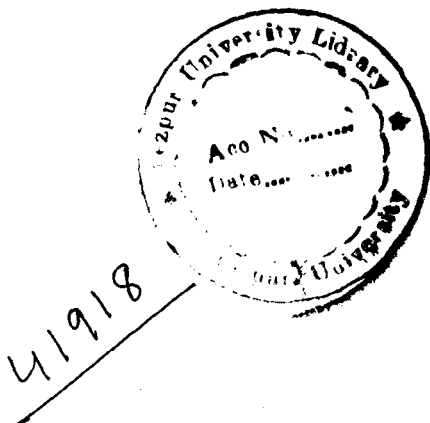


T 108



T 108
26/02/13

REFERENCE BOOK
NOT TO BE ISSUED
TEZPUR UNIVERSITY LIBRARY

A Study on Clustering Techniques for Numeric, Categorical and Mixed-type Data

*A thesis submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy*

Bhogeswar Borah

Registration No. 001 of 2007



School of Engineering
Department of Computer Science and Engineering
Tezpur University
July 2008

Abstract

Rapid advances in data capture, transmission and storage technologies have enabled modern business and science to collect increasingly large volumes of data. Data mining is the technique of analyzing such large datasets in order to reveal embedded patterns (regularities and relationships) that are nontrivial. Clustering is one of the primary data analysis tasks in data mining. Clustering techniques partition a set of objects so that objects with similar characteristics are grouped together and different groups contain objects with dissimilar characteristics. It is either used as a stand alone tool to get insight into the data distribution pattern of a dataset or as a preprocessing step for other data mining algorithms operating on the detected clusters.

The attributes used to describe data objects can be quantitative, qualitative or mixture of both. The types of attributes determine the clustering techniques to be used to analyze the data.

Data mining applications place special requirements on clustering algorithms including : scalability, ability to find clusters embedded in subspaces of high dimensional data, ability to find clusters with widely varying sizes, shapes and densities, ability to deal with mixture of attribute types, and insensitivity to the order of input records.

We have developed separate algorithms for clustering numeric, categorical and mixed-type data satisfying these requirements. Two application specific techniques are also developed. The following are the different algorithms included in the thesis.

1. *An Improved Sampling-Based DBSCAN for Large Spatial Databases.*
2. *A Parallelization of Density-Based Clustering Technique on Distributed Memory Multicomputers.*

3. *DDSC: A Density Differentiated Spatial Clustering Technique* to detect clusters with widely differing densities.
4. *CatSub: Clustering Categorical Data Based on Subspace.*
5. *SMIC: A Subspace Preferred Mixed Type Data Clustering Technique* to find clusters in large high dimensional datasets with mixture of numeric and categorical attributes.
6. *Biclustering Gene Expression Data Using A Node Addition Algorithm.*
7. *A Clustering Based Technique For Network Intrusion Detection.*

Experimental results establish the validity of the algorithms proposed.

Keywords: *Clustering, query sampling, distributed clustering, variable density, categorical data, mixed-type data, incremental clustering, scalability, outliers, gene expression data, biclustering, intrusion detection.*



TEZPUR UNIVERSITY

Certificate

This is to certify that the thesis entitled “*A Study on Clustering Techniques for Numeric, Categorical and Mixed-type Data*” submitted to the Tezpur University in the Department of Computer Science and Engineering under the School of Engineering in partial fulfillment of the requirements for the award of the degree of Doctor of Philosophy in Computer Science is a record of research work carried out by Mr. Bhogeswar Borah under my personal supervision and guidance.

All helps received by him from various sources have been duly acknowledged.

No part of this thesis has been reproduced elsewhere for award of any other degree.

A handwritten signature in black ink, appearing to read 'Dhruva K. Bhattacharyya'.

(Dhruva K. Bhattacharyya)

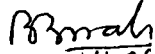
Professor,
Department of Computer Science
and Engineering,
Tezpur University

Head
Department of Computer Science & Engr.
Tezpur University

Date: 14/7/08
Place: Tezpur

Declaration

I, Bhogeswar Borah, hereby declare that the thesis entitled "*A Study on Clustering Techniques for Numeric, Categorical and Mixed-type Data*" submitted to the Department of Computer Science and Engineering under the School of Engineering, Tezpur University, in partial fulfillment of the requirements for the award of the degree of Doctor of Philosophy, is based on bonafide work carried out by me. The results embodied in this thesis have not been submitted in part or in full, to any other university or institute for award of any degree or diploma.


14.06.08
(Bhogeswar Borah)

Acknowledgments

First of all, I wish to express my gratitude to Prof. D. K. Bhattacharyya, my supervisor, for his guidance and continuing support. His constructive advice, constant motivation and encouragement have been responsible for the successful completion of this thesis. I knew little about research at the beginning. He improved my research and writing skills significantly, which will be useful in my further career.

My Sincere thanks go to Prof. M. Dutta, Prof. D. K. Saikia and Prof. D. K. Bhattacharyya for their co-ordination in extending every possible facility as head of the department for completion of this research work. The help I received from my dear colleagues need special mention. In particular I would like to thank Dr. U. Sharma for his help in writing the thesis.

I am thankful to the Tezpur University Authority and the Department of Science and Technology, Govt of India for providing necessary grants for attending conferences, publishing journal papers etc.

I am very grateful to my wife and son for their prayers, continuous encouragement, love and the patience they showed me. I hope they will be proud of my achievements, as proud as I am of them.

Last but not the least, I would like to thank all those technical and non-teaching staff of the department who had helped directly or indirectly towards the completion of this thesis.

Contents

Part I Introduction	1
1 Introduction	2
1.1 Data Mining	2
1.1.1 Data mining tasks	3
1.2 Cluster Analysis	4
1.2.1 Type of data	4
1.2.2 Types of clustering methods	6
1.2.3 Important issues in data clustering	8
1.3 Research Goals and Objectives	10
1.4 Organization of the Thesis	12
Part II Clustering Numeric Data	13
2 Introduction to DBSCAN	14
2.1 Introduction	14
2.2 The Algorithm	16
3 An Improved Sampling-Based DBSCAN for Large Spatial Databases	20
3.1 Introduction	20
3.2 The Query Sampling Procedure	21
3.2.1 Complexity analysis	26
3.3 Experimental Results	26

3.3.1	Experiment 1	26
3.3.2	Experiment 2	27
3.3.3	Experiment 3	28
3.3.4	Experiment 4	30
4	A Parallelization of Density-based Clustering Technique on a Distributed Memory Multicomputer	33
4.1	Introduction	33
4.2	Related Works	34
4.3	The Proposed Algorithm	35
4.3.1	Data placement	35
4.3.2	Local clustering	38
4.3.3	Merging of local cluster models	38
4.3.4	Complexity analysis	41
4.4	Performance Evaluation	41
5	DDSC: A Density Differentiated Spatial Clustering Technique	46
5.1	Introduction	46
5.2	Related Works	47
5.3	Proposed Algorithm	50
5.3.1	Ordered expansion process	51
5.3.2	Homogeneity test	55
5.3.3	Cardinality test	58
5.3.4	Special treatment for the first core object	58
5.4	The Algorithm	59
5.4.1	Complexity analysis	61
5.5	Experimental Results	61
5.5.1	Discussion on Parameters	65
	Part III Clustering Categorical Data	67
6	CatSub: Clustering Categorical Data Based on Subspace	68

6.1	Introduction	68
6.2	Related Works	69
6.3	Problem Formulation	71
6.4	Proposed Algorithm	73
6.4.1	Outlier handling	74
6.4.2	Complexity analysis	78
6.5	Experimental Validation	79
6.5.1	Accuracy calculation of clustering results	79
6.5.2	Data sets	79
6.5.3	Result on soybean dataset	81
6.5.4	Result on zoo dataset	82
6.5.5	Result on congressional voting dataset	83
6.5.6	Result on Wisconsin breast cancer dataset	84
6.5.7	Result on mushroom dataset	85
6.5.8	Scalability test	86
6.5.9	Discussion on parameters	87

Part IV Clustering Mixed-type Data 89

7	SMIC: A Subspace Preferred Mixed Type Data Clustering Technique 90
7.1	Introduction 90
7.2	Related Works 92
7.3	Problem Formulation 93
7.3.1	Entropy 94
7.3.2	Entropy of a categorical attribute 94
7.3.3	Entropy for a numerical attribute 95
7.3.4	Dissimilarity measure of a cluster 96
7.3.5	Subspace based similarity measure 97
7.3.6	Summary measures 97
7.3.7	Cluster structure 98
7.4	Proposed Algorithm 100

7.4.1	Incremental clustering	101
7.4.2	Hierarchical clustering	102
7.4.3	Complexity analysis	106
7.5	Experimental Validation	106
7.5.1	Accuracy calculation of clustering result	106
7.5.2	Data sets	107
7.5.3	Result on credit approval dataset	108
7.5.4	Result on KDD CUP Corrected dataset	109
7.5.5	Result on soybean dataset	109
7.5.6	Result on zoo dataset	110
7.5.7	Result on congressional voting dataset	110
7.5.8	Result on Wisconsin breast cancer dataset	110
7.5.9	Result on mushroom dataset	111
7.5.10	Scalability test	111
7.5.11	Order dependency and parameter sensitivity	111

Part V Application Development 117

8 Biclustering Gene Expression Data Using A Node Addition Algorithm 118

8.1	Introduction	118
8.2	Problem Formulation	120
8.2.1	Incremental computation of score	124
8.3	Proposed Algorithm	126
8.3.1	Initializing the cluster	126
8.3.2	Extending the cluster	126
8.3.3	Row extension	127
8.3.4	First screening	127
8.3.5	Second screening	128
8.3.6	Column extension	129
8.4	Experimental Results	131

9	A Clustering Based Technique For Network Intrusion Detection	138
9.1	Introduction	138
9.2	Related Works	140
9.3	Methodology	142
9.3.1	Clustering	143
9.3.2	Detection	143
9.4	Experimental Results	144
9.4.1	Dataset description	144
9.4.2	Performance measures	145
10	Conclusion	149
10.1	Directions for future works	150

List of Tables

3.1	Indexing and determination of diagonal <i>MBOs</i>	24
3.2	Indexing and determination of non-diagonal <i>MBOs</i>	25
3.3	Run-time of <i>IDBSCAN</i> and <i>DBSCAN</i> for increasing dataset sizes	28
3.4	Run-time of <i>IDBSCAN</i> and <i>DBSCAN</i> for constant <i>MinPts</i> values	29
3.5	Run-time of <i>IDBSCAN</i> and <i>DBSCAN</i> for increasing <i>MinPts</i>	30
6.1	A sample dataset	73
6.2	Datasets used in the experiments.	81
6.3	Clustering result on soybean dataset.	82
6.4	Accuracy on soybean dataset.	82
6.5	The misclassification matrix of clustering result on zoo dataset.	83
6.6	The misclassification matrix on zoo dataset reported by Li et al.	83
6.7	Clustering result on Congressional Voting dataset.	84
6.8	Clustering result of <i>ROCK</i> on Congressional Voting dataset.	84
6.9	Clustering result on breast cancer dataset.	85
6.10	Clustering result on breast cancer dataset reported by SUBCAD.	85
6.11	Clustering result on Mushroom dataset.	86
6.12	Clustering result by <i>ROCK</i> on Mushroom dataset.	86
6.13	Execution time for the datasets.	87
7.1	A sample dataset.	99
7.2	Summary measure	100

7.3	Clustering result on Credit approval dataset.	108
7.4	Accuracy on credit approval dataset.	108
7.5	Clustering result on KDD CUP Corrected dataset.	113
7.6	Clustering result on soybean dataset.	114
7.7	Accuracy on soybean dataset.	114
7.8	The misclassification matrix of result on zoo dataset.	114
7.9	Clustering result on Congressional Voting dataset.	115
7.10	Clustering result on breast cancer dataset.	115
7.11	Accuracy on breast cancer dataset.	115
7.12	Clustering result on Mushroom dataset.	116
7.13	Clustering result by ROCK on Mushroom dataset.	116
7.14	Execution time for the datasets.	116
8.1	Sample biclusters in yeast dataset	132
8.2	Sample biclusters in human dataset	133
8.3	Performance of proposed algorithm on yeast data set	136
8.4	Performance of proposed algorithm on lymphoma data set	136
8.5	Summary of biclusters with score less than 100 from yeast data set	137
9.1	Attacks distribution in <i>Corrected</i> KDD dataset	145
9.2	Performance of first phase.	146
9.3	<i>Detection rate</i> of individual attack classes after the first phase . .	147
9.4	Performance of the algorithm.	147
9.5	<i>Detection rate</i> of individual attack classes after the second phase	147

List of Figures

2.1	ϵ -neighbourhood	15
2.2	Density reachability and density connectivity	16
3.1	Location of <i>MBOs</i> for 2-dimensional space	22
3.2	Dataset with 13 clusters of various shapes and sizes	27
3.3	Run-time of <i>IDBSCAN</i> vs <i>DBSCAN</i> for increasing dataset sizes	28
3.4	Run-time of <i>IDBSCAN</i> vs <i>DBSCAN</i> for constant <i>MinPts</i> values	29
3.5	Run-time of <i>IDBSCAN</i> vs <i>DBSCAN</i> for increasing <i>MinPts</i>	30
3.6	Clustering result of <i>DBSCAN</i> on second dataset	31
3.7	Clustering result of <i>IDBSCAN</i> on second dataset	32
4.1	Overlapped spatial partitioning of a 2-dimensional dataset	36
4.2	Overlap width of 2ϵ	39
4.3	Merging of local clusters to form global clusters	40
4.4	Execution time on different number of processors.	43
4.5	Speedup of parallel <i>DBSCAN</i> on different number of processors	43
4.6	Efficiency vs. number of processors employed	44
4.7	Scaleup of parallel <i>DBSCAN</i> on different sets of data scaled by the number of processors	45
5.1	Clusters with varying densities.	47
5.2	Already processed objects within the neighbourhood of currently processed object <i>p</i>	54

5.3	Density variation pattern produced by two adjacent regions with different densities.	56
5.4	Dataset2	62
5.5	Result on <i>Dataset1</i>	63
5.6	Result on <i>Dataset2</i>	63
5.7	Result on <i>t4.8k.dat</i>	63
5.8	Result on <i>t7.10k.dat</i>	64
5.9	Result on <i>t8.8k.dat</i>	64
5.10	Result on <i>t5.8k.dat</i>	64
5.11	Result on <i>t7.10k.dat</i> with increased <i>MinPts</i>	66
6.1	Scalability of CatSub to the no. of records when clustering KDD CUP Corrected dataset.	87
7.1	Scalability of <i>SMIC</i> to the no. of records.	112
8.1	Coherence patterns in submatrix A_{IJ}	121
8.2	First bicluster in <i>Table 8.1</i>	133
8.3	Second bicluster in <i>Table 8.1</i>	134
8.4	Third bicluster in <i>Table 8.1</i>	134
8.5	First bicluster in <i>Table 8.2</i>	135
8.6	Second bicluster in <i>Table 8.2</i>	135
8.7	Third bicluster in <i>Table 8.2</i>	136
9.1	ROC curve	148

Part I

Introduction

The objective of Part I is to give basic ideas related to clustering in data mining highlighting the problem areas that we want to solve. The goals and objectives of the research work are stated.

Chapter 1

Introduction

1.1 Data Mining

Rapid advances in data capture, transmission and storage technologies have enabled modern business and science to collect increasingly large volumes of data. Retailers are accumulating their daily transactions into large databases. Besides direct use of the databases the enterprises can benefit immensely in the areas of marketing, advertising and sales if interesting previously unknown customer buying patterns can be discovered from the volume of gathered past data. In the domain of scientific computing large amount of remote sensing data, protein data and genome data are collected for inferring some valuable information from them. Data mining [HK06, TSK06, HMS04] is the technique of analyzing such large datasets in order to extract implicit, previously unknown, and potentially useful information that might otherwise remain unknown. It is defined in [HMS04] as follows:

Data mining is the analysis of (often large) observational datasets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner.

The relationships and summaries, for example linear equations, rules, clusters, graphs etc., derived through a data mining exercise are often referred to as models or patterns. A model structure is a global summary of datasets, while pattern structure makes statements only about restricted regions of the space spanned by the variables. The observational data referred to in the definition deals with data that have already been collected for some purpose other than the data mining analysis. This means that the objective of data mining exercise play no role in the data collection strategy. For this reason data mining is often referred to as secondary data analysis.

1.1.1 Data mining tasks

Data mining is categorized into different types of tasks based upon the different types of models or patterns they find. In general, data mining tasks are classified into two categories: predictive and descriptive. Predictive mining tasks perform inference on the current data in order to make predictions. Descriptive mining tasks characterize the general properties of the data in the database. Some of the important data mining tasks [HK06] include:

1. *Mining frequent patterns, Association and Correlations*: Frequent patterns are patterns that occur frequently in data. Mining frequent patterns leads to the discovery of interesting associations and correlations within data.
2. *Classification and prediction*: Classification is the process of finding a model (or function) that describes and distinguishes data classes or concepts, for the purpose of being able to use the model to predict the class of objects whose class label is unknown. The derived model is based on the analysis of training data (i.e. data objects whose class labels are known).
3. *Outlier analysis*: A database may contain data objects whose characteristics are significantly different from the rest of the data. These data objects are

known as outliers. In some applications such as fraud detection, rare events can be more interesting than the regularly occurring ones. The analysis of outlier data is referred to as outlier mining.

4. *Evolution analysis*: Data evolution analysis describes and models trends or regularities for objects whose behaviour changes over time.
5. *Cluster analysis*: Cluster analysis groups data objects based on information found in the data that describes the objects and their relationships. The goal is to partition a set of objects into groups, so that objects with similar characteristics are grouped together and different groups contain objects with dissimilar characteristics. The greater the similarity within a group and the greater the difference between groups, the better or more distinct is the clustering.

Our work is primarily on clustering techniques. So, we concentrate on cluster analysis only.

1.2 Cluster Analysis

Cluster analysis is a primary method of data mining. It is either used as a stand alone tool to get insight into the data distribution pattern of a dataset or as a preprocessing step for other algorithms operating on the detected clusters.

1.2.1 Type of data

Datasets differ in a number of ways. For example, the attributes used to describe data objects can be quantitative or qualitative, some datasets contain objects with explicit relationship to one another such as time series. The type of data determines which tools and techniques can be used to analyze the data.

A dataset can often be viewed as a collection of data objects (also called records, points, patterns, vectors). In turn data objects are described by a number of attributes (also called fields, dimensions, features, variables) that capture the basic characteristics of an object. Most frequently a dataset is a file, in which the objects are records (or rows) in the file and each field (or column) corresponds to an attribute.

Data representation

A dataset contains n objects such as employees, with d attributes such as employee ID, name, address, age, date of joining and so on. The dataset can be thought of as a $n \times d$ data matrix $\{x_{ij}, i = 1, 2, \dots, n \text{ and } j = 1, 2, \dots, d\}$, where each row i represents an object (X_i) and each column j represents an attribute (A_j). The possible values that any object can take for the j -th attribute are defined in the domain D_j of the attribute A_j .

Types of attributes

Based on scale of measurement attributes can be divided into four types: nominal, ordinal, interval and ratio. It is convenient to illustrate the concept with two values of an attribute. Let us consider the values taken by k -th attribute on i -th and j -th objects i.e. x_{ik} and x_{jk} .

1. A *nominal scale* merely distinguishes between categories. That is with respect to x_{ik} and x_{jk} one can only say $x_{ik} = x_{jk}$ or $x_{ik} \neq x_{jk}$.
2. An *ordinal scale* induces an ordering of the values of the attribute. In addition to distinguishing between $x_{ik} = x_{jk}$ and $x_{ik} \neq x_{jk}$, the case of inequality is further refined to distinguish between $x_{ik} > x_{jk}$ and $x_{ik} < x_{jk}$.
3. An *interval scale* assigns a meaningful measure of the difference between two values of the attribute. One may say not only that $x_{ik} > x_{jk}$ but also

that x_{ik} is $x_{ik} - x_{jk}$ units different than x_{jk} .

4. A *ratio scale* is interval scale with a meaningful zero point. If $x_{ik} > x_{jk}$ then one may say that x_{ik} is x_{ik}/x_{jk} times superior to x_{jk} .

Nominal and ordinal attributes are collectively referred to as categorical or qualitative attributes. Qualitative attributes, such as employee ID, lack most of the properties of numbers. Even if they are represented by numbers, i.e. integers, they should be treated more like symbols. The remaining two types of attributes, interval and ratio, are collectively referred to as quantitative or numeric attributes. Quantitative attributes can be discrete(integer) or continuous. Generally data objects are described by attributes of the same type. However, in many real databases objects are described by a mixture of attribute types, which we refer as mixed-type data.

1.2.2 Types of clustering methods

In general, major clustering methods can be classified into the following categories.

- *Partitioning methods.* Given a set of objects and a clustering criterion, partitional clustering obtains a partition of the objects into k clusters such that each cluster contains at least one object and each object belongs to exactly one cluster. A partitioning method creates an initial partitioning. Then an iterative relocation technique attempts to improve the partitioning with respect to an objective function by moving objects from one cluster to another. When swapping does not yield any improvements in the objective function, it completes finding a locally optimal partition. The general criterion of a good partitioning is that objects in a cluster are more similar to each other than they are to objects in other clusters. Partitioning clustering methods work well for finding spherical-shaped clusters. Main difficulties

with these methods include: (1) the number of clusters, k need to be known prior to clustering; (2) the method detects spherical-shaped clusters only; (3) Detected clusters tend to become uniformly sized.

- *Hierarchical methods.* Hierarchical clustering is a set of nested clusters that are organized as a tree which is called dendrogram. A hierarchical method can be classified as being either agglomerative or divisive. An agglomerative hierarchical clustering starts by placing each object in its own cluster and then iteratively merges these clusters into larger clusters, until all objects are in a single cluster or a termination condition holds. The divisive approach uses a reverse process by starting with all the objects in the same cluster. In each successive iteration, a cluster is split up into smaller clusters, until eventually each object is in one cluster, or until a termination condition holds. Detection of arbitrary shaped clusters is possible using some hierarchical clustering algorithms. Hierarchical methods suffer from the fact that once a step (merge or split) is done, it can never be undone. These algorithms are computationally expensive. Some hierarchical methods have chaining effect - a few objects located so as to form a bridge between two clusters causes objects across the clusters to be grouped into a single elongated cluster.
- *Density-based methods.* In density-based concept a cluster is a dense region of objects surrounded by a region of low or no density. Density here is considered as the number of data objects in the particular neighbourhood of a data object. In this approach general idea is to continue growing the given cluster as long as the density in the neighbourhood exceeds some threshold. Density-based clustering algorithms suitably handle arbitrary shaped clusters as well as clusters of different sizes. Moreover, they can effectively separate noise and outliers. Density-based definition of a cluster is often employed when the clusters are irregular or intertwined, and when

noise and outliers are present.

- *Grid-based methods.* Grid-based methods quantize the object space into a finite number of cells that form a grid structure. All of the clustering operations are performed on the grid structure. The main advantage of this approach is its fast processing time, which is typically independent of the number of data objects and dependent only on the number of cells in each dimension in the quantized space. There are a number of obvious concerns about grid-based clustering methods. The grids are square or rectangular and do not necessarily fit the shape of the clusters. This can be handled by increasing the number of grid cells, but at the price of increasing amount of work, if the grid size is halved the number of cells increases by a factor of 2^d , where d is the number of dimensions.
- *Model-based methods.* Model-based methods hypothesize a model for each of the clusters and find the best fit of the data to the given model. A model-based algorithm may locate clusters by constructing a density function that reflects the spatial distribution of the data points. It also leads to a way of automatically determining the number of clusters based on standard statistics.

The choice of clustering algorithm depends both on the type of data and on the particular purpose of the application. Some clustering algorithms integrate the idea of several clustering methods, so that it is sometimes difficult to classify a given algorithm as uniquely belonging to only one clustering method category.

1.2.3 Important issues in data clustering

Clustering is a challenging field of research in which its potential applications pose their own special requirements. The following are typical requirements of clustering in data mining.

- *Discovery of clusters with arbitrary sizes, shapes and densities:* Many algorithms tend to find spherical clusters with similar sizes. They do not work well when clusters have different sizes. Clusters that have widely varying densities are harder to detect. It is important to develop algorithms that can detect clusters with arbitrary sizes, shapes and densities.
- *Ability to deal with different types of attributes:* The ability to analyze single as well as mixture of attribute types is demanded by real life applications.
- *Ability to cluster huge volume of data (scalability):* Algorithms used to cluster huge volume of data should have linear or near-linear time and space complexities. Furthermore, algorithms that assume that all the data will fit in main memory are infeasible for large datasets.
- *Ability to cluster high dimensional data:* Many clustering algorithms are good at having low dimensional data involving less than ten dimensions. It is a challenge to cluster high dimensional data, especially considering that such data can be sparse and highly skewed.
- *Ability to deal with noise and outliers:* Clustering algorithms should be able to handle outliers in order to improve cluster quality.
- *Finding subspace clusters:* Clusters may exist in a subset of attributes. It is not feasible to simply look for clusters in all possible subsets of attributes for datasets having large number of attributes.
- *Order dependence:* When the same dataset is presented in different order, the results produced by some clustering algorithms may become drastically different. While it would seem desirable to avoid such algorithms, sometimes the order dependence is relatively minor or the algorithm may have other desirable characteristics.

- *Parameter selection*: Some user defined parameters are required by many algorithms. It should not be difficult to choose the proper value of the parameters. In order to avoid bias over the result it is desirable that a method requires only limited guidance from the user.
- *Interpretability and usability of the clustering results*: Clustering algorithms should produce easy to understand, usable and interpretable results.

A large number of clustering algorithms have been developed in a variety of domains for different types of applications. None of these algorithms is suitable for all types of data, clusters, and applications. A specific method can perform well on one dataset, but very poorly on another. In fact, it seems that there is always room for a new clustering algorithm that is more efficient or better suited to a particular type of data, cluster, or application.

1.3 Research Goals and Objectives

The research goals focus on the following issues:

1. Develop scalable algorithms for finding clusters with arbitrary sizes, shapes, and densities in spatial(numeric) data.
2. Develop an algorithm to cluster large high dimensional categorical datasets.
3. Develop an algorithm to cluster large high dimensional datasets with mixture of categorical and numeric attributes.
4. Develop algorithms to solve real life problems of
 - (a) Biclustering gene expression data.
 - (b) Network intrusion detection.

Research objectives supporting these goals are:

1. The *DBSCAN* algorithm, that detects clusters with variable sizes and shapes, is to be extended to make it scalable to large spatial datasets.
2. The *DBSCAN* algorithm is also to be extended in a different way so that it detects clusters with variable densities.
3. The categorical clustering algorithm to be developed should be a single-pass incremental one without the need of storing the data objects in main memory so that large datasets can be handled. The algorithm should also be subspace-based in order to cluster high dimensional datasets. Outliers handling capability is needed to make the algorithm more efficient.
4. The mixed categorical and numeric data clustering algorithm to be developed should also be subspace-based incremental algorithm so that it becomes suitable for clustering large high dimensional datasets. Incremental algorithm may produce a large number of clusters. So the algorithm should use a second phase consisting of an agglomerative hierarchical clustering technique to reduce the number of clusters to the desired level.
5. An algorithm is to be developed to find biclusters in gene expression data. The algorithm should find small or big biclusters as desired by the user. It should be able to extend a smaller bicluster by adding more rows and columns if possible.
6. One of the algorithms to be developed for clustering large high dimensional datasets is to be utilized for network intrusion detection.

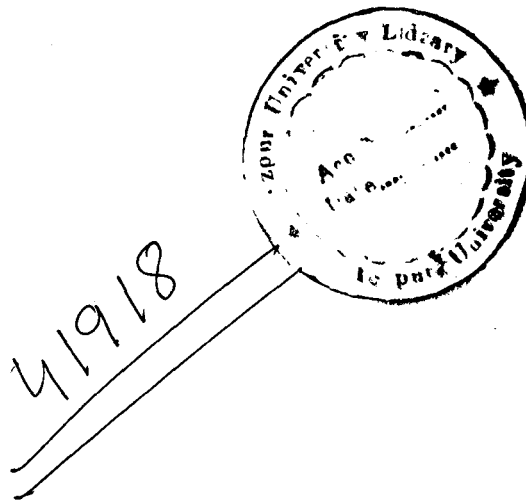
1.4 Organization of the Thesis

The remainder of the thesis is organized into four parts corresponding to the works on numeric, categorical and mixed-type data and application development. *Part II* consists of *Chapters 2–5* related to the extension of *DBSCAN* algorithm, which is introduced in *Chapter 2*. *Chapter 3* presents our sampling-based *DBSCAN* algorithm for large spatial datasets. We parallelize *DBSCAN* in *Chapter 4*. *Chapter 5* includes our *DDSC* algorithm to detect clusters with widely differing densities. The only chapter in *Part III* contains the new algorithm *CatSub*, we have developed to cluster large high dimensional categorical datasets. An algorithm (*SMIC*) for clustering datasets with mixture of categorical and numeric attributes is presented in *Chapter 7* of *Part IV*. *Part V* consists of *Chapter 8 & 9* containing techniques for extracting biclusters in gene expression data and network intrusion detection respectively. Finally the concluding remarks are given in *Chapter 10*.

Part II

Clustering Numeric Data

The *DBSCAN* [EKSX96] is an important clustering algorithm that can detect clusters with arbitrary sizes and shapes beside detecting noise and outliers in spatial (numeric) datasets. But the algorithm becomes very slow if applied to cluster a huge volume of data. Another shortcoming of *DBSCAN* is that it cannot detect clusters with widely varying densities. In this part we present three different algorithms that extend *DBSCAN* so that the drawbacks are removed. Firstly, an introduction to *DBSCAN* is presented in *Chapter 2*. We attempt to make *DBSCAN* scalable by using i) query sampling and ii) parallel processing. The first technique is presented in *Chapter 3* and the second technique is presented in *Chapter 4*. In *Chapter 5* we extended *DBSCAN* to detect clusters with widely varying densities.



Chapter 2

Introduction to DBSCAN

2.1 Introduction

Density-based clustering locates regions of high densities separated from one another by regions of low densities. The *DBSCAN* (Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise) [EKSX96] is a simple and effective density-based algorithm for clustering spatial datasets. Data objects to be clustered consists of numeric attributes so that the objects can be treated as points in a multi-dimensional real space. The algorithm provides a number of important concepts essential for any density-based clustering approach. Density of a particular point in the dataset is estimated by counting the number of points within a specified radius, ϵ of the point including the point itself. The technique is graphically illustrated in *Figure 2.1*. The basic ideas of *DBSCAN* clustering involve a number of definitions, which are produced below. The set of points is represented by X and the distance function between any two points p and q is represented by $dist(p, q)$.

- ϵ -neighbourhood: The ϵ -neighbourhood of a point p , denoted by $N_\epsilon(p)$, is defined as $N_\epsilon(p) = \{q \in X \mid dist(p, q) \leq \epsilon\}$.

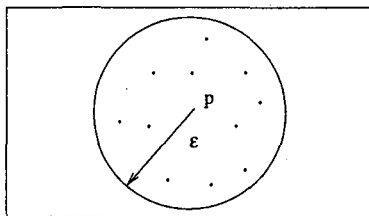


Figure 2.1: ϵ -neighbourhood

- *Core point* : If the ϵ -neighbourhood of a point contains at least $MinPts$ number of points, then the point is called a core point i.e. a point p is core if $|N_\epsilon(p)| \geq MinPts$.
- *Direct density-reachability* : A point p is directly density-reachable from a point q with respect to ϵ and $MinPts$ if $p \in N_\epsilon(q)$ and $|N_\epsilon(q)| \geq MinPts$. Directly density-reachable is symmetric for pairs of core points. In general, it is not symmetric if one core point and one border point are involved.
- *Density reachability*: A point p is density-reachable from a point q with respect to ϵ and $MinPts$ if there is a chain of points p_1, \dots, p_n , $p_1 = q$, $p_n = p$ such that p_{i+1} is directly density-reachable from p_i . Density-reachability is a canonical extension of direct density-reachability. This relation is transitive, symmetric for core points, although not symmetric in general.
- *Density-connectivity* : A point p is density-connected (refer Figure 2.2) to a point q with respect to ϵ and $MinPts$ if there is a point o such that both, p and q are density-reachable from o with respect to ϵ and $MinPts$. Density-connectivity is a symmetric relation.
- *Cluster* : A cluster C with respect to ϵ and $MinPts$ is a non-empty subset of X satisfying the following conditions :

1. $\forall p, q : \text{if } p \in C \text{ and } q \text{ is density-reachable from } p \text{ with respect to } \epsilon \text{ and } MinPts, \text{ then } q \in C. \text{ (Maximality).}$
 2. $\forall p, q \in C : p \text{ is density-connected to } q \text{ with respect to } \epsilon \text{ and } MinPts \text{ (Connectivity).}$
- *Noise* : Let C_1, \dots, C_k be the clusters of the dataset X with respect to parameters ϵ and $MinPts$. Then noise is defined as the set of points in the database X not belonging to any cluster C_i i.e. $\text{noise} = \{p \mid p \in X, \forall i : p \notin C_i\}$.
 - *Border point* : A border point is not a core point, but it falls within the ϵ -neighbourhood of a core point.

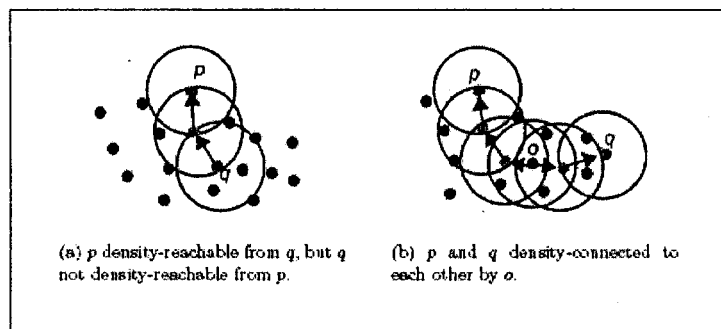


Figure 2.2: Density reachability and density connectivity

2.2 The Algorithm

To find a cluster, *DBSCAN* starts with an arbitrary point p and retrieves all points density-reachable from p with respect to ϵ and $MinPts$. If p is a core point, this procedure yields a cluster with respect to ϵ and $MinPts$. If p is a border point, no points are density-reachable from p and *DBSCAN* visits the next point of the

database. Below, a basic version of *DBSCAN* is reproduced from [EKSX96].

```
DBSCAN(SetOfPoints,  $\epsilon$ , MinPts)
// SetOfPoints is UNCLASSIFIED
ClusterId := nextId(NOISE);
FOR i FROM 1 TO SetOfPoints.size DO
    Point := SetOfPoints.get(i);
    IF Point.CId = UNCLASSIFIED THEN
        IF ExpandCluster(SetOfPoints, Point, ClusterId,  $\epsilon$ , MinPts) THEN
            ClusterId := nextId(ClusterId)
        END IF
    END IF
END FOR
END; // DBSCAN
```

The most important function used by *DBSCAN* is *ExpandCluster* which is presented below:

```
ExpandCluster(SetOfPoints, Point, CIId,  $\epsilon$ , MinPts) : Boolean;
seeds := SetOfPoints.regionQuery(Point,  $\epsilon$ );
IF seeds.size < MinPts THEN // no core point
    SetOfPoint.changeCIId(Point, NOISE);
    RETURN False;
ELSE // all points in seeds are density-reachable from Point
    SetOfPoints.changeCIIds(seeds, CIId);
    seeds.delete(Point);
    WHILE seeds <> Empty DO
        currentP := seeds.first();
        result := SetOfPoints.regionQuery(currentP,  $\epsilon$ );
```

```

IF result.size >= MinPts THEN
  FOR i FROM 1 TO result.size DO
    resultP := result.get(i);
    IF resultP.CId IN {UNCLASSIFIED, NOISE} THEN
      IF resultP.CId = UNCLASSIFIED THEN
        seeds.append(resultP);
      END IF;
      SetOfPoints.changeCId(resultP, CId);
    END IF; // UNCLASSIFIED or NOISE
  END FOR;
END IF; // result.size >= MinPts
seeds.delete(currentP);
END WHILE; // seeds <> Empty
RETURN True;
END IF
END; // ExpandCluster

```

A call of *SetOfPoints.regionQuery(Point, ϵ)* returns the ϵ -neighborhood of *Point* in *SetOfPoints* as a list of points. Region queries can be supported efficiently by spatial access methods such as R*-trees [BKSS90]. The height of an R*-tree is $O(\log n)$ for a dataset of n points in the worst case and a query with a small query region has to traverse only a limited number of paths in the R*-tree. Since the ϵ -neighborhoods are expected to be small compared to the size of the whole data space, the average run time complexity of a single region query is $O(\log n)$. For each of the n points of the database, there is one region query. Thus, the average run time complexity of *DBSCAN* is $O(n \log n)$. The *CId* (clusterId) of points which have been marked to be *NOISE* may be changed later, if they are density-reachable from some other point of the database. This happens for border points of a cluster.

If two clusters $C1$ and $C2$ are very close to each other, it might happen that some point p belongs to both, $C1$ and $C2$. Then p must be a border point in both clusters because otherwise $C1$ would be equal to $C2$ since global parameters are used. In this case, point p will be assigned to the cluster discovered first. Except from these rare situations, the result of *DBSCAN* is independent of the order in which the points of the database are visited.

Since global values are used for ϵ and *MinPts*, *DBSCAN* may merge two clusters into one cluster, if two clusters of different density are close to each other. Let the distance between two sets of points $S1$ and $S2$ be defined as $dist(S1, S2) = \min\{dist(p, q) \mid p \in S1, q \in S2\}$. Then, two sets of points having at least the density of the thinnest cluster will be separated from each other only if the distance between the two sets is larger than ϵ . It means that *DBSCAN* can not detect clusters with variable densities.

Subsequent three chapters contain our work on extending *DBSCAN* so that it can cluster very large datasets consuming lesser execution time and detect clusters with variable densities.

Chapter 3

An Improved Sampling-Based DBSCAN for Large Spatial Databases

3.1 Introduction

DBSCAN computes ϵ -neighbourhood for each of the n objects in the dataset. The complexity of a neighbourhood query is $O(n)$ without using any spatial index structure and using a spatial index structure such as a R*-tree [BKSS90] it is $O(\log n)$. Accordingly the run-time complexity of *DBSCAN* becomes $O(n^2)$ or $O(n \log n)$. For very large databases the neighbourhood query becomes time consuming even if a spatial index is used. Execution time of the algorithm can be reduced using two approaches : (i) by reducing the query time; (ii) by reducing the number of queries. We proposed to speed up *DBSCAN* using the second approach. Zhou et al. [ZZH00] used a query sampling method to reduce the number of queries performed but the algorithm is not detailed. We provide an

improved *DBSCAN* named *IDBSCAN* algorithm by incorporating an efficient sampling method that greatly reduces the number of queries without introduction of much error.

3.2 The Query Sampling Procedure

Starting with the first object in the dataset *DBSCAN* serially examines each object performing neighbourhood query whenever a previously unlabeled object is found. If the object happens to be a core one, all the unlabeled objects in its neighbourhood are labeled with a new cluster-id and copied to the seeds-list. Then the expand-cluster procedure goes on deleting one object at a time from the seeds-list, evaluates neighbourhood query for the deleted object and appends to the seeds-list all the unlabeled neighbours found after marking them with present cluster-id. When the seeds-list becomes empty a cluster is completed and search for next cluster begins.

Instead of copying to the seeds-list all the unlabeled neighbours of a core object selectively copying a few of them may suffice. Because neighbourhood of a core object present in the seeds-list may be covered by neighbourhoods of already processed objects leaving no unlabeled objects to be copied to the seeds-list. Such queries can be avoided without losing accuracy in the clustering result. A procedure to select only the necessary seeds leaving out avoidable ones is needed. Certainly, some of the outer neighbours are good candidates to be selected as seeds while inner neighbours may be left out. We provide an efficient procedure for selecting only the necessary seeds so that a large number of queries are avoided. The procedure may occasionally eliminate a few necessary seeds, resulting in breaking down of a bigger cluster into smaller ones at loosely connected sparse regions. Otherwise the clustering results should be the same as produced by original *DBSCAN*.

The shape of the neighbourhood will be a hyper sphere with radius ϵ as the objects considered are multi-dimensional. Consider a query object lying at the origin of a d -dimensional space. The coordinate axes intersect the hyper sphere at $2d$ points, which are called marked boundary objects (*MBOs*). The axes also divides the hyper sphere into 2^d quadrants. Consider a hyper cube with side length 2ϵ completely enclosing the hyper sphere with radius ϵ . The corner points of the hyper cube are also considered as *MBOs*. Thus, there are $2d + 2^d$ *MBOs*. For example, consider the two-dimensional object q located at coordinate $(0,0)$ as shown in *Figure 3.1*. We have eight distinct *MBOs* : $A(-\epsilon, -\epsilon)$, $B(0, -\epsilon)$, $C(\epsilon, -\epsilon)$, $D(\epsilon, 0)$, $E(\epsilon, \epsilon)$, $F(0, \epsilon)$, $G(-\epsilon, \epsilon)$, $H(-\epsilon, 0)$. The neighbourhood region of object p is divided into four quadrants. In each quadrant we can identify three *MBOs*. In lower left quadrant - H , A , and B , in lower right quadrant - B , C , and D etc.

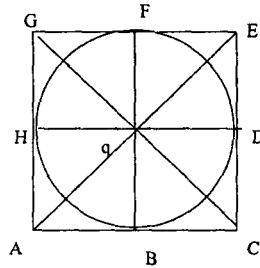


Figure 3.1: Location of *MBOs* for 2-dimensional space

Given the neighbours of a query object our sampling procedure will select at most $2d + 2^d$ objects as seeds. Corresponding to each *MBO* the unlabeled object closest to it is selected as seed provided that the object is not closer to any other *MBO*. Thus some *MBO* may not contribute any seed. Furthermore within the neighbourhood of the current query object there will be many already labeled objects that can not be new seeds. Therefore we adopt an efficient procedure to select the seeds requiring a single pass over the previously unlabeled neighbours only. Let, the seed attached to each *MBO* be initialized to null and the distance of the seed to the *MBO* be set to a high value. Some of the null values will be

replaced by selected seeds using the following procedure :

For each unlabeled object in the neighbourhood of the query object do :

1. Locate the quadrant in which the object lies.
2. Compute the distances of the object to each of the $(d + 1)$ *MBOs* in that quadrant.
3. Find the shortest distance and the corresponding *MBO*.
4. Replace the seed attached to the *MBO* by the object if the object is closer to the *MBO* than the previous seed.

How to find the quadrant in which an object lies and also the corresponding *MBOs*? For this the *MBOs* need to be indexed so that they can be accessed directly using their index numbers. We identify two types of *MBOs* for the sake of indexing:

- *Diagonal MBOs* which are the corner points of the hyper cube as stated earlier.
- *Non-diagonal MBOs* which are the points of intersection of the hyper sphere with the axes.

Index for a diagonal *MBO* becomes the index for the corresponding quadrant also. Let, the q -th object in the dataset, $(x_{q1}, x_{q2}, \dots, x_{qd})$ be a query object which has the k -th object, $(x_{k1}, x_{k2}, \dots, x_{kd})$ as one of the neighbours. Let, $W = \{w_i, i = 1, 2, \dots, d\}$ represents weights such that:

$$w_i = \begin{cases} +1 & \text{if } x_{qi} - x_{ki} > 0 \\ -1 & \text{if } x_{qi} - x_{ki} < 0 \end{cases} \quad (3.1)$$

When the weight vector multiplied by the neighbourhood radius ϵ is added to the query object we get the diagonal *MBO* of the quadrant in which the neighbour is located. Treat w_i as zero if it is negative, consider the modified weight sequence as a binary number and convert the binary number to decimal. It will give the index for this diagonal *MBO* as well as the index for the quadrant. The indices of the diagonal *MBOs* will be in the range $[0, 2^d - 1]$. For three-dimensional objects the diagonal *MBOs* along with their indices are shown in *Table 3.1*.

Table 3.1: Indexing and determination of diagonal *MBOs*

w_1	w_2	w_3	Binary(124)	Index	<i>MBO</i>
-1	-1	-1	000	0	$(x_{q1} - \epsilon, x_{q2} - \epsilon, x_{q3} - \epsilon)$
+1	-1	-1	100	1	$(x_{q1} + \epsilon, x_{q2} - \epsilon, x_{q3} - \epsilon)$
-1	+1	-1	010	2	$(x_{q1} - \epsilon, x_{q2} + \epsilon, x_{q3} - \epsilon)$
+1	+1	-1	110	3	$(x_{q1} + \epsilon, x_{q2} + \epsilon, x_{q3} - \epsilon)$
-1	-1	+1	001	4	$(x_{q1} - \epsilon, x_{q2} - \epsilon, x_{q3} + \epsilon)$
+1	-1	+1	101	5	$(x_{q1} + \epsilon, x_{q2} - \epsilon, x_{q3} + \epsilon)$
-1	+1	+1	011	6	$(x_{q1} - \epsilon, x_{q2} + \epsilon, x_{q3} + \epsilon)$
+1	+1	+1	111	7	$(x_{q1} + \epsilon, x_{q2} + \epsilon, x_{q3} + \epsilon)$

There are $2d$ non-diagonal *MBOs*. Each axis has two *MBOs* - one to the positive side and the other to the negative side of the axis. The indices for these *MBOs* are also found based on the weights vector computed in *Equation 3.1*. At a time consider the weight (+1 or -1) on a single dimension only and treat weights of remaining dimensions to be zeros. Convert the number so obtained to decimal by taking place value of different dimensions to be 1, 2, 3, \dots , d starting from the lowest dimension. These indices will be in the range $[-d, +d]$. Add a bias of $2^d + d$ so that the indices become positive and come after the indices for diagonal *MBOs*. All the d number of *MBOs* in a quadrant can be located in this manner

as shown in *Table 3.2* for 3-dimensional data.

Table 3.2: Indexing and determination of non-diagonal *MBOs*

w_1	w_2	w_3	Binary(123)	Decimal	Index	<i>MBO</i>
-1	-1	-1	-100	-1	10	$(x_{q1} - \epsilon, 0, 0)$
			0 - 10	-2	9	$(0, x_{q2} - \epsilon, 0)$
			00 - 1	-3	8	$(0, 0, x_{q3} - \epsilon)$
+1	-1	-1	+100	1	12	$(x_{q1} + \epsilon, 0, 0)$
			0 - 10	-2	9	$(0, x_{q2} - \epsilon, 0)$
			00 - 1	-3	8	$(0, 0, x_{q3} - \epsilon)$
-1	+1	-1	-100	-1	10	$(x_{q1} - \epsilon, 0, 0)$
			0 + 10	2	13	$(0, x_{q2} + \epsilon, 0)$
			00 - 1	-3	8	$(0, 0, x_{q3} - \epsilon)$
+1	+1	-1	+100	1	12	$(x_{q1} + \epsilon, 0, 0)$
			0 + 10	2	13	$(0, x_{q2} + \epsilon, 0)$
			00 - 1	-3	8	$(0, 0, X_{q3} - \epsilon)$
-1	-1	+1	-100	-1	10	$(x_{q1} - \epsilon, 0, 0)$
			0 - 10	-2	9	$(0, x_{q2} - \epsilon, 0)$
			00 + 1	3	14	$(0, 0, x_{q3} + \epsilon)$
+1	-1	+1	+100	1	12	$(x_{q1} + \epsilon, 0, 0)$
			0 - 10	-2	9	$(0, x_{q2} - \epsilon, 0)$
			00 + 1	3	14	$(0, 0, x_{q3} + \epsilon)$
-1	+1	+1	-100	-1	10	$(x_{q1} - \epsilon, 0, 0)$
			0 + 10	2	13	$(0, x_{q2} + \epsilon, 0)$
			00 + 1	3	14	$(0, 0, x_{q3} + \epsilon)$
+1	+1	+1	+100	1	12	$(x_{q1} + \epsilon, 0, 0)$
			0 + 10	2	13	$(0, x_{q2} + \epsilon, 0)$
			00 + 1	3	14	$(0, 0, x_{q3} + \epsilon)$

3.2.1 Complexity analysis

IDBSCAN uses one more function named *findseeds()* than *DBSCAN*. Use of this function does not increase the overall complexity of *DBSCAN* which is $O(n \log n)$. The extra function used has worst time complexity $O(sd)$, where s is the neighbourhood size and d is the dimensionality of the dataset which are expected to be small compared to the size n of the dataset.

3.3 Experimental Results

We have evaluated the performance of the *IDBSCAN* algorithm in comparison to that of *DBSCAN*. For doing this two 2-dimensional synthetic datasets were created containing circular, rectangular, triangular and S-shaped clusters having different sizes. Values of each attribute fall in the range $[0, 1000]$. The first dataset contains 13 clusters shown in *Figure 3.2* by black dots separated by white spaces. Retaining the cluster shapes to be the same several variants of the dataset were created with increased number of objects (without duplicates). There are some random variations of densities at different regions of the datasets. The second dataset shown in *Figure 3.6* contains 4 clusters. The proposed *IDBSCAN* and *DBSCAN* were implemented in C++ in a 1.1 GHz, HCL Infinity-2000 machine with 128 MB RAM. R*-tree indexing was also used with both of the algorithms.

In *Experiments 1–3* *IDBSCAN* as well as *DBSCAN* recovered the same set of 13 clusters present in the first dataset.

3.3.1 Experiment 1

The execution performances of *IDBSCAN* and *DBSCAN* were compared for datasets of increasing sizes. Value of ϵ parameter were kept constant for all datasets while using progressively higher values for *MinPts* as density of the

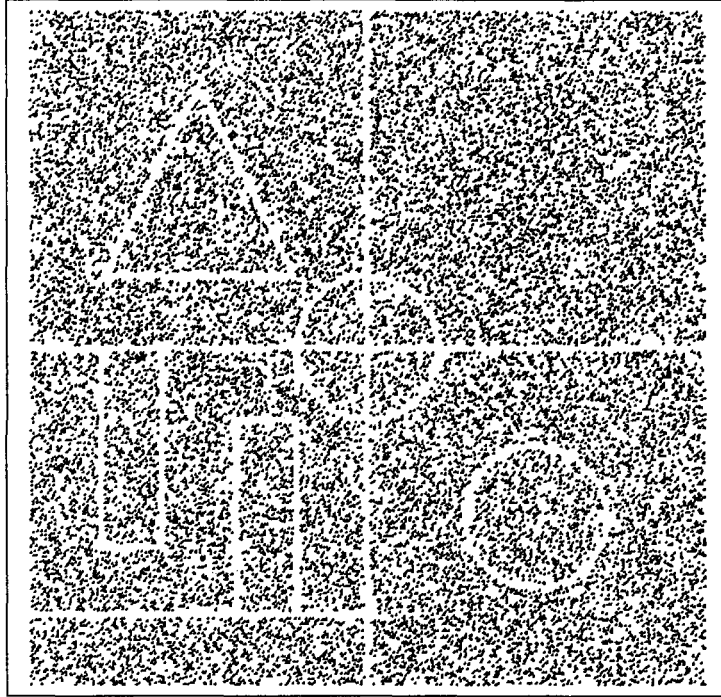


Figure 3.2: Dataset with 13 clusters of various shapes and sizes

datasets increases with increase in size. *Table 3.3* presents the results. It is clear from the graphical presentation (*Figure 3.3*) of the results that *IDBSCAN* is faster than *DBSCAN*. As the dataset size increases the speed difference between *DBSCAN* and *IDBSCAN* also increases.

3.3.2 Experiment 2

In this experiment, with increasing dataset sizes, ϵ values were decreased so that the *MinPts* values remained constant. *Table 3.4* shows the results for two different *MinPts* values. Besides showing speed difference between *DBSCAN* and *IDBSCAN* the graph presented in *Figure 3.4* also shows that *IDBSCAN*

Table 3.3: Run-time of *IDBSCAN* and *DBSCAN* for increasing dataset sizes

Data size	ϵ	$MinPts$	Time(sec.) <i>IDBSCAN</i>	Time(sec.) <i>DBSCAN</i>
100000	8	7	32	62
200000	8	20	63	284
400000	8	40	125	1160
800000	8	75	423	5834

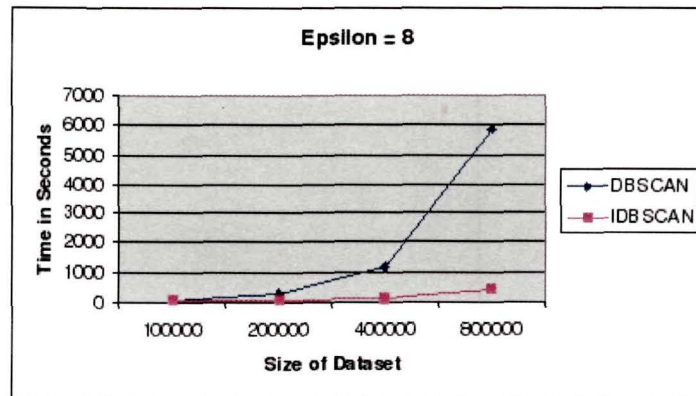


Figure 3.3: Run-time of *IDBSCAN* vs *DBSCAN* for increasing dataset sizes

is less affected by variation in $MinPts$. Execution performance of *DBSCAN* deteriorates for higher values of $MinPts$.

3.3.3 Experiment 3

In this experiment the run-time behaviour of both the algorithms have been studied by varying $MinPts$ in the same dataset. The results are presented in *Table 3.5* and *Figure 3.5*. As $MinPts$ increases, *DBSCAN* becomes slower, whereas the

Table 3.4: Run-time of *IDBSCAN* and *DBSCAN* for constant *MinPts* values

Data size	ϵ	<i>MinPts</i>	Time(sec.) <i>IDBSCAN</i>	Time(sec.) <i>DBSCAN</i>
100000	7	3	33	56
200000	5	3	78	124
400000	5	3	158	624
800000	4	3	498	1410
100000	10	12	27	73
200000	8	12	65	211
400000	6	12	140	749
800000	5	12	455	2366

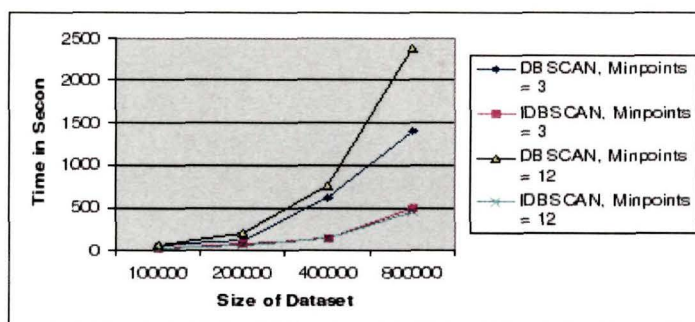


Figure 3.4: Run-time of *IDBSCAN* vs *DBSCAN* for constant *MinPts* values

performance of the *IDBSCAN* improves. Even for smaller values of *MinPts*, a distinct time gap between *IDBSCAN* and *DBSCAN* can be observed.

In the above experiments results produced by *IDBSCAN* and *DBSCAN* are the same. But sometimes *IDBSCAN* may break down a bigger cluster at

Table 3.5: Run-time of *IDBSCAN* and *DBSCAN* for increasing *MinPts*

Data size	ϵ	<i>MinPts</i>	Time(sec.) <i>IDBSCAN</i>	Time(sec.) <i>DBSCAN</i>
400000	5	3	158	624
400000	6	20	141	878
400000	8	40	125	1160
400000	10	60	119	1454

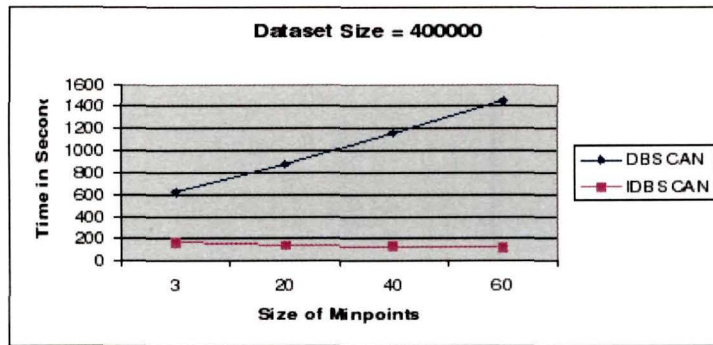


Figure 3.5: Run-time of *IDBSCAN* vs *DBSCAN* for increasing *MinPts*

loosely connected sparse regions as shown in the following experiment.

3.3.4 Experiment 4

Results produced by *DBSCAN* and *IDBSCAN* on the second dataset are shown in *Figures 3.6* and *3.7* respectively. *DBSCAN* produced 4 clusters while *IDBSCAN* produced 5 clusters as it had broken down the S-shaped cluster into two. It can be noticed that the region where the breaking occurs is relatively

sparse. Such results may be desired in some applications.

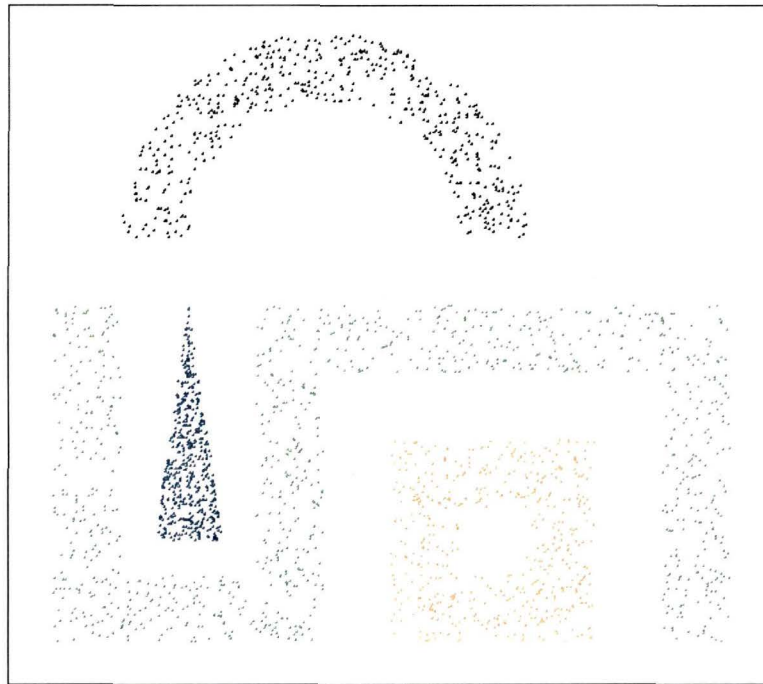


Figure 3.6: Clustering result of *DBSCAN* on second dataset

Based on the results of the experiments, it can be concluded that the proposed *IDBSCAN* speeds up *DBSCAN* by a constant factor and is capable of handling larger volume of data. Occasionally, a *DBSCAN* cluster may be broken into smaller clusters by *IDBSCAN*.

In the next chapter we speed up *DBSCAN* by a large factor using parallel processing without any compromise on the clustering results produced.

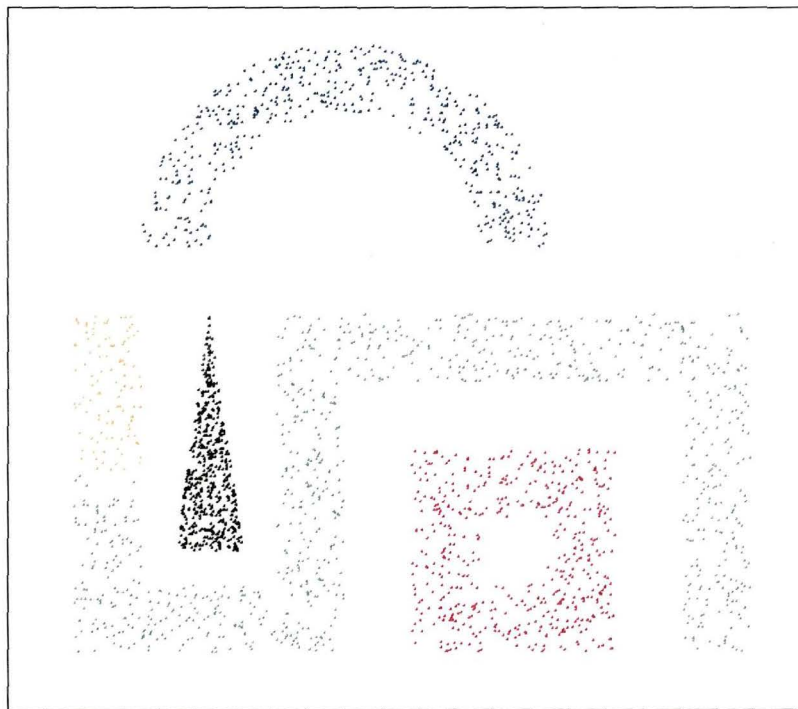


Figure 3.7: Clustering result of *IDBSCAN* on second dataset

Chapter 4

A Parallelization of Density-based Clustering Technique on a Distributed Memory Multicomputer

4.1 Introduction

Massive datasets measuring in gigabytes and even terabytes containing millions of data objects are quite common in business and scientific world today. When used to cluster large datasets, clustering algorithms put high demand on space and time requiring high performance machines to get results in a reasonable time. Very large datasets cannot be processed in-core, that is, in the main memory of a single processor machine. Disk-based algorithms are likely to be considerably slower. In such a situation parallel clustering can be employed to exploit main memories of several processors. Even for the cases when datasets can be processed in-core, the fastest available serial computer may fail to deliver results in a reasonable time. Parallel and distributed computing [Zak00] is expected to relieve current

clustering methods from the sequential bottleneck, providing the ability to scale to massive datasets and improving the response time. Such algorithms divide the data into partitions, which are processed in parallel. The results from the partitions are then merged.

In this chapter a parallel implementation of *DBSCAN* algorithm is presented on a distributed memory multicomputer [WA03] i.e. a low cost shared-nothing parallel environment where each of the processors is a personal computer having private memory and disk. The processors are connected by a communication network. Data parallelization is used with static load balancing. We provide a good data decomposition among the processors and minimize communication between them. Our parallelization is analytically and empirically validated.

4.2 Related Works

There are several efforts directed towards scaling up clustering algorithms for huge datasets commonly encountered in data mining. Parallelization of *k-means* algorithm [SB99, DM00, ZSP03] received a lot of attention in the past. In [Ols95, JK00] parallel agglomerative hierarchical clustering algorithms were reported. A parallel implementation of *AutoClass* is presented in [FLPT00]. A parallel *DBSCAN* algorithm (*PDBSCAN*) is presented in [XJK99] using the shared-nothing architecture. Here, the authors used a distributed version of the R*-tree (dR*-tree), in which the data is spread among multiple computers and the indices on the data are replicated on every computer. The main program of *PDBSCAN*, i.e. the master, starts a clustering slave on each available computer in the network and distributes the whole data onto the slaves. Every slave clusters only its local data and there is some interference between computers while performing local clustering. Local cluster models are then merged together to obtain global clustering.

The method proposed in this paper is more efficient than the earlier method proposed in [XJK99]. In the proposed approach only the master communicates with the slaves for data transferring and collecting local clustering results back. No interference between computers in the parallel environment is needed reducing the communication overhead.

4.3 The Proposed Algorithm

The *DBSCAN* algorithm is adapted for parallel clustering using a message-passing multicomputer. The parallel environment is created by connecting a number of personal computers through a network. Each computer consists of a processor, local memory and disk. Processors can send messages to other processors through the network. The volume of data to be clustered is retained in the secondary memory of a particular computer, which we call as the client. The large dataset may not be accommodated into the main memory of the client. So the dataset is divided into P roughly equal parts and transmitted to P different computers, which we call as servers or processors. The servers process the received data concurrently to produce local clusters, which are then sent back to the client for merging in order to obtain global clusters for the original dataset. There is no communication amongst the servers. Communication is needed only for data transfer in blocks between the client and the servers. The main tasks to be performed are : data placement, local clustering, and merging local cluster models to get global clusters for the whole dataset. Methods for performing each of the tasks are presented below.

4.3.1 Data placement

Data division parallel processing with static load balancing is used here. The dataset is spatially divided into nearly equal partitions with some overlap between

two adjacent partitions and no subsequent movements of data between partitions take place. The input dataset X consists of n d -dimensional objects with i -th object being represented as $X_i = \{x_{i1}, x_{i2}, \dots, x_{id}\}$. Values of each attribute $A_j, j \in \{1, 2, \dots, d\}$ fall in the given range $[\min A_j, \max A_j]$. The dataset X is spatially divided into P partitions S_1, S_2, \dots, S_P based on the values of a particular dimension (say A_k). There is some overlap between adjacent partitions such that $X = S_1 \cup S_2 \cup \dots \cup S_P$ with $S_i \cap S_{i+1} \neq \emptyset$ for $i = 1, 2, \dots, P - 1$ and $S_i \cap S_j = \emptyset$ for $|i - j| \geq 2, i, j \in \{1, 2, \dots, P\}$. The partially overlapped partitions are shown in *Figure 4.1* for 2-dimensional case. An overlap of width 2ϵ occurs between two adjacent partitions. Overlapped regions are much smaller than the partitions. The objects falling in an overlapped region are locally clustered in both the adjacent partitions. Thus they provide the necessary information for merging together the local clustering results of the two partitions. The overlap width should be at least 2ϵ , because the neighbourhood radius is ϵ and the spatial width of the smallest possible cluster will be 2ϵ . To create P partitions of the

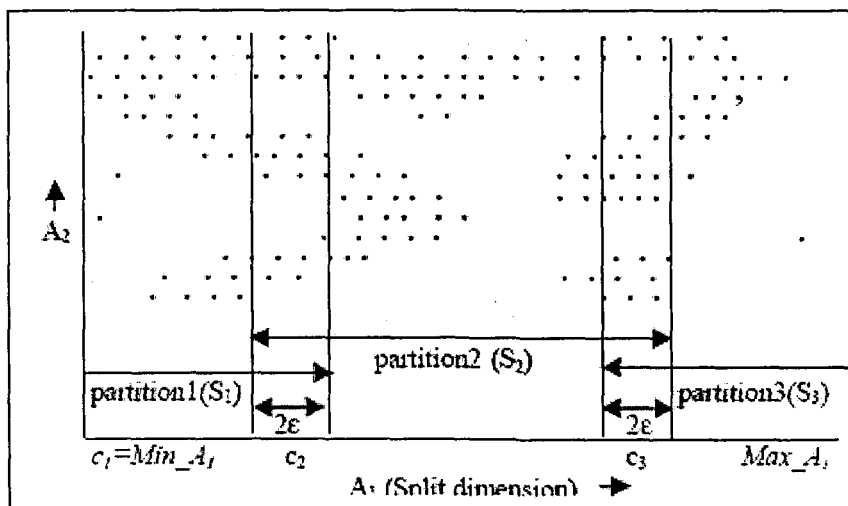


Figure 4.1: Overlapped spatial partitioning of a 2-dimensional dataset

dataset in this manner based on a particular dimension A_k , we need to select $P + 1$ constants in the value range $[MinA_k, MaxA_k]$ for convenience of marking the boundary of the partitions. Let $c_i, i = 1, 2, \dots, P + 1$ represent constants such that $c_1 = MinA_k, c_{P+1} = MaxA_k$, and $c_i < c_{i+1}$. Now, overlapped partitions are created as :

$$S_1 = \{X_j \mid X_j \in X, c_1 \leq x_{jk} \leq c_2 + \epsilon\}$$

$$S_i = \{X_j \mid c_i - \epsilon \leq x_{jk} \leq c_{i+1} + \epsilon\}, i = 2, 3, \dots, P - 1$$

$$S_P = \{X_j \mid X_j \in X, c_P - \epsilon \leq x_{jk} \leq c_{P+1}\}$$

Load balancing

Partition S_i is sent to server $M_i, i = 1, 2, \dots, P$ for concurrent clustering. Since no data movement takes place after the partitions are created, care should be taken so that each server receives nearly equal number of data objects for processing. This will ensure that all the servers finish clustering job at the same time provided the servers have same processing speed. If the processors have different processing speed then the input data should be distributed to the servers proportionate to their processing speed. We assume equal processing speed for the servers so that they receive nearly equal amount of data. To ensure this the value range of the selected dimension A_k i.e. $[minA_k, maxA_k]$ is divided into m intervals each having a width of ϵ by using the limits $a_i, i = 1, 2, \dots, m + 1$. Thus, $m = \lceil \frac{(MaxA_k - MinA_k)}{\epsilon} \rceil$

$$a_1 = MinA_k;$$

$$a_i = a_{i-1} + \epsilon, i = 2, 3, \dots, m + 1;$$

Let, frequency of data objects with k -th dimension value falling in the i -th interval be represented by f_i so that :

$$f_i = |\{j \mid (X_j \in X), a_i \leq x_{jk} < a_{i+1}\}|, i = 1, 2, \dots, m;$$

Now, the constants c_i described in the previous section are computed as $c_i = a_t$ for some $t \in \{1, 2, \dots, m+1\}$ such that :

$$\sum_1^t f_j \leq i.n' \leq \sum_1^{t+1} f_j \text{ for } i = 1, 2, \dots, P$$

where, $n' = \lceil \frac{n}{P} \rceil$ is the average number of objects in each partition.

This will ensure that each partition gets number of objects nearly equal to n' .

4.3.2 Local clustering

When the client finishes sending data, the process of clustering will concurrently run in all the servers. First R*-tree is created for the received data and then DBSCAN algorithm is run. When clustering job is finished the servers will remain ready for sending the results of clustering back to the client.

4.3.3 Merging of local cluster models

The partitioning process may convert an inherently core (i.e. core in the original dataset) object to non-core if the object falls near the boundary of a partition. In this context the following lemma is important.

Lemma 1 *If an overlap of width at least 2ϵ between two adjacent partitions S_i and S_{i+1} is used then an inherently core object $p \in S_i \cap S_{i+1}$ will remain as core in at least one of the two partitions.*

Proof: Consider a core object p in the original dataset. After partitioning $p \in S_i \cap S_{i+1}$ such that p is placed at a distance d_1 from the right boundary of S_i and at a distance d_2 from left boundary of S_{i+1} so that $d_1 + d_2 = 2\epsilon$. It is shown in Figure 4.2 for 2-dimensional case. $N_\epsilon(p)$ represent the set of objects in ϵ -neighbourhood of object p . If $d_1 < \epsilon$ then p may not remain core in S_i , since $N_\epsilon(p)$ will not be a subset of S_i . But in that case $d_2 > \epsilon$ because $d_1 + d_2 = 2\epsilon$ and $d_1 < \epsilon$. So, $N_\epsilon(p)$ will be a subset of S_{i+1} and consequently p will remain a core object. Similarly p may be found to be core in S_i but non-core in S_{i+1} . \square

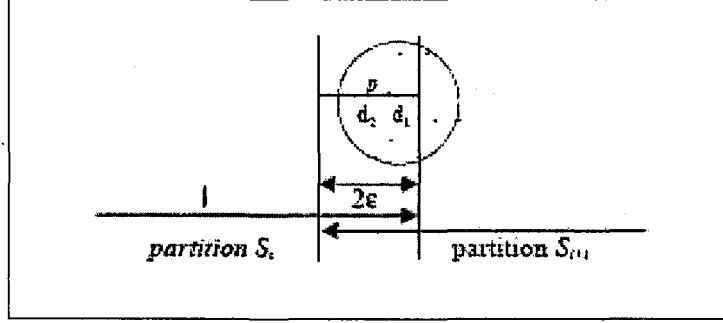


Figure 4.2: Overlap width of 2ϵ

Each server M_i finds clusters in partition S_i . With respect to the dataset X , we can think of two types of clusters. A cluster may be totally contained within a partition S_i or a cluster may span over more than one partitions. In the later case we need to merge the clusters obtained in two adjacent partitions. The following lemma states the conditions for merger of two clusters found in the two adjacent partitions.

Lemma 2 *Let C_1 and C_2 are clusters found in two adjacent partitions S_i and S_{i+1} respectively. If $p \in C_1 \cap C_2$ and p is a core object in at least one of the partitions then C_1 and C_2 need to be merged. If p is not a core object in either of the partitions then p should be included either in C_1 or in C_2 but not in both.*

Proof: Suppose $x \in C_1$ in S_i and $y \in C_2$ in S_{i+1} . If p is found to be a core object either in C_i or in C_2 or in both, then x and y are density reachable from p since p is a core object and $p \in C_1 \cap C_2$. So x is density connected to y and x, y should belong to the same cluster i.e. clusters C_1 and C_2 need to be merged. If p is not core in either of the partitions then p is a border object which can be included in any one of C_1 or C_2 . \square

The merging of clusters is done as follows. Each server M_i , $i = 1, 2, \dots, P$ while clustering the partition S_i prepares a list L_i recording whether each object $p \in S_i \cap S_{i+1}$ is a core, non-core or noise object. The job of merging is done by the

client. It first collects the local clustering result from partition S_1 along with list L_1 from M_1 and forms the initial merged result G . It then collects clustering result of partition S_i and list L_i , $i = 2, 3, \dots, P-1$. The lists L_{i-1} and L_i are consulted by the client to determine the status of each object $p \in S_{i-1} \cap S_i$ and merges S_i with G based on *lemma 2*. The following lemma proves that the resulting clusters are the same as the clusters obtained by applying *DBSCAN* on dataset D .

Lemma 3 *If clustering results of all the partitions are merged based on lemma 2 then the merged result is the same as the clustering result obtained by DBSCAN.*

Proof. *DBSCAN* clusters are maximal set of density connected objects. The condition of density connectivity is established by *lemma 2* and the condition of maximality is satisfied by merging clustering results of all the partitions. \square

The ideas of connectivity and maximality are also depicted in *Figure 4.3*. A directed edge indicates the two clusters that are to be merged across partitions. A maximal connected sub-graph of directed edges indicate a cluster in the given dataset X .

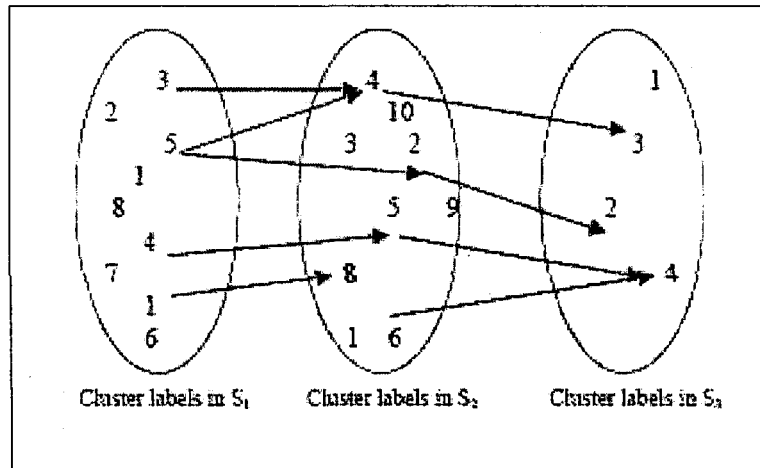


Figure 4.3: Merging of local clusters to form global clusters

4.3.4 Complexity analysis

The complexity of *DBSCAN* algorithm with R*-tree is $O(n \log n)$. Each of the P computers runs *DBSCAN* in parallel with a dataset of size $\frac{n}{P}+v$, where v is the average number of objects present in an overlapped region. Therefore complexity of *DBSCAN* now becomes $O((\frac{n}{P}+v)\log(\frac{n}{P}+v))$. The time complexity for load balancing step will be $O(n)$. For ideal cases time complexity for data placement step can be taken to be $O(n)$. Global clustering step will visit each object of each overlapped area to determine clusters to be merged. Also it will update global clusters for each object of the local clusters. So complexity of global clustering step will be $O(P.v + n)$. Therefore the overall complexity can be obtained as $O((\frac{n}{P}+v)\log(\frac{n}{P}+v) + P.v + n)$. If P is increased with increase in n so that n/P remains manageable by a single processor the algorithm will remain scalable.

4.4 Performance Evaluation

A sequential algorithm is evaluated in terms of its execution time expressed as a function of its input size. On the other hand the execution time of a parallel algorithm depends not only on the input size but also on the parallel architecture and the number of processors employed. By adding more processors we would like to decrease the execution time or increase the volume of data handled. We now empirically study the desirable characteristics of our parallel algorithm by measuring execution time, speedup, efficiency and scaleup factors. The performance of the parallel *DBSCAN* will be measured relative to the sequential *DBSCAN*. The sequential *DBSCAN* is very slow for larger datasets. So, we have limited the dataset size to maximum of 800000 2-dimensional data objects in our experiments. Since the datasets are limited, we limit the number of processors used to maximum of 6 only. We have synthetically generated different datasets containing 13 arbitrarily shaped clusters, circular, semicircular,

triangular, rectangular, S-shaped etc. Datasets of sizes 100000, 200000, 300000, 400000, 500000, 600000 and 800000 are created containing the same 13 clusters but density increasing gradually. The structure of the datasets was shown in *Figure 3.2*.

Since there is no inter-processor communication except for a single processor communicating with each of the remaining processors, we used client-server computing. Programs are developed using C++ in LINUX environment. Each processor has the same specification Pentium III with 1.0 Ghz speed and 128 MB RAM. The processors are connected through a 10/100 Mbit Ethernet LAN. To smooth out any fluctuations each measurement was repeated 5 times and the average was taken.

1. *Parallel Execution Time* : The parallel execution time, denoted by $T(P)$, of a program is the time required to run the program on a P -processor parallel computer. When $P = 1$, $T(1)$ denotes the sequential run time of a program on a single processor. *Figure 4.4* shows the graph of execution time versus number of processors used. It can be seen that the execution time decreases significantly as the number of processors increase.
2. *Speedup* : A measure of relative performance between a multi-processor system and a single processor system is the speedup factor defined as $S(P) = T(1)/T(P)$. Ideally a system with m processors should yield a speedup of m (linear speedup) when the sequential algorithm that is parallelize is of linear ($O(n)$) complexity. However, parallelizing a linear algorithm, linear speedup is difficult to achieve because of the communication cost and speed difference of the processors. The relation between speedup and number of processors used is shown in *Figure 4.5*. Note that for our algorithm $S(P) > 1$, i.e. super-linear speedup is achieved. Because the sequential algorithm adapted here is of complexity $O(n \log n)$. This can be compared to the sub-linear speedup reported by

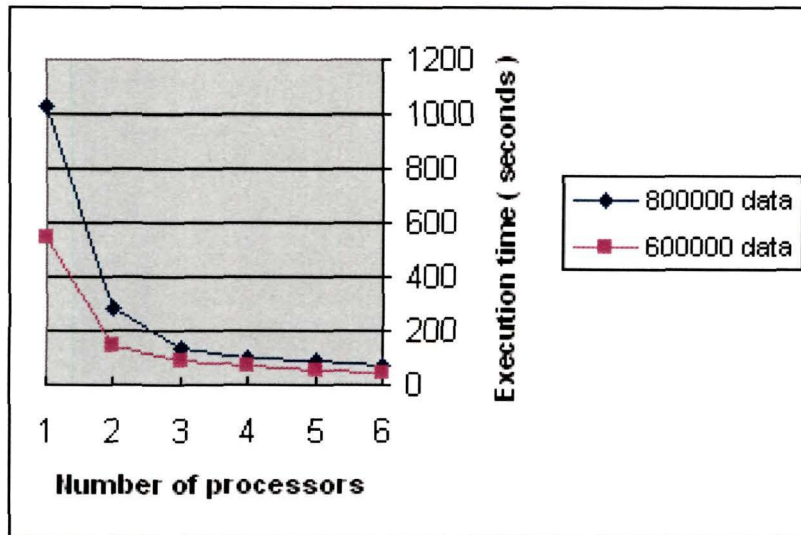


Figure 4.4: Execution time on different number of processors.

Xu et al. [XJK99] for *PDBSCAN*. Our algorithm is better as it achieves higher speedup.

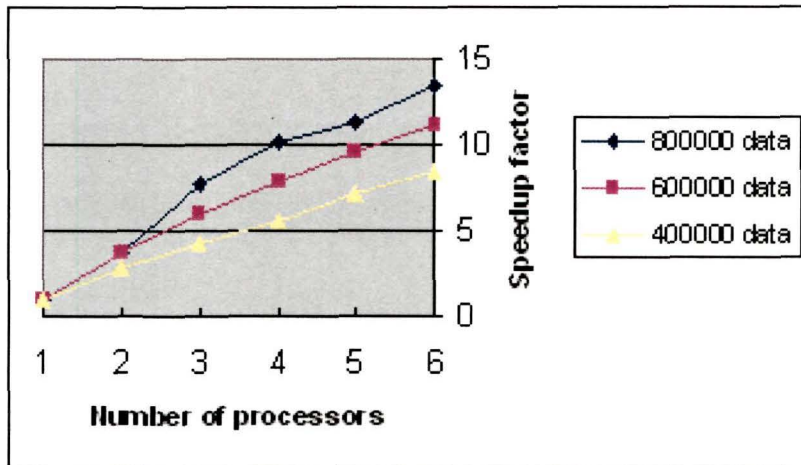


Figure 4.5: Speedup of parallel *DBSCAN* on different number of processors

3. *Efficiency* : Speedup does not measure whether the processors in a parallel computer are being used efficiently. The efficiency of a program on P processors, $E(P)$, is defined as the ratio of speedup achieved and the number of processors used to achieve it. Thus, $E(P) = S(P)/P = T(1)/(P.T(P))$ For our parallel program $E(P)$ is shown in *Figure 4.6*. It is seen from the graph that if too many processors are used then efficiency is dropped.

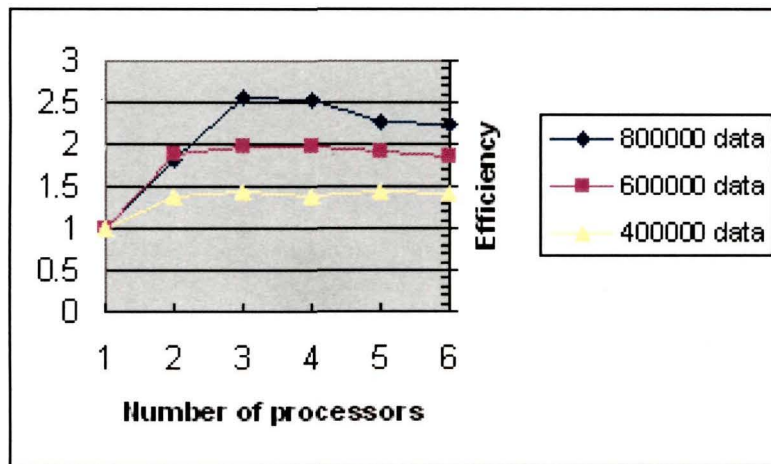


Figure 4.6: Efficiency vs. number of processors employed

4. *Scaleup*: Another figure of merit of a parallel algorithm is scaleup which captures how well the parallel algorithm handles larger datasets when more processors are available. Scalability has been a rather imprecise term. Our scalability study measures execution time by keeping the problem size per processor fixed while increasing the number of processors. The scaleup characteristic for the proposed parallel *DBSCAN* is shown in *Figure 4.7*. It is clear that the algorithm scales well.

The experimental results presented here demonstrate that the proposed algorithm

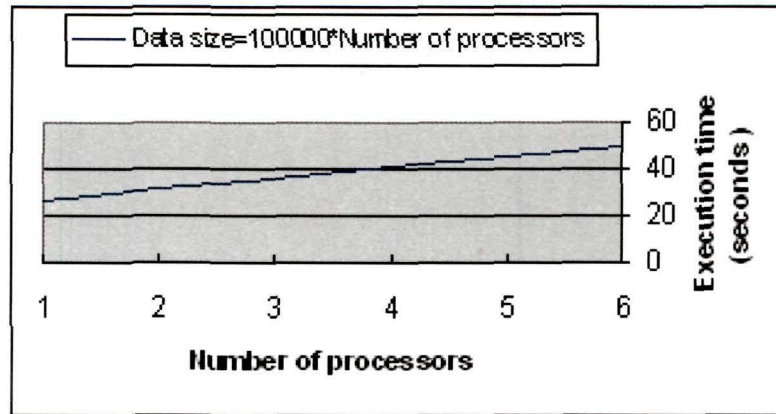


Figure 4.7: Scaleup of parallel *DBSCAN* on different sets of data scaled by the number of processors

is indeed a feasible approach to parallel clustering. The algorithm is found to be scalable both in terms of speedup and scaleup. Thus, large spatial datasets can be clustered efficiently.

In the previous chapter and the present one we extended *DBSCAN* to cluster larger datasets. In the next chapter we concentrate on extending the algorithm so that nested clusters, i.e. clusters within a cluster, can be extracted based on density difference.

Chapter 5

DDSC: A Density Differentiated Spatial Clustering Technique

5.1 Introduction

Although many algorithms exist for finding clusters with different sizes and shapes, there are a few algorithms that can detect clusters with different densities. Basic density based clustering techniques such as *DBSCAN* and *DENCLUE* [HK98] treat clusters as regions of high densities separated by regions of no or low densities. So they are able to suitably handle clusters of different sizes and shapes besides effectively separating noise (outliers). But they fail to identify clusters with differing densities unless the clusters are separated by sparse regions. For example, in the dataset shown in Figure 5.1, *DBSCAN* finds a single cluster instead of finding the three distinct clusters that can be visualized based on density.

We propose an extension of the *DBSCAN* algorithm to detect clusters with differing densities. Extracted clusters are non-overlapped spatial regions such that within a region the density is reasonably homogeneous. Adjacent regions are

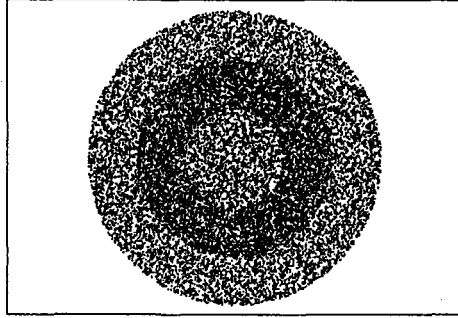


Figure 5.1: Clusters with varying densities.

separated into different clusters if there is significant change in densities. The clusters may be contiguous i.e. not separated by any sparse region as required by *DBSCAN*. Thus natural clusters in a dataset can be extracted. An added advantage is that the sensitivity of the input parameter ϵ , which is an important disadvantage of *DBSCAN*, is reduced significantly.

5.2 Related Works

The *DBSCAN* [EKSX96] is a basic density based clustering algorithm. The density associated with an object is obtained by counting the number of objects in a region of specified radius, ϵ , around the object. An object with density greater than or equal to a specified threshold, *MinPts*, is treated as core (dense), otherwise non-core (sparse). Non-core objects that do not have a core object within the specified radius are discarded as noise. Clusters are formed around core objects by finding sets of density connected objects that are maximal with respect to density-reachability. *DBSCAN* can find clusters with variable sizes and shapes, but there may be wide variation in local densities within a cluster since it uses global density parameters *MinPts* and ϵ , which specify only the lowest possible density of any cluster.

To find clusters that are naturally present in a dataset different local densities need to be identified and separated into clusters. The *OPTICS* [ABKS99] algorithm adopts *DBSCAN* to achieve this goal. The proposed algorithm also extends *DBSCAN* in a different manner to achieve the same goal. *OPTICS* computes an ordering of the objects augmented by reachability distance, representing the intrinsic hierarchical clustering structure. This cluster ordering, displayed by the so called reachability-plots, is the basis for both automatic and interactive cluster analysis. Valleys in this plot indicate clusters. The parameter ξ is crucial for identifying the valleys as ξ -clusters.

DENCLUE (DENsity CLUstEring) [HK98] takes a more formal approach to density based clustering by modeling the overall density of a set of objects as the sum of influence functions associated with each object. The resulting overall density function will have local peaks, i.e., local density maxima, and these local peaks can be used to define clusters in a straightforward way. Specifically, for each data object, a hill climbing procedure finds the nearest peak associated with that object, and the set of all data objects associated with a particular peak (called a local density attractor) becomes a (center-defined) cluster. However, if the density at a local peak is too low, then the objects in the associated clusters are classified as noise and discarded. Also, if a local peak can be connected to a second local peak by a path of data objects, and the density at each object on the path is above a minimum density threshold, ξ , then the clusters associated with these local peaks are merged. Thus, clusters of any shape can be discovered. It has trouble with data that contains clusters of widely different densities.

CHAMELEON [KHK99] and *SNN* [ESK03] algorithms attempts to obtain clusters with variable sizes, shapes and densities based on k -nearest neighbour graphs. *CHAMELEON* finds the clusters in a dataset by using a two-phase algorithm. In the first phase it generates a k -nearest neighbour graph that contains links between a point and its k -nearest neighbours. This approach reduces the influence of noise and outliers and provides an automatic adjustment

for differences in densities. Then it uses a graph partitioning algorithm to cluster the data items into a large number of relatively small sub-clusters. During the second phase, it uses an agglomerative hierarchical clustering algorithm to find the genuine clusters by repeatedly combining sub-clusters. No cluster can contain less than a user specified number of instances. It has problems when the partitioning process does not produce sub-clusters.

The *SNN* (Shared Nearest Neighbour) clustering algorithm uses k -nearest neighbour approach to density estimation. It constructs a k -nearest neighbour graph in which each data object corresponds to a node which is connected to the nodes of the k -nearest neighbours of that data object. From the k -nearest neighbour graph a shared nearest neighbour graph is constructed, in which edges exist only between data objects that have each other in their nearest neighbour lists. A weight is assigned to each edge based on the number and ordering of shared neighbours. Clusters are obtained by removing all edges from the shared nearest neighbour graph that have a weight below a certain threshold τ . *SNN* can detect clusters of different sizes, shapes and densities.

The clustering techniques stated above try to find clusters with variable sizes, shapes and densities. The proposed algorithm is an alternative to these algorithms. It is simpler and produces good quality results consuming less execution time. For example *OPTICS* produces an ordering of the objects by performing k -NN queries in the first step and then it produces variable density clusters using a second step requiring more execution time. *DENCLUE* and *SNN* use several parameters, proper tuning of the parameter values is very important for getting good quality results.

5.3 Proposed Algorithm

The proposed algorithm partitions given dataset into a set of spatial regions (clusters) such that adjacent regions significantly differ in density. Lesser amount of local density variations exist within a cluster, but going from the present region to a neighbouring region greater amount of local density variation will be noticed.

As before, given numeric dataset, X , consists of n d -dimensional objects represented by x_{ij} , $i = 1, 2, \dots, n$, $j = 1, 2, \dots, d$. Recall that the neighbourhood within a given radius ϵ of an object p is represented by $N_\epsilon(p) = \{q \in X \mid \text{dist}(p, q) \leq \epsilon\}$. It is spherically shaped for Euclidean distance function $\text{dist}(p, q)$. The neighbourhood size of an object p i.e. $|N_\epsilon(p)|$ represents the density around it. Let us use a list w_p , $p = 1, 2, \dots, n$ to store density of each object in the dataset X . Initially, density of each object is unknown, which is represented by $w_p = -1$, $p = 1, 2, \dots, n$. When neighbourhood query is performed, density of p is assigned as $w_p = |N_\epsilon(p)|$. Object p is called a core object if $w_p \geq \text{MinPts}$. The dataset is to be partitioned into a set of non-overlapped clusters. Let us denote the cluster label of p by c_p . Initially all objects are assigned the label -1 to indicate unlabeled objects, that is $c_p = -1$, $\forall p \in \{1, 2, \dots, n\}$.

For detection of clusters separated by density variations the concepts of processed objects, candidate objects, unprocessed objects and homogeneous core objects are required. The definitions are presented below.

- A *processed object* p is one, whose density is already evaluated, i.e. $w_p \geq 1$. Evaluating the density of an object by performing neighbourhood query is called processing.
- A *candidate object* is already included in a cluster, but its density is yet to be evaluated, i.e. $c_p \geq 0$ and $w_p = -1$.
- An *unprocessed object* p has $w_p = -1$, $c_p = -1$, that is its density as well

as cluster label are not evaluated.

- A *homogeneous core object* p is a core object ($w_p \geq MinPts$) such that the density of any of its neighbours does not differ with respect to the density of the core object itself by more than a specified threshold α . That is, $\forall q \in N_\epsilon(p)$, $w_p/w_q \leq \alpha$ if $w_p \geq w_q$ or $w_q/w_p \leq \alpha$ if $w_p < w_q$ where $\alpha > 1$ is a constant.

The algorithm starts a cluster with a homogeneous core object and goes on expanding it by including other directly density reachable homogeneous core objects until non-homogeneous core objects, that indicate wide variation in densities, are detected. An ordering is imposed upon the sequence in which the objects will be processed while expanding a cluster.

5.3.1 Ordered expansion process

A new cluster is created with a core object and its neighbours, that are inserted into the seeds-list. This initial cluster is expanded when each object in the seeds-list is processed in turn. Objects are deleted from the front end of the seeds-list for processing while new members are entered at the back end. When an object is processed it may contribute some new objects which are ordered before entering into the seeds-list. The following are the steps for ordered-processing of an object p taken out from the seeds-list.

1. If p is a core object perform steps 2-5;
2. Find the list L_p of unlabeled objects in $N_\epsilon(p)$: $L_p = \{q \mid q \in N_\epsilon(p), c_q = -1\}$;
3. Arrange the objects in L_p in ascending order of their distance to p to obtain the sorted list $L'_p = \{q_1, q_2, q_3, \dots, q_t\}$ with size $t = |L_p|$, $q_0 = p$ such that: $L'_p = \{q_i \mid q_i \in L_p, i = 1, 2, \dots, t, dist(q_{i-1}, p) \leq dist(q_i, p)\}$;

4. Append L'_p to seeds-list;
5. Mark all unlabeled and noise objects in $N_\epsilon(p)$ with present *cluster-id*:
 $\forall q \{q \in N_\epsilon(p), c_q \leq 0\} : c_q = \text{cluster-id};$

Steps 2–3 impose an ordering on the seeds for entering into the seeds-list. Steps 4–5 cause expansion of the cluster. The ordered expansion process of a cluster iteratively deletes an object from the seeds-list and performs ordered processing (steps 1–5) for each deleted object until the list becomes empty, when detection of a cluster completes.

This ordered expansion process has some important properties as presented in the lemmas to follow. In *DBSCAN*, unlabeled neighbours are inserted into the seeds-list in the order in which they are obtained. So already processed objects and candidate objects (waiting in the seeds-list to be processed) are intermixed in the same spatial region. In the discussions to follow we consider 2-dimensional objects for simplicity in graphical presentation, although the ideas are applicable to higher dimensions as well.

Lemma 4 *During ordered expansion process, already processed objects form a spatial region which is contiguous and non-overlapped with the region formed by candidate objects.*

Proof : When a cluster is first created by processing a core object o , all its neighbours inside the circle of radius ϵ become candidates to be processed next. Presently, there is only a single object o in the region of already processed objects, which is surrounded by the region formed by candidate objects. The region of already processed objects grows as candidates become processed and contribute some new candidates. Consider that the next object to be processed currently is p . Let, q be the object which has contributed p to the seeds-list i.e. $p \in N_\epsilon(q)$ and q is already processed. Let, $s = \text{dist}(p, q)$. Draw a circle with radius s centered

at q . All objects inside the circle will be already processed objects, because according to the ordered expansion process, each object r inside the circle will be processed before p , which is lying on the circle, since $dist(r, q) < dist(p, q)$. This contiguous region of already processed objects grows by one object after including the currently processed object p . So, the region will still remain contiguous after inclusion of the object p . \square

At least one processed object is present in the neighbourhood of currently processed object. There is a maximum limit for the number of processed objects that may be present in the neighbourhood of currently processed object.

Lemma 5 *For uniformly distributed objects at most 50% neighbours in the neighbourhood of the currently processed object are already processed.*

Proof : Consider that o is the first core object detected for expanding a cluster. Ordered expansion procedure processes the objects one by one starting from the nearest neighbour of o , initial few objects have less than 50% already processed objects in their neighbourhoods at the time of processing them. Let the currently processed object p , lying on the circle c_1 with radius ϵ centered at o as shown in *Figure 5.2*, be the farthest object in the neighbourhood of object o . The neighbourhood of p is shown by circle c_2 . The area of intersection of the two circles ($N_\epsilon(o) \cap N_\epsilon(p)$) contains already processed objects. Using the formula for circular segment¹, the area of intersection [Wei] of the two circles is calculated to be 39% of the area of circle c_2 . Assuming uniform distribution of objects and $m = |N_\epsilon(p)|$, this region will contain 0.39m objects. Here, p is selected such that the present region of already processed objects is small enough to be included inside the circle of radius ϵ . But as the cluster grows the region of already processed objects grows in size. Then the already processed objects and candidate objects in the neighbourhood of the currently processed object can be separated

¹ $A(R, d) = R^2 \cos^{-1}(d/R) - d\sqrt{R^2 - d^2}$, R is the radius, d is distance of the segment from the center

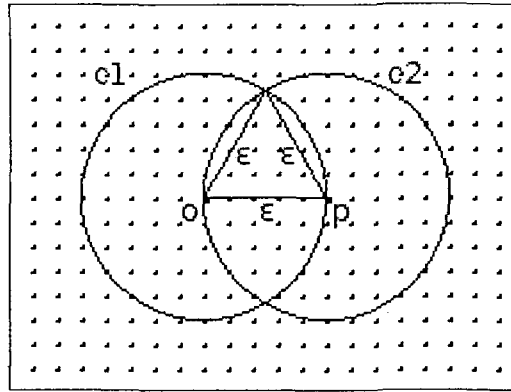


Figure 5.2: Already processed objects within the neighbourhood of currently processed object p .

with an arc of a circle of larger radius. When the cluster becomes bigger this boundary can be a straight line, in which case 50% of the neighbourhood of the currently processed object will be already processed. \square

Proposed algorithm does not require that objects are uniformly distributed. It detects clusters that are reasonably homogeneous i.e. some amount of density variation is allowed within a cluster. Significant variation of density will cause separate clusters to be identified. *Lemmas 4 & 5* provide us an approach for detecting density variations while a cluster is being expanded. The density of each of the already processed object is known as its density value was stored at the time of processing it. So, we can ensure that the density of the current object processed should not differ much with those of already processed objects in its neighbourhood, otherwise this current object should not be expanded i.e. previously unclustered objects found in its neighbourhood should not be added to the seeds-list. Below we formalize this homogeneity test.

5.3.2 Homogeneity test

Let, p be the current object being processed and L_p be the list of already processed objects ($w_p \geq 1, \forall p \in L_p$) present in the neighbourhood of p . The current object p is homogeneous to the region of already processed objects if the following conditions hold for each $q \in L_p$:

$$\frac{w_p}{w_q} > \alpha_1 \text{ if } w_q \geq w_p \quad (5.1)$$

$$\frac{w_q}{w_p} > \alpha_2 \text{ if } w_q < w_p \quad (5.2)$$

In the *Inequalities 5.1* and *5.2* $\alpha_1, \alpha_2 \in (0, 1]$ are two constants indicating allowed density difference limits within the neighbourhood of an object. The values of α_1 and α_2 can be determined based upon an input parameter α as described below.

Let us consider two contiguous uniformly distributed regions R_1 and R_2 as shown in *Figure 5.3*, such that R_2 is α times denser than R_1 , with $\alpha > 1$. The minimum density difference required for separating clusters is indicated by α . If the density difference is less than α , the two regions will be merged into a single cluster. Assume that the current object to be processed, p is located at the boundary of the two regions. Consider two objects $q \in R_1$ and $r \in R_2$ such that $dist(p, q) = dist(p, r) = \epsilon$ and p, q, r are in a straight line. Let, $w_q = |N_\epsilon(q)| = m$. Then, $w_r = |N_\epsilon(r)| = \alpha m$, and $w_p = |N_\epsilon(p)| = (1 + \alpha)\frac{m}{2}$. Density of any object between q and p will be higher than m but less than $(1 + \alpha)\frac{m}{2}$. Similarly, density of each object between p and r will be higher than $(1 + \alpha)\frac{m}{2}$ but less than αm . So, the objects between q and r form a transition (bordering) region containing objects with different densities. When a transition region is encountered cluster expansion in that direction may get stopped. A transition region may be encountered while going from a lower density region to a higher density one or from a higher density region to a lower density one. So, two different density factors α_1 and α_2 are needed to avoid order dependency. Values

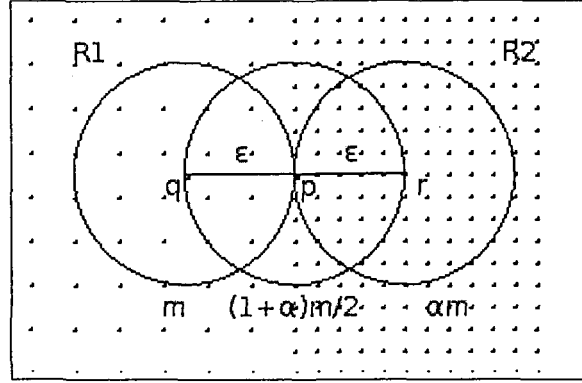


Figure 5.3: Density variation pattern produced by two adjacent regions with different densities.

of the two factors can be computed based on the object p . While expanding a cluster, if a lower density region is entered, the density difference limit between the density of the current object with any of the already processed objects in its neighbourhood, α_1 is computed as :

$$\alpha_1 = \frac{w_p}{w_r} = \frac{(1 + \alpha) \frac{m}{2}}{\alpha m} = \frac{1 + \alpha}{2\alpha} \quad (5.3)$$

Similarly, entering a higher density region, the density difference limit α_2 is computed as :

$$\alpha_2 = \frac{w_q}{w_p} = \frac{m}{(1 + \alpha) \frac{m}{2}} = \frac{2}{1 + \alpha} \quad (5.4)$$

The two factors α_1 and α_2 determines the allowed variation in local density within a cluster so that the density of the cluster can be called relatively homogeneous.

Above, we have stated about the maximum density difference allowed for a single object to be called homogeneous to the region of already processed objects. To stop growth of a cluster in any spatial direction a non-homogeneous region of width at least ϵ should be encountered in that direction. The following lemma establishes the idea.

Lemma 6 *The growth of a cluster in any spatial direction is stopped if a non-homogeneous region of width at least ϵ is encountered in that direction.*

Proof : Referring back to *Figure 5.3*, object p is the current object being processed. Object p becomes non-homogeneous, if in the neighbourhood of p there is at least one already processed object that crosses the allowed density variation limit. Let, the region $N_\epsilon(q) \cap N_\epsilon(p)$ contain processed objects and $w_q/w_p \leq \alpha_2$, causing object p to become non-homogeneous. Then p will not be expanded but growth of the present cluster can not be stopped by p alone. Since, there are some candidates for expanding the cluster lying after p and those candidates were contributed by the objects present between q and p when they were processed. These candidate objects form a region of width at most ϵ , that is spread up to just before object r . To stop growth of the cluster in the direction of q to r , none of these objects should expand when processed. That is, each of these objects should become non-homogeneous because of presence of some predecessors, lying between q and p , that have density difference greater than allowed limit. This will really be the case if the region between p and r (region R_2) is denser than the region between q and p (region R_1) by a factor greater than α . \square

From *lemma 6* it becomes clear that a cluster extends beyond its expected boundary as some non-homogeneous objects (border objects) are also included in the cluster. It is because we are performing homogeneity test only on one part of the neighbourhood. We cannot test the remaining part simultaneously because density information of these objects will be obtained only when they are processed. Another problem is that the region of already processed objects falling in the neighbourhood of currently processed object may contain very few objects that may lead to the single linkage effect. To alleviate these two problems we impose the following requirements on the currently processed object. We call it cardinality test.

5.3.3 Cardinality test

The number of already processed objects present in the neighbourhood of currently processed object should be within a certain minimum and maximum limits. The maximum limit is taken to be 50% of the neighbourhood size based on *lemma 5*. The volume of intersection of two d -dimensional hyper spheres with radius ϵ situated at a distance of ϵ apart gives the minimum limit for d -dimensional data objects. The situation is shown for 2-dimensional data in *Figure 5.2*. The area of intersection for two circles is approximately 39% of the area of a circle. For two spheres the volume of intersection is approximately 31% [Wei]. As the dimension increases this volume decreases. We take the minimum limit to be $\frac{1}{1+d}\%$, where d is the dimension of the data objects. Consider currently processed object p in *Figure 5.3*. Proceeding from q to p i.e. going from lower to higher density, minimum possible number of already processed objects contained in the neighbourhood of p are $\frac{m}{1+d}$. Similarly, proceeding from r to p i.e. going from higher to lower density, maximum possible number of already processed objects contained in the neighbourhood of p is $\frac{\alpha m}{2}$. So, the two limits expressed as a fraction to the density of the currently processed object are

$$\beta_{min} = \frac{\frac{m}{1+d}}{(1+\alpha)\frac{m}{2}} = \frac{2}{(1+d)(1+\alpha)} \quad (5.5)$$

$$\beta_{max} = \frac{\frac{\alpha m}{2}}{(1+\alpha)\frac{m}{2}} = \frac{\alpha}{1+\alpha} \quad (5.6)$$

5.3.4 Special treatment for the first core object

The homogeneity test and cardinality test are not applicable to the starting core object of the cluster, as no objects of the cluster are processed before it. However, it must be ensured that the first object does not lie at the boundary of two widely differing density regions. In fact, it must not lie within a distance of $\epsilon/2$ from the boundary. Otherwise the two differing density regions will be merged into a

single cluster. Because, a non-homogeneous region of width at least ϵ will not be encountered in that case to stop the growth of the cluster across the boundary as required by *lemma 6*.

To avoid this problem we reject the outer neighbours of the first core object and enter into the seeds-list only those neighbours that lie within a radius of $\epsilon/2$ from the object. Cardinality test is also not applied while these few seeds are expanded.

5.4 The Algorithm

The steps in *DDSC* clustering algorithm are listed below.

Algorithm DDSC

Inputs: $X, n, d, \epsilon, MinPts, \alpha$;

Outputs: Cluster label for each of the n object in X ;

Steps:

01. Set $cluster-id=0$;
02. FOR $p=1$ TO n DO
03. {Set $w_p=-1$;
04. Set $c_p=-1$;
05. }
06. Compute :
 - $\alpha_1 = (1 + \alpha)/(2 * \alpha)$;
 - $\alpha_2 = 2/(1 + \alpha)$;
 - $\beta_{min} = 2/((1 + d) * (1 + \alpha))$;
 - $\beta_{max} = \alpha/(1 + \alpha)$;

```

07. For  $p=1$  TO  $n$  DO
08.   {IF ( $c_p == -1$ ) THEN
09.     {Find  $N_\epsilon(p)$ ;
10.     Set  $w_p = |N_\epsilon(p)|$ ;
11.     IF ( $w_p < MinPts$ ) THEN
12.       Set  $c_p=0$ ; // noise object
13.     ELSE
14.       {Set  $cluster-id=cluster-id+1$ ;
15.       Set  $c_p=cluster-id$ ;
16.       Find  $R = \{q \mid q \in N_\epsilon(p), dist(p, q) \leq \epsilon/2, c_q = -1\}$ ;
17.       Append list  $R$  to  $seeds-list$ ;
18.       Set  $m = |R|$ ;
19.       FOR  $i=1$  TO  $m$ 
20.         {Delete an object  $q$  from  $seeds-list$ ;
21.         Set  $w_q = |N_\epsilon(q)|$ 
22.         Perform ordered-processing for object  $q$ ;
23.         } //END FOR
24.       WHILE ( $seeds-list$  is NOT EMPTY) DO
25.         {Delete an object  $q$  from the  $seeds-list$ ;
26.         Set  $w_q = |N_\epsilon(q)|$ ;
27.         IF ( $w_p \geq MinPts$ ) THEN
28.           IF (homogeneity-test( $q, \alpha_1, \alpha_2$ ) succeeds) THEN
29.             IF (cardinality-test( $q, \beta_{min}, \beta_{max}$ ) succeeds) THEN
30.               perform ordered-processing for  $q$ ;
31.             END IF;
32.           END IF;
33.         END IF;
33.         } // END WHILE
34.       } // END IF

```

```
35.     } // END IF
36.   } // END FOR
37. END DDSC;
```

5.4.1 Complexity analysis

The most time consuming part of the algorithm is the neighbourhood queries. The neighbourhood size is expected to be small compared to the size of the dataset. So, the different tests performed on the neighbourhood will not consume much time. While expanding a cluster the list of newly contributed seeds by each object of the cluster need to be sorted. For all objects only a small fraction of the neighbours become new seeds, whereas some objects contribute no new seeds at all. Sorting the lists will not consume much time as the size of the list to be sorted is small. The time required for a neighbourhood query is $O(\log n)$ by using a spatial access method such as R*-tree. Neighbourhood query is performed for each of the n objects in the dataset. So the run time complexity is $O(n \log n)$.

5.5 Experimental Results

In this section we evaluate the performance of the *DDSC* and compare the result with *CHAMELEON* and *SNN* algorithms. We implemented the algorithm in C++. Experiments were conducted on a 1.66 GHz HCL laptop with 512 MB RAM running LINUX operating system.

Synthetic datasets are used in the experiments. The *CHAMELEON* datasets - *t4.8k.dat*, *t7.10k.dat*, *t8.8k.dat* and *t5.8k.dat*, used in [KHK99] are downloaded from [gla]. We have created two dataset - *Dataset1* shown in *Figure 5.1* and *Dataset2* shown in *Figure 5.4*. *Dataset1* contains 24000 objects arranged in three nested circular regions, the middle region being twice as much

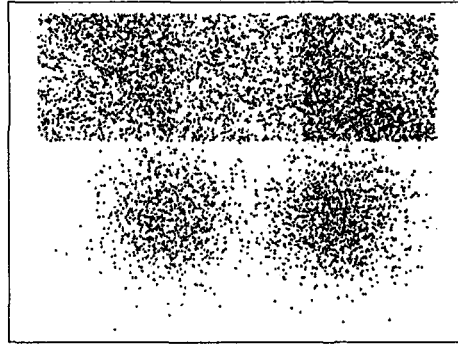


Figure 5.4: Dataset2

denser than the neighbouring ones. *Dataset2* contains 8100 objects. Four triangular regions and a rectangular region are generated such that a region is at least two times denser than the neighbouring regions. These can be visualized in the upper part of the figure. Local density variations are present within each region. Two regions are produced using Gaussian density generator.

The clustering results for *Dataset1* and *Dataset2* are shown in *Figures 5.5* and *5.6*. Different colours are used to indicate the clusters. It can be seen from the figures that the circular, triangular and rectangular clusters are extracted based on differences in densities although they are not separated by sparse regions. The three nested clusters with different densities in *Dataset1* are properly extracted. Bigger portions of the Gaussian clusters in *Dataset2* are also detected, which means that inside a cluster the local densities may gradually change within limits, bigger changes prevent the expansion of the clusters.

Figures 5.7– 5.10 show clustering result of our algorithm on *CHAMELEON* datasets : *t4.8k.dat*, *t7.10k.dat*, *t8.8k.dat* and *t5.8k.dat* respectively. Clusters with different sizes, shapes and densities are extracted and noises are discarded. Similar results were reported for *CHAMELEON* [KHK99] and *SNN* [ESK03] algorithms.

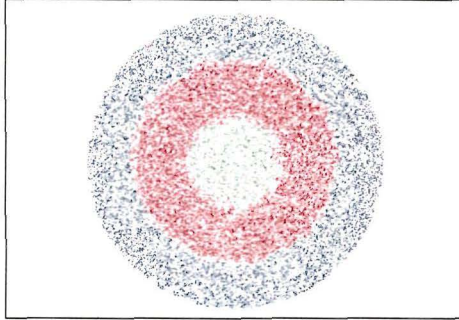


Figure 5.5: Result on *Dataset1*

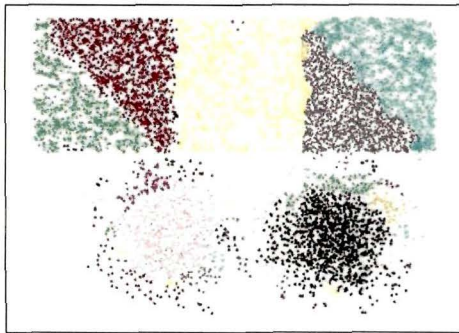


Figure 5.6: Result on *Dataset2*

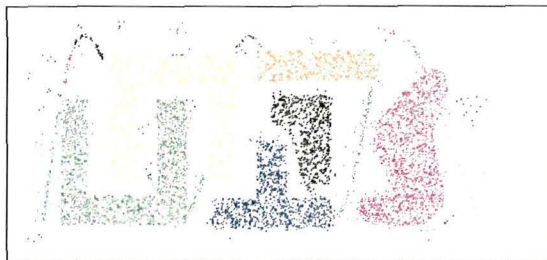


Figure 5.7: Result on *t4.8k.dat*

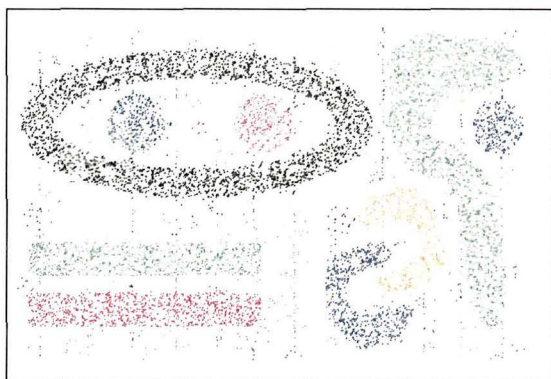


Figure 5.8: Result on *t7.10k.dat*

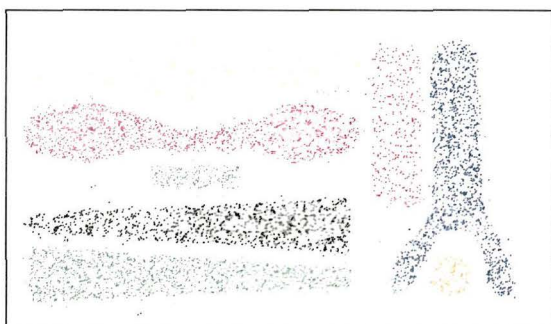


Figure 5.9: Result on *t8.8k.dat*

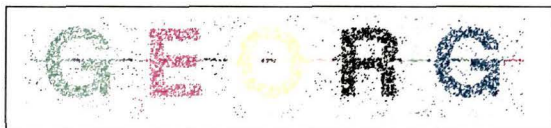


Figure 5.10: Result on *t5.8k.dat*

5.5.1 Discussion on Parameters

We have performed several experiments on the datasets to study the effects on changing values of the parameters α , ϵ and $MinPts$. It is observed that, the proposed algorithm is less sensitive to the input parameters ϵ and $MinPts$. For example, *DBSCAN* produces the clustering result shown in *Figure 5.8* for the dataset *t7.10k.dat* with ϵ values in the range [5.7, 6.1] and $MinPts=4$ only. It shows that accuracy of result of *DBSCAN* depends on proper selection of parameter values within a narrow range of possibility. But proposed algorithm produces this result for ϵ in the range [13.0, 17.0] and $MinPts$ in the range [4, 29]. For increased values of $MinPts$, some very small clusters may be treated as noise. For example the result for *t7.10k.dat* with $MinPts=29$ and $\epsilon=17$ is shown in *Figure 5.11*. Here the smaller clusters found in *Figure 5.8* are not present. If value of α is increased significantly allowing more density variations, adjacent clusters may be merged together. On the other hand significant decrease in α value, increases the number of clusters as bigger clusters are broken down. Smaller change in α values does not cause noticeable change in the detected clustering structure. Thus *DDSC* algorithm offers a wide range for choosing the parameter values, increasing the scope of getting correct result even if parameter values are not selected very carefully.

We have repeated the above mentioned experiments several times, each time the objects of the dataset are shuffled randomly. The results produced the same set of clusters except changes in cluster memberships of a few bordering objects.

All these results show that our algorithm can find clusters with variable sizes, shapes, and densities. Noises are also properly separated.

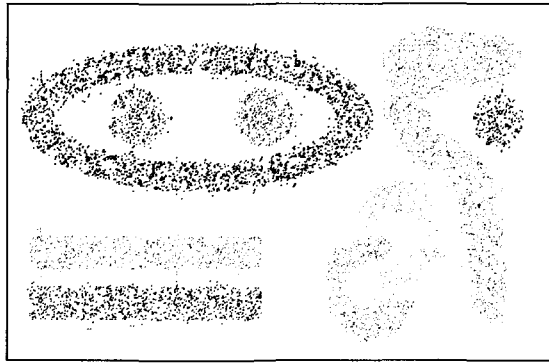


Figure 5.11: Result on *t7.10k.dat* with increased *MinPts*

Part III

Clustering Categorical Data

We have developed a scalable, subspace based algorithm for clustering high-dimensional categorical data. The algorithm is presented in *Chapter 6*.

Chapter 6

CatSub: Clustering Categorical Data Based on Subspace

6.1 Introduction

Many algorithms [And02] have been developed for clustering numeric data, based on the use of similarity measures that exploit inherent geometrical structures of numeric data. Comparatively lesser number of studies have focused on clustering categorical data, where the domains of the individual attributes are discrete valued and not naturally ordered and therefore distance functions are not naturally defined. Moreover, categorical datasets are generally high dimensional. Most of the common clustering algorithms fail to perform efficiently and accurately for high dimensional data, because such dataset do not exhibit clusters over the full set of dimensions. Many of the dimensions are often irrelevant or correlated and different clusters may have different subsets of relevant dimensions. Subspace clustering algorithms [PHL04] find a subset of relevant dimensions for each cluster. Subspaces of different clusters are almost always allowed to be overlapping. Some algorithms allow the clusters to be overlapping, while others

find a set of disjoint clusters that cover the entire dataset. Some algorithms also detect outliers, which are the objects that do not belong to any of the clusters.

This chapter presents our algorithm named *CatSub* (Clustering Categorical Data Based on Subspace) to efficiently cluster large, high dimensional datasets containing categorical attributes. We define a similarity measure and a searching strategy to find relevant subspaces and corresponding clusters. Outliers, if any, are detected along with the set of disjoint clusters. The proposed algorithm scales well to larger datasets as it requires only a single scan of the dataset which need not be stored in main memory.

6.2 Related Works

Huang [Hua98] proposed the *k-modes* algorithm to tackle the problem of clustering large categorical datasets in data mining. The *k-modes* algorithm extends the *k-means* algorithm by using a simple matching dissimilarity measure for categorical objects, modes instead of means for clusters, and a frequency based method to update modes in the clustering process to minimize the clustering cost function. However, due to non-uniqueness of the modes the clustering results depend strongly on the selection of modes during the clustering process.

CACTUS [GGR99] computes summary information from the dataset and use them to discover a set of candidate clusters which are then validated to determine the actual set of clusters. The algorithm uses two scans over the dataset and can find clusters in subsets of attributes. The disadvantage of this algorithm is the fast increase in running time when the number of dimensions grow.

Guha et al. [RGS99] introduced *ROCK*, an adaptation of an agglomerative hierarchical clustering algorithm, which heuristically optimizes a criterion function defined in terms of the number of links between tuples. Informally, the number of links between two tuples is the number of common neighbours they

have in the dataset. Starting with each tuple in its own cluster, they repeatedly merge the two closest clusters till the required number of clusters remain. Since the complexity of the algorithm is cubic in the number of tuples in the dataset, they cluster a sample randomly drawn from the dataset, and then partition the entire dataset based on the clusters from the sample.

STIRR [GKR88] is a categorical clustering method, investigated in terms of certain types of non-linear dynamical systems, for assigning and propagating weights on the categorical values in a table. *STIRR* highly depends on the choice of its combining operator, and it produces clusters that might require a heavy post-processing stage.

Squeezer is a scalable categorical clustering algorithm introduced in [HXD02]. The algorithm reads each tuple t in sequence, either assigns t to an existing cluster (initially none), or creates t as a new cluster, which is determined by the similarities between t and a cluster. Number of clusters created by the algorithm may grow faster making the algorithm slower.

Barbara et al. introduced *COOLCAT* [BCL02], a categorical clustering algorithm based on the idea of entropy reduction within the generated clusters. It first bootstraps itself using a sample of maximally dissimilar pairs from the dataset to create initial clusters. The remaining objects are then added incrementally. The authors propose to remove the wrong fitting points at defined times during the execution and re-clustering them. In [LMO04] a Monte-Carlo procedure to find optimal partitions by minimizing an entropy-based criterion is presented for categorical data.

The algorithm, *CLICK* [PZ04] is able to detect subspace clusters in categorical datasets. It finds clusters based on search method for k -partite maximal cliques.

LIMBO [ATMS04] is a scalable hierarchical categorical clustering algorithm that builds on the information bottleneck (IB) framework for quantifying

the relevant information preserved when clustering. The IB framework is used to define a distance measure for categorical attributes and tuples. *LIMBO* handles large datasets by producing a memory bounded summary model for the data.

The *MULIC* [AAW04] algorithm offers major improvements over traditional *k*-modes algorithm, so that the results are more accurate. A preprocessing of the objects in the dataset is performed, that imposes an ordering of the objects. Each cluster consists of layers formed gradually through iterations, by reducing the similarity criterion for inserting objects in layers of a cluster at different iterations.

A technique, *SUBCAD* (SUBspace Clustering for high Dimensional Categorical Data) is presented in [GW04]. An objective function is used to determine the subspace associated with each cluster. A biclustering framework is proposed in [PRB05] to compute a bi-partition from collections of local patterns which capture locally strong associations between objects and properties. A Subspace Clustering Algorithm, *PARTCAT* [GWY06] proposes a neural network architecture for clustering high dimensional categorical data.

6.3 Problem Formulation

The dataset to be clustered, $X = \{X_1, X_2, \dots, X_n\}$ contains n objects, each described by d categorical attributes A_1, A_2, \dots, A_d having finite and discrete valued domains D_1, D_2, \dots, D_d respectively. For each i ($1 \leq i \leq n$) and for each j ($1 \leq j \leq d$) let, x_{ij} be the j -th component of object X_i and x_{ij} take on one of the possible values defined in domain D_j of attribute A_j . Each object is represented as

$$X_i = \{x_{i1}, x_{i2}, \dots, x_{id}\} \in D_1 \times D_2 \times \dots \times D_d \quad (6.1)$$

Let, the i -th and k -th objects be such that $(\forall j \in \{1, 2, \dots, d\} : x_{ij} = x_{kj})$ i.e. the two objects have a common value in each of the attributes. We call an attribute

having a common value over a set of objects to be a matching attribute. Ideally, all attributes in a cluster should be matching attributes. In real life situations, all the attributes may not be equally important for determining cluster membership of an object. Thus in a cluster only a subset of attributes may be matching attributes. Minimum number of such matching attributes (*MinAtt*) needed to form a cluster can be specified. A cluster should also contain at least *MinObj* number of objects. Two clusters may have the same set of matching attributes with differing matching values. It means that the matching value should be indicated along with a matching attribute. Let, a set of matching attribute and value pairs be represented by the set:

$$MAV = \{(j, v) \mid j \subseteq \{1, 2, \dots, d\}, v \in D_j\} \quad (6.2)$$

The set *MAV* together with the set of objects $T \subseteq \{1, 2, \dots, n\}$ represent a cluster *C*, which is the set :

$$C = \{T, MAV\} \quad (6.3)$$

There is no overlap between the objects of any two clusters, but the subspaces consisting of matching attributes may overlap. If $C_1.T$ and $C_2.T$ denote the set of objects in any two of the set of clusters the dataset is partitioned into then $C_1.T \cap C_2.T = \Phi$.

Example: Consider a small dataset shown in Table 6.1 with seven objects defined over five attributes *A*, *B*, *C*, *D* and *E*. The domains for the attributes are respectively, $D_1 = \{a1, a2, a3\}$, $D_2 = \{b1, b2\}$, $D_3 = \{c1, c2, c3, c4\}$, $D_4 = \{d1, d2, d3\}$ and $D_5 = \{e1, e2\}$. Clusters C_1 and C_2 can be identified in the dataset with parameters *MinAtt* = 2 and *MinObj* = 3:

$$C_1 = \{T = \{1, 2, 4\}, MAV = \{(2, b2), (3, c4), (5, e2)\}\}$$

$$C_2 = \{T = \{3, 5, 7\}, MAV = \{(1, a3), (2, b1), (3, c2)\}\}$$

Let us consider a new object (8-th in the dataset) with the values (*a1*, *b2*, *c3*, *d1*, *e2*). This object can be inserted into cluster C_1 so that after

Table 6.1: A sample dataset

<i>Serialno.</i>	A_1	A_2	A_3	A_4	A_5
1	a3	b2	c4	d1	e2
2	a2	b2	c4	d3	e2
3	a3	b1	c2	d1	e1
4	a2	b2	c4	d1	e2
5	a3	b1	c2	d3	e1
6	a1	b2	c1	d2	e2
7	a3	b1	c2	d2	e1

insertion of the object the cluster becomes,

$$C_1 = \{T = \{1, 2, 4, 8\}, MAV = \{(2, b2), (5, e2)\}\}$$

Notice that after inserting the new object the number of matching attributes for C_1 get reduced to 2 which is still not less than $MinAtt$. If $MinAtt > 2$, the object can not be inserted in cluster C_1 .

6.4 Proposed Algorithm

The proposed algorithm *CatSub* finds a set of disjoint clusters and outliers that cover the given dataset. A single-pass incremental algorithm is developed without the need of storing the data objects in main memory. Clusters are determined by the subspaces of matching attributes. It is expected that the clusters found should be of bigger sizes having more objects as well as attributes. Finding such clusters in noisy high dimensional data is a difficult job. A strategy is provided here to find the subspaces and the corresponding clusters in an optimal way. The strategy is based on defining a similarity measure $sim(C, C')$ of a cluster C' with another cluster C so that C' can be merged with C if found similar. For measuring

similarity between a cluster C and an object t a cluster C' is created with t such that :

$$C' = \{T = \{t\}, MAV = \{(1, x_{t1}), (2, x_{t2}), \dots, (d, x_{td})\}\} \quad (6.4)$$

In general, the cluster C' will be a temporary cluster which has not collected the required number of objects to be recognized as a permanent cluster, while the cluster C is an existing cluster. Therefore, the function $sim(C, C')$ need not be symmetrical. The subspace based similarity function is given below.

$$sim(C, C') = \begin{cases} 0 & \text{if } |C.MAV| - m \geq \delta \\ m - MinAtt + \frac{1.0}{(2.0 + |C.MAV| - m)} & \text{otherwise} \end{cases} \quad (6.5)$$

where, $m = |C.MAV \cap C'.MAV|$ is the cardinality of the set of matching attributes that remains if C' is merged with C . The expression $(|C.MAV| - m)$ in *Equation 6.5* computes the reduction in number of matching attributes after the merger. This reduction should be less than a specified threshold, δ , otherwise the similarity value returned should be set to zero. The lesser is the reduction, the higher will be the value of the fractional part of the similarity measure indicating more similarity.

A simple incremental clustering algorithm reads each object t in sequence, inserts t in an existing cluster based upon the similarity between t and the clusters or a new cluster is created with t if it is not similar enough to be inserted in any one of the existing clusters. This procedure may create a large number of smaller clusters making the algorithm slower. The problem becomes more prominent in datasets containing outliers. So, outliers handling procedures are needed.

6.4.1 Outlier handling

Besides a set of valid clusters the algorithm also creates an extra *Outliers* cluster containing outlier objects. Initially a cluster is created with a single object in it.

As similar objects are inserted the number of objects in the cluster may cross the minimum number of objects (*MinObj*) limit, otherwise it will be merged with *Outliers* cluster. In order to prevent outliers from occupying space and consuming search time, we create three different lists of clusters - *CandidateList*, *ClusterList* and *ExtraList*. Elements of each list are clusters as defined by *Equation 6.3*. Creating the three different lists also helps in finding larger subspaces by gradual reduction of the similarity threshold. Unlike *ClusterList* that can grow to any size, *CandidateList* and *ExtraList* are of fixed size as specified by the parameter *MaxSize*. At the beginning of clustering all the lists are empty. The similarity threshold δ defined in *Equation 6.5* takes on three different values - δ_1 , δ_2 and δ_3 for inserting an object in a cluster present in *ClusterList*, *CandidateList* and *ExtraList* respectively. The three thresholds take low, medium and high values in the range $[1, d]$, where d is the number of attributes. An object read from hard disk is first tried for insertion in a cluster present in *ClusterList* with similarity threshold δ_1 allowing for a small or no decrease in the number of matching attributes. If the object could not be inserted in *ClusterList*, then *CandidateList* is tried with threshold δ_2 , which assumes a value less than say 30% of d with a minimum value of 1. Failure in inserting again will invite *ExtraList* for trial with a very loose threshold value δ_3 allowing for much higher decrease in the number of matching attributes. Maximum possible value is $\delta_3 = d - MinAtt$. If the object could not be inserted in a cluster in *ExtraList* also, a new cluster is created with the object and it is inserted in *CandidateList*. When *CandidateList* becomes full, a cluster in it is transferred to *ExtraList* to make room for the new cluster. If *ExtraList* also becomes full with transferred clusters a cluster is removed from it and merged with the *Outliers* cluster. Whenever an object gets inserted in a cluster, present in either *CandidateList* or *ExtraList*, the number of objects in the cluster should be examined. If it collects *MinObj* objects the cluster is transferred to *ClusterList*, which is the list of valid clusters.

After all the objects in the dataset are processed, detected clusters are found in the *ClusterList*. At this time, *CandidateList* and *ExtraList* may contain some clusters with number of objects less than *MinObj*. Now, an attempt is made to merge each of those clusters with the best fit cluster in *ClusterList* using the loose threshold δ_3 . If the merging is not possible, the clusters are merged with the *Outliers* cluster.

The proposed algorithm is presented below.

Algorithm CatSub

Inputs: $X, n, d, MinAtt, MinObj, MaxSize, \delta_1, \delta_2, \delta_3$;

Outputs: The list of valid clusters found in *ClusterList* and *Outliers* cluster;

Steps:

01. Initialize *ClusterList*, *CandidateList* and *ExtraList* to NULL;
02. FOR $i=1$ to n DO
03. $\{x=X.getNextObject()\}$;
04. $index=ClusterList.findBestCluster(x, MinAtt, \delta_1)$;
05. IF ($index \neq NULL$) THEN
06. $ClusterList[index].merge(i, x)$;
07. ELSE
08. $\{index=CandidateList.findBestCluster(x, MinAtt, \delta_2)$;
09. IF ($index \neq NULL$) THEN
10. $\{CandidateList[index].merge(i, x)$;
11. IF ($sizeof(CandidateList[index].T) == MinObj$) THEN
12. Transfer cluster $CandidateList[index]$ to *ClusterList*;
13. $\}$
14. ELSE
15. $\{index=ExtraList.findBestCluster(x, MinAtt, \delta_3)$;
16. IF ($index \neq NULL$) THEN

```

17.         {ExtraList[index].merge(i, x);
18.         IF (sizeof(ExtraList[index].T) == MinObj) THEN
19.             Transfer cluster ExtraList[index] to ClusterList;
20.         }
21.     ELSE
22.         {C=createClusterStructure(i, x);
23.         Insert cluster C in CandidateList;
24.         IF ( sizeof(CandidateList) == MaxSize ) THEN
25.             {Transfer the oldest cluster in CandidateList to ExtraList;
26.             IF (sizeof(ExtraList) == MaxSize ) THEN
27.                 Merge the oldest cluster from ExtraList with Outliers;
28.             } //end IF
29.         } //end IF
30.     } //end IF
31. } //end IF
32. } //end FOR
33. FOR each cluster C in CandidateList DO
34.     {index=ClusterList.findBestCluster(C,  $\delta_3$ );
35.     IF (index != NULL) THEN
36.         ClusterList[index].merge(C);
37.     ELSE Outliers.merge(C);
38.     } //end FOR
39. FOR each cluster C in ExtraList DO
40.     { index=ClusterList.findBestCluster(C,  $\delta_3$ );
41.     IF (index != NULL) THEN
42.         ClusterList[index].merge(C);
43.     ELSE Outliers.merge(C);
44.     } //end FOR
45. END CatSub.

```

An object read from the dataset is inserted in an existing cluster returning the maximum *sim* value defined in *Equation 6.5*. The searching is done by the function *findBestCluster()* and the procedure *merge()* inserts the object in the best cluster found. If an object cannot be inserted in any one of the existing clusters in *ClusterList* and after loosening the similarity threshold in *CandidateList* and *ExtraList*, a new cluster structure is created as defined in *Equation 6.3* and it is inserted in *CandidateList*. When *CandidateList* becomes full due to its fixed size, the oldest cluster in it is transferred to *ExtraList* to make room for the new cluster. Instead of searching for the oldest cluster a pointer can be maintained to point to the clusters in a round robin manner and the cluster pointed to by it can be transferred. *ExtraList* is also dealt with in a similar manner.

6.4.2 Complexity analysis

The algorithm requires only one pass through the dataset to produce a set of clusters. Number of comparisons required for each object is proportional to the number of clusters(*c*) already existing in *ClusterList*, since at most *MaxSize* (a constant) comparisons are needed for each of *CandidateList* and *ExtraList* which are small in size. Therefore, the maximum time complexity becomes, $O(nc)$, where *n* is the total number of objects in the dataset. It is expected that large datasets also possess large clusters causing *c* to be smaller. For each cluster the (*Attribute, value*) pairs are stored in main memory. Size of the (*Attribute, value*) pairs depends upon the number of attributes (*d*). Therefore, space complexity is $O(cd)$.

6.5 Experimental Validation

Experimental evaluation of our algorithm is performed with some of the commonly used real-life datasets available in the UCI Machine Learning Repository [BM98]. The selected datasets have labeled objects i.e. they are already classified into some classes(clusters). We attempt to recover those clusters for measuring the accuracy of the algorithm. The results are compared with results reported by other algorithms such as *k-modes* [Hua98], *ROCK* [RGS99], *SUBCAD* [GW04] etc. for the same datasets. We implemented the algorithm in C++. As our algorithm produces a set of disjoint clusters, the clusters can be considered to be defined over the full set of attributes. So, we have not reported the subspaces which has caused the discovery of the clusters. Experiments were conducted on a 1.66 GHZ HCL laptop with 512 MB RAM running LINUX operating system.

6.5.1 Accuracy calculation of clustering results

A commonly used measure for evaluating the quality of a clustering result is the clustering accuracy(r). It is defined in [Hua98] as follows:

$$r = \frac{1}{n} \sum_{i=1}^k a_i \quad (6.6)$$

where a_i is the number of data objects occurring in both cluster i and its corresponding class, and k is the number of clusters obtained in a dataset with n objects. The clustering error e is defined as: $e = 1 - r$.

6.5.2 Data sets

The datasets used are described below :

- **The Soybean (small) disease dataset** has 47 instances, each being described by 35 attributes, all categorical. Each instance is labeled as one of four diseases: D1, D2, D3 and D4. Except for D4, which has 17 instances, all other diseases have 10 instances each.
- **The zoo dataset.** The database contains 101 animals, each of which has 15 Boolean attributes and one categorical attribute besides animal name and type. The animals are divided into seven classes.
- **The Congressional votes dataset.** This dataset contains the United States Congressional Voting Records for 1984. Each record contains a Congressman's votes on 16 issues(e.g. education spending, crime etc.). All the attributes are Boolean("yes" or "no") with a few of votes containing missing values. We treated missing values as another domain value for the attribute. A classification field with the labels "Democrat" or "Republican" is provided for each record. There are 435 records, 168 Republican(R) and 267 Democrat(D) instances.
- **Wisconsin breast cancer dataset.** The Wisconsin breast cancer dataset has 699 records, each of which is described by 10 categorical attributes. There are 16 records that have missing values. Records are labeled with two classes - 458 Benign and 241 Malignant.
- **Mushroom dataset.** It contains 8124 tuples, each representing a mushroom characterized by 22 attributes, such as color, shape, odor etc. Mushroom are classified as either poisonous (3916 tuples) or edible (4208 tuples). There are 2480 missing values.
- **KDD CUP 1999 Corrected Network Intrusion Data.** The dataset contains 311029 data records, each representing a connection between two network hosts according to some well defined network protocol and is described by 41 attributes (38 continuous or discrete numerical attributes

and 3 categorical attributes) such as duration of connection, number of bytes transferred, number of failed login attempts etc. Each record was labeled as either normal or one specific kind of attack. There are 37 different attacks present in the dataset. We have converted the numeric attributes to categorical attributes by discretization (for example, taking logarithm to the base 2 of all numeric values).

The datasets used are summarized in *Table 6.2*

Table 6.2: Datasets used in the experiments.

Datasets	Objects	Attributes	Classes
Soybean small	47	35	4
Zoo	101	16	7
Congressional Votes	435	16	2
Wisconsin breast cancer	699	10	2
Mushroom	8124	22	2
KDD CUP Corrected	311029	41	38

6.5.3 Result on soybean dataset

The *CatSub* algorithm clustered the soybean small disease dataset into four clusters C_1 , C_2 , C_3 and C_4 . The dataset also contains four original classes D1, D2, D3 and D4. The misclassification matrix of the result obtained is shown in *Table 6.3*. Parameter values used corresponding to the result are: $MinObj=4$, $MinAtt=20$, $\delta_1=1$, $\delta_2=8$, $\delta_3=10$. The algorithm is able to cluster the dataset with an accuracy of 0.99. Accuracies reported by *k-modes* [Hua98] and *SUBCAD* [GW04] for the same dataset are also shown in *Table 6.4*.

Table 6.3: Clustering result on soybean dataset.

Clusters	D1	D2	D3	D4
C_1	10	0	0	0
C_2	0	10	0	0
C_3	0	0	10	1
C_4	0	0	0	16

Table 6.4: Accuracy on soybean dataset.

Algorithm	Accuracy
<i>CatSub</i>	0.99
<i>k-modes</i>	0.95
<i>SUBCAD</i>	0.93

6.5.4 Result on zoo dataset

The zoo dataset is already classified into 7 classes. The misclassification matrix of the clustering result (parameter values used are: $MinObj=4$, $MinAtt=9$, $\delta_1=1$, $\delta_2=1$, $\delta_3=1$) obtained by applying our algorithm on the dataset is shown in *Table 6.5*. Column headings label the existing classes in the dataset and row headings label clusters obtained. The accuracy achieved is $r = \frac{36+20+5+13+0+8+8}{101} = 0.89$. Li et al. [LMO04] also reported clustering result for zoo dataset. It is reproduced in *Table 6.6* for comparison. Accuracy obtained by the algorithm is 0.79.

Table 6.5: The misclassification matrix of clustering result on zoo dataset.

Clusters	1	2	3	4	5	6	7
C_1	36	0	0	0	0	0	0
C_2	0	20	0	0	0	0	0
C_3	0	0	5	0	4	0	0
C_4	2	0	0	13	0	0	0
C_5	3	0	0	0	0	0	0
C_6	0	0	0	0	0	8	2
C_7	0	0	0	0	0	0	8

Table 6.6: The misclassification matrix on zoo dataset reported by Li et al.

Clusters	1	2	3	4	5	6	7
C_1	32	0	0	0	1	2	2
C_2	0	20	0	0	0	0	0
C_3	9	0	1	0	0	0	0
C_4	0	0	0	13	0	0	0
C_5	0	0	0	0	0	0	0
C_6	0	0	0	0	0	6	0
C_7	0	0	4	0	2	0	8

6.5.5 Result on congressional voting dataset

We treated missing values in the congressional voting dataset as separate categories in the domains of the attributes and clustered the full dataset with 435 records. Our algorithm detected the clusters shown in *Table 6.7* (parameter values: $MinObj=4$, $MinAtt=1$, $\delta_1=1$, $\delta_2=1$, $\delta_3=1$). Note that the third cluster is formed

with outliers. Accuracy obtained, excluding the outliers, is $r = \frac{156+224}{419} = 0.91$. Some algorithms like *ROCK* [RGS99] clustered 372 records of the dataset after eliminating records with missing values. Clustering result reported for *ROCK* is reproduced in *Table 6.8*. Accuracy of the result obtained is 0.93.

Table 6.7: Clustering result on Congressional Voting dataset.

Cluster	Republican	Democrat
C_1	156	37
C_2	2	224
C_3	10	6

Table 6.8: Clustering result of *ROCK* on Congressional Voting dataset.

Cluster	Republican	Democrat
C_1	144	22
C_3	5	201

6.5.6 Result on Wisconsin breast cancer dataset

Clustering result obtained for the breast cancer dataset is presented in *Table 6.9*. Parameter values used are: $MinObj=4$, $MinAtt=2$, $\delta_1=1$, $\delta_2=1$, $\delta_3=1$. The third cluster consists of outliers. Note that most of the records in this cluster belong to the Malignant class. The objects in Malignant class differ widely from one another, therefore the algorithm collected them into the *Outliers* cluster. In contrast the objects from the Benign class are separated into the first two clusters of somewhat similar objects. Considering the third cluster as a valid one the

accuracy of the result obtained is 0.91. Gan and Wu [GW04] applied *SUBCAD* to cluster 683 records of the dataset after eliminating 16 records with missing values. They obtained an accuracy of 0.87 with the result shown in *Table 6.10*.

Table 6.9: Clustering result on breast cancer dataset.

Cluster	Benign	Malignant
C_1	359	7
C_2	57	12
C_3	42	222

Table 6.10: Clustering result on breast cancer dataset reported by *SUBCAD*.

Cluster	Benign	Malignant
C_1	4	158
C_3	440	81

6.5.7 Result on mushroom dataset

Our algorithm extracted 20 clusters (parameter values used: $MinObj=8$, $MinAtt=8$, $\delta_1=1$, $\delta_2=5$, $\delta_3=8$) from the Mushroom dataset. The result is shown in *Table 6.11*. All of the clusters are pure. The *ROCK* algorithm had reported 21 clusters shown in *Table 6.12*. *ROCK* had found one impure cluster with 32 edible and 72 poisonous mushrooms.

Table 6.11: Clustering result on Mushroom dataset.

Clusters	1	2	3	4	5	6	7	8	9	10
<i>Edible</i>	192	0	288	0	32	16	0	0	0	192
<i>Poisonous</i>	0	1728	0	36	0	0	8	288	1296	0
Clusters	11	12	13	14	15	16	17	18	19	20
<i>Edible</i>	0	48	48	0	1728	0	0	768	0	896
<i>Poisonous</i>	72	0	0	32	0	192	8	0	256	0

Table 6.12: Clustering result by ROCK on Mushroom dataset.

Clusters	1	2	3	4	5	6	7	8	9	10	11
<i>Edible</i>	96	0	704	96	768	0	1728	0	0	0	48
<i>Poisonous</i>	0	256	0	0	0	192	0	32	1296	8	0
Clusters	12	13	14	15	16	17	18	19	20	21	
<i>Edible</i>	48	0	192	32	0	288	0	192	16	0	
<i>Poisonous</i>	0	288	0	72	1728	0	8	0	0	36	

6.5.8 Scalability test

Execution time needed for each of the experiments are shown in *Table 6.13*. To ascertain the nature of scalability, execution times needed by *CatSub* algorithm to cluster first 50000, 100000, 150000, 200000, 250000, and 300000 records of the KDD CUP Corrected dataset are evaluated and plotted in *Figure 6.1*. The graph shows that the execution time tend to increase almost linearly. Most of the clusters obtained in each case were pure clusters containing either attack or normal records. We also computed the accuracy in retrieving attack or normal records and found that the accuracy is more than 0.94 in each case.

Table 6.13: Execution time for the datasets.

Datasets	Objects	Attributes	Classes	Accuracy	Time(s)
Soybean small	47	35	4	1.00	0.00
Zoo	101	16	7	0.90	0.00
Congressional Votes	435	16	2	0.89	0.0
Wisconsin breast cancer	699	10	2	0.92	0.01
Mushroom	8124	22	2	0.99	0.22

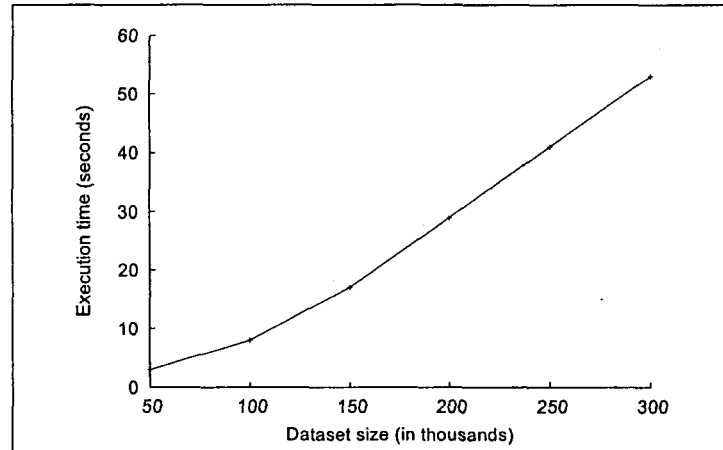


Figure 6.1: Scalability of CatSub to the no. of records when clustering KDD CUP Corrected dataset.

6.5.9 Discussion on parameters

Although five parameters are used in the algorithm they are not difficult to determine. The parameter *MinObj* specifies the minimum number of objects for recognizing a cluster. Using a small value such as 4 or 8 is recommended if no domain knowledge is available to fix the value of the parameter. *MinAtt* specifies the minimum number of attributes over which all the objects in a cluster

should agree. It can have a value as small as 1. Most frequently the parameter δ_1 should be set to 1. A value of zero can also be used to obtain tight clusters with larger number of matching attributes. The parameter δ_2 as well as δ_3 should have a minimum value of 1. Bigger values can be used if the dataset contains large number of attributes. In that case δ_2 should be less than 30% and δ_3 should be less than 50% of total number of attributes. For low dimensional data the value for each of δ_1 , δ_2 and δ_3 can be set to 1.

Incremental algorithms generally become order dependent. The order dependency of our algorithm becomes minor due to use of the three parameters δ_1 , δ_2 and δ_3 . The experiments mentioned above were repeated several times, each time the records are shuffled randomly. The results obtained did not differ much, which means that order dependency of the algorithm is minor.

All the results presented suggest that the proposed algorithm can produce good quality results for small or large categorical datasets.

Part IV

Clustering Mixed-type Data

There are not many algorithms available for clustering datasets containing mixture of numeric and categorical attributes. In *Chapter 7* we present an efficient new algorithm developed by us for clustering large high dimensional mixed-type datasets. The algorithm works based on entropy calculations of clusters using different methods for numeric and categorical attributes.

Chapter 7

SMIC: A Subspace Preferred Mixed Type Data Clustering Technique

7.1 Introduction

Very often, real world databases contain both numeric and categorical attributes, requiring specialized algorithms to cluster such data. Some strategies [And73] for dealing with such problems are :

1. *Partitioning of attributes*: Two parallel but separate analyzes can be performed, one based on numeric attributes and the other based on categorical attributes. Relative weighing of the attribute types and the joint or interactive effects between attributes would be of importance. A systematic and meaningful method of integrating such separate analysis is required.

2. *Conversion of attributes*: Attributes can be converted from one type to another. The choice of how to homogenize the set of attributes should be influenced strongly by which attribute type is the most numerous.
3. *Disagreement indices*: It is necessary to equalize the attributes in some appropriate sense. The attributes can be equalized by attribute-by-attribute disagreement between data objects. When two data objects have identical responses on an attribute there is zero difference or disagreement between them. Within a finite dataset there is a maximum level of observed disagreement on any attribute. If the maximum disagreement is scored as one, then all disagreement on an attribute may be represented by a disagreement index ranging from zero to one. Disagreement on attributes of every type may be expressed in this manner.

Using the third approach we present an efficient algorithm for clustering large high dimensional datasets containing mixture of categorical and numeric attributes. Disagreement between data objects for each attribute is measured by entropy computation. The algorithm can be used for clustering categorical datasets as well. Although the algorithm can be used for clustering datasets with numeric attributes alone it is not recommended, since nature of a numeric dataset is very much different than nature of a categorical or mixed-type dataset and our algorithm is specifically designed for mixed-type datasets. Important features of the algorithm are :

- It provides a solution for the mixed-type attribute clustering problem.
- It produces good quality results efficiently.
- Use of subspace based similarity measure makes the algorithm suitable for clustering high dimensional datasets.

- The algorithm is scalable as it uses an incremental algorithm to group similar objects together using only a single pass over the dataset. Then clusters are merged hierarchically to produce desired number of clusters.
- Outliers can be handled efficiently.

7.2 Related Works

The proposed algorithm can cluster datasets with mixture of categorical and numeric attributes as well as datasets with categorical attributes alone. The number of works available on mixed-type data is small. Some of them are mentioned below. Related works on some categorical clustering algorithms were presented in chapter 6.

Huang [Hua98] extended the *k-means* algorithm to the *k-modes* algorithm to tackle the problem of clustering large categorical datasets in data mining. Further, Huang also combined the *k-modes* algorithm with the *k-means* algorithm resulting in the so-called *k-prototypes* algorithm for clustering objects described by mixed numeric and categorical attributes. However, *k-prototypes* also produce locally optimal results like *k-means*.

A clustering algorithm for mixed-type data was proposed by Le [LH03]. The algorithm chooses k number of largest sets from non-expandable strongly connected sets, which had been built by using breadth first search algorithm. The remaining objects are assigned to some clusters by testing the minimum distance of the object with all clusters. Problem with this algorithm is that it may produce less than k number of clusters initially. Also, some clusters are reprocessed.

In [YTRC05], cluster ensemble approach based on divide-and-conquer technique is presented for clustering mixed type datasets. First, the original mixed dataset is divided into two sub-datasets : the pure categorical dataset and pure numeric dataset. Next, existing well established clustering algorithms designed

for different types of datasets are employed to produce corresponding clusters. Last, the clustering results on the categorical and numeric datasets are combined as a categorical dataset, on which the categorical data clustering algorithm is used to get the final clusters.

An algorithm for clustering mixed type data is presented in [HXd]. It uses a CF*-tree to pre-cluster datasets. Then the dense regions stored in the leaf nodes are treated as single points and *k-prototype* algorithm is used to cluster such points.

7.3 Problem Formulation

For each individual attribute of the dataset, entropy is calculated and normalized to the range $[0, 1]$. Different methods are used for calculating entropy for numeric and categorical attributes. Computing a dissimilarity measure of individual attributes based on normalized entropy values causes homogenization of the attributes. Then a subspace-based similarity measure is defined for an individual cluster. To achieve scalability in clustering large high dimensional datasets, an incremental method of clustering is to be used avoiding storage of the data objects in the main memory. Therefore, a cluster summary measure is defined based upon which the similarity of the cluster obtained by merging an object or another cluster to an existing cluster can be computed easily. A cluster structure is the ultimate data structure to be stored in the main memory.

The given set of n objects $X = \{X_1, X_2, \dots, X_n\}$ is described by d attributes A_1, A_2, \dots, A_d that may be either numeric or categorical. For each i ($1 \leq i \leq n$) and for each j ($1 \leq j \leq d$), x_{ij} represent the j -th component of object X_i and x_{ij} take on one of the possible values defined in domain D_j of attribute A_j . An attribute A_j is categorical if its domain D_j is discrete valued, ordered or unordered while A_j is numeric if its domain is continuous valued and ordered. An object X_i

will be represented by its object-id i and an attribute A_j will be represented by its attribute-id j so that a set of objects $T \subseteq \{1, 2, 3, \dots, n\}$ together with a set of attributes represent a subspace based cluster C . Each attribute A_j of a cluster C can be treated as a random variable taking a number of possible values defined in its domain D_j . So, entropy of each attribute and hence entropy of a cluster can be computed. The dataset will be partitioned into a set of disjoint clusters and a set of outliers.

7.3.1 Entropy

Suppose that a probabilistic experiment involves the observation of a discrete random variable Y . Let, $D_Y = \{y_1, y_2, \dots, y_T\}$ is the set of T possible values that Y can take on and probability of $Y = y_i$ is p_{y_i} , $1 \leq i \leq T$ so that

$$p_{y_1} + p_{y_2} + \dots + p_{y_T} = 1. \quad (7.1)$$

It is assumed that all p_{y_i} are strictly greater than zero. Then entropy, H_Y of the random variable Y , is to be interpreted as the average uncertainty associated with the events ($Y = y_i$). It is defined in [Ash90] as:

$$H_Y = - \sum_{i=1}^T p_{y_i} \log_2(p_{y_i}). \quad (7.2)$$

H_Y is a bounded variable. Its lower value is 0 and upper value is $\log_2(T)$. The joint entropy of m independent random variables Y_1, Y_2, \dots, Y_m is obtained as

$$H_{(Y_1, Y_2, \dots, Y_m)} = H_{Y_1} + H_{Y_2} + \dots + H_{Y_m}. \quad (7.3)$$

7.3.2 Entropy of a categorical attribute

Let the domain of a categorical attribute A_j , $1 \leq j \leq d$ be represented as :

$$D_j = \{v_{j1}, v_{j2}, \dots, v_{jd_j}\} \quad (7.4)$$

where, v_{jk} denotes the k -th category and d_j is the total number of categories in D_j i.e. $d_j = |D_j|$. Let, $t_{jk}(C)$ denote the frequency of occurrence of attribute value v_{jk} , $1 \leq k \leq d_j$ in a given cluster C consisting of set of objects T , so that

$$t_{jk}(C) = |\{i \mid i \in T, x_{ij} = v_{jk}\}| \quad (7.5)$$

Frequency of occurrence t_{jk} divided by total number of objects in the cluster gives the probability p_{jk} of category v_{jk} , that is

$$p_{jk}(C) = \frac{t_{jk}(C)}{n_C} \quad (7.6)$$

where $n_C = |T|$ denotes the total number of objects in cluster C . So, entropy of attribute A_j for the cluster C , denoted by $H_j(C)$ is computed as

$$H_j(C) = - \sum_{k=1}^{d_j} p_{jk}(C) \log_2(p_{jk}(C)). \quad (7.7)$$

It is assumed here that $0 = \log_2(0)$. The maximum value that $H_j(C)$ can attain is $\log_2(d_j)$. This value is achieved when all d_j categories of the attribute A_j are present in the cluster with equal frequencies. This entropy is to be minimized for better clusters.

7.3.3 Entropy for a numerical attribute

A numeric attribute takes continuous values. Therefore, method of calculating entropy is somewhat different than for categorical attributes. Given a cluster C with n_C objects, a numeric attribute A_j can be thought of as a random variable with n_C possible values. Hence, entropy can be computed with n_C probabilities as shown below. We assume that the dataset is preprocessed such that values taken by all numeric attributes are positive (nonzero) real numbers. Zero values, if present, can be replaced by a very small positive quantity less than all valid

values in the dataset. Let,

$$sum_j(C) = \sum_{k \in T} x_{kj} \quad (7.8)$$

represent the sum of the j -th attribute values of the set of objects (T) in the cluster. Probability of each value x_{kj} , $k \in T$ taken by j -th attribute becomes $\frac{x_{kj}}{sum_j(C)}$, and hence entropy $H_j(C)$ is computed as :

$$H_j(C) = - \sum_{k \in C} \frac{x_{kj}}{sum_j(C)} \log_2 \left(\frac{x_{kj}}{sum_j(C)} \right). \quad (7.9)$$

The maximum value that $H_j(C)$ can take is $\log_2(n_C)$. This value is attained when all of the n_C values of the numeric attribute are the same. Which means that entropy is more when the data values are uniform and less when data values are more random. So, for better clusters entropy for numeric