods

We have shown that the AOSS based methods are effective and efficient in subspace clustering for finding the dense units belonging to the various subspaces of a high dimensional dense unit. The AOSS method however is also applicable to mining other kinds of knowledge and solving some other interesting high dimensional data processing problems. In this section, we discuss some examples.

6.2.1 Mining Multi-dimensional Sequential Patterns from high-dimensional data

Sequential pattern mining, which finds the set of frequent subsequences in sequence databases, is an important data-mining task and has broad applications. Mining of sequential patterns from very high dimensional datasets which is a common requirement in the emerging new applications like protein classification, keyword extraction from text documents, etc is a very interesting task and can greatly benefit from the AOSS structure.

6.2.2 Mining Closed Association Rules from high-dimensional data

In order to reduce the generation of redundant association rules the use of the closed frequent itemsets has been proposed by Pasquier in [26a]. The AOSS can be used for the extraction of the frequent closed itemsets from the high dimensional datasets in an efficient manner

6.2.3 Categorization of high dimensional datasets using subspace clustering

Many real datasets like collection of documents on various subjects, can be viewed as having very high dimensionality and missing dimensional values. Subspace clustering based on the AOSS can be used here, to find efficiently all the clusters some of which may be overlapping.

6.2.4 Mining long Sequences from high-dimensional data

Applications like protein classification, and other bio-informatics applications require effective and efficient mining of long sequences from their high dimensional

data sets. Since, AOSS based methods are efficient for the depth-first search, it can be used to extract the long sequences efficiently from the high dimensional datasets.

Chapter 7

Conclusions

The amount of raw data and information being captured and stored in computer files and databases in almost every field has been growing at a tremendous pace. In recent years, there has been an increase in the number of new database applications dealing with very large high dimensional data sets. These applications place special requirements on clustering algorithms: the ability to find good quality clusters embedded in subspaces of high dimensional data preferably without taking any inputs from the user (which requires the user to have good domain knowledge), scalability, end-user comprehensibility of the results, non-presumption of any canonical data distribution, and insensitivity to the order of input records. In this thesis, we studied the problem of subspace clustering for very high dimensional huge data sets with missing values.

In this chapter, we summarize the thesis, and then present some directions for future work.

7.1 Summary of the Thesis.

Clustering has been a very active area of research in data mining for the past several years. But, most of the clustering algorithms designed work on the full dimensional data space and cannot be used for finding clusters in datasets having a very large number of attributes. The subspace clustering algorithm *CLIQUE* identifies the subspace clusters in the high dimensional data by finding all the sets of connected dense units existing in the various subspaces. However, it requires the user to give the inputs, τ (threshold value) and ξ(number of intervals) in order to find the dense units. Hence the accuracy of the results obtained depends on the values input by the user. It uses the level-wise *apriori* algorithm for finding the dense units. Hence suffers from the same problems as the *apriori* algorithm. In this thesis, we study the problem of subspace clustering for very high dimensional huge data sets with missing values and make the following contributions.

- We propose an Attribute Oriented Storage Structure (AOSS) for storing very high dimensional huge data sets considering the requirements of the subspace clustering algorithms for very high dimensional large datasets containing missing values.

- We have used the sampling technique to reduce the number of database passes required to find the dense units. The *SAMCLIQ* algorithm developed using sampling technique gave us very efficient results when compared with the *CLIQUE* algorithm.

- We have used the maximal dense units to identify the subspace clusters in order to improve the efficiency of the first step which used dense units for this purpose. Again here we used the AOSS method of storage representation and found that it gives very good results for very high dimensional huge datasets with missing value attributes.

- We extended the AOSS method to develop a multi-level subspace clustering algorithm to allow the mining of subspace clusters at different levels from a dataset having attributes with varied threshold requirements.

7.2 Future Research Directions

With the increase in the desire and ease of collecting data, most of the resulting databases in today's information era will be very high dimensional in nature with a lot of missing values also and huge in size. Hence, it will be very interesting and

challenging to re-examine and explore many related problems, extensions and applications of subspace clustering for these databases. Some of them are listed here.

- Visual Subspace Clustering.

- Interactive Subspace Clustering - some effort has been put here but the results were not all that satisfactory, hence not reported.

- Subspace Clustering for streaming data.

- Subspace Clustering for Keyword extraction – Some work in this direction has been carried out, but not included in this thesis.

References

[1] R. Agarwal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In Proceedings of the ACM SIGMOD Conference on Management of Data, Montreal, Canada, 1998.

[2] C. C. Agarwal, C. Procopiuc, J. L. Wolf, P.S. Yu and J. S. Park. A Framework for Finding Projected Clusters in High Dimensional Spaces. In Proceedings of ACM SIGMOD International Conference on Management of Data, 1999.

[3] C. C. Agarwal, and P. Yu. Finding Generalized Projected Clusters in High Dimensional Space. In Proc. ACM SIGMOD Int. Conf. On Management of Data, Dallas, TX.(2000)

[4] R. Agarwal, H. Manilla, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast Discovery of Association Rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, Advances in Knowledge Discovery and Data Mining, chapter 12, pages 307 -328. AAAI/MIT Press , 1996.

[5] A. Aho, J. Hopcroft, and J. Ullman. The Design and Analysis of Computer Algorithms. Addison -Welsley,1974.

[6] S. Berchtold, C.Bohm, D. Keim, and H.-P. Kriegel. A cost model for nearest neighbour search in high-dimensional data space. In *Proceedings of the 16th Symposium on Principles of Database Systems(PODS)*, pages 78-86, 1997.

[7] S. Brin, R. Motwani, J. D. Ullman and S. Tsur. Dynamic Itemset Counting and Implication Rules for Market Basket Data. In Proceedings of the ACM SIGMOD Conference on Management of Data, May 1997.

[8] C. Cheng , A. W. Fu and Y. Zhang. Entropy-based Subspace Clustering for Mining Numerical Data. In Proceedings of ACM SIGMOD Conference, August 1999.

[9] P. Chundi and U. Dayal. An Application of Adaptive Data Mining : Facilitating Web Information Access. 19 97 SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery, Arizona, USA.

[10] P. Cheeseman and J. Stutz. Bayesian classification (AutoClass): Theory and results. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining,* pages 153-180. AAA/MIT Press, 1996.

[11] M. Ester, H. -P. Kriegel, and X. Xu. Knowledge discovery in large spatial databases: Focusing techniques for efficient class identification. *In Proc. 4th Int. Symp. On Large Spatial Databases(SSD'95),* pages 67-82, Portland, Maine, August 1995.

[12] M. Ester, H. -P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Proceedings of the 2nd International Conference on Knowledge Discovery in Databases and Data Mining , Portland, Oregon, August 1996.

[13] R. Feldman, Y. Aumann, A. Amir and H. Manilla. Efficient algorithms for discovering frequent sets in incremental database s. In Proceedings of SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, May 1997.

[14] D. Fisher. Improving inference through conceptual clustering. In Proc. 1987 AAAI Conf., pages 461-465, Seattle, Washington, July 1987.

[15] D. Fisher. Optimization and simplification of hierarchical clusterings. First International Conference on Knowledge Discovery and Data Mining (KDD - 95), Montreal, Quebec, Canada, AAA1 Press, Menlo Park, California. Pp. 118-123.

[16] S. Goil, H. Nagesh, A. Choudhary. MAFIA : Efficient and Scalable Subspace Clustering for Very Large Data Sets. Technical Report No. CPDC-TR-9906-010 , Center for Parallel and Distributed Computing, June 1999.

[17] S. Guha, R. Rastogi, and K. Shim. CURE : An efficient clustering algorithm for large databases. In Proceedings of the ACM SIGMOD Conference on Management of Data, Montreal, Canada, June 1996.

[18] T. Hagerup and C. Rub. A guided tour of chernoff bounds. In Information Processing Letters, pages 305 -308. North -Holland, 1989/90.

[19] Heikki Mannila. Data mining: machine learning, statistics, and databases. URL : http://www.cs.helsinki.fi/~mannila/

[20] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, 1988.

[21] J. Kivinen and H. Mannila. The power of sampling in knowledge discovery . In Proceedings of the Thirteenth ACM SIGACT -SIGMOD-SIGART Symposium on Principles of Database Systems(PODS'94),PAGES 77 -85, Minneapolis, MN, May 1994.

[22] H. Manilla and H. Toivonen. On an algorithm for finding all interesting sentences. In Cybernetics and Systems, Volume II, The Thirteenth European Meeting on Cybernetics and Systems Research, pages 973 - 978, Vienna, Austria, April 1996.

[23] H. Mannila , H. Toivonen and I. Verkamo. Efficient algorithms for discovering association rules. In AAAI Wkshp. Knowledge Discovery in Databases, July 1994.

[24] H. Mannila. Methods and problems in data mining. In Proceedings of International Conference on Database Theory, Delphi, Greece, January 1997, F. Afrati and P. Kolaitis (ed.), Springer-Verlag.

[25] T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In Proceedings of the 20th VLDB Conference , Santiago, Chile, 1994.

[26a] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In Proc. 7th Int. Conf. Databases Theory(ICDT '99), pages 398-416, Jerusalem, Isreal, Jan 1999.

[26] G. Piatetsky-Shapiro and W. J. Frawley. *Knowledge Discovery in Databases.* AAAI/MIT Press, 1991.

[27] V. Pudi and J. Haritsa. Incremental Mining of Association Rules. Technical Report, DSL, Indian Institut e of Science, 1999.

[28] S. Thomas, S. Bodagala, K. Alsabati and S. Ranka. An Efficient algorithm for the incremental updation of association rules in large databases. In Proceedings of 3rd KDD Conference , August 1997.

[29] H. Toivonen. Sampling Large Databases for Association Rules. In Proceedings of the 22nd VLDB Conference, Mumbai,India 1996.

[30] J. S. Vitter An Efficient algorithm for sequential random Sampling. Technical Report 624, INRIA, Feb. 1987.

[31]M. J. Zaki, S. Parthasarathy, W. Li, M. Ogihara Evaluation of Sampling for Data Mining of Association Rules . Technical Report 614, The University of Rochester Computer Science Department , New York.

[32]T. Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH : An efficient data clustering method for very large databases. In Proceedings of the ACM SIGMOD Conference on Management of Data, Montreal, Canada, pages 103-114, June 1996.

[33] Doug Burdick, Manuel Calimlim, Jason Flannick, J. Gehreke and Tomi Yiu. MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases. In Proceedings of the 17th International Conference on Data Engineering, Heidelberg, Germany, April 2001.

[34] M. J. Zaki, K. Gouda. GenMax: An Efficient Algorithm for Mining Maximal Frequent Itemsets. Data Mining and Knowledge Discovery.

[35] Bayardo, R. J. Efficiently mining long patterns from databases. In ACM SIGMOD Conference of Data, 1998, pp. 85-93.

[36] J. D. Pawar, P. R. Rao. A subspace Clustering Algorithm for Finding Clusters in Large Databases Using Sampling. In Proceedings of the International Conference KBCS – 2002, pp. 27-36.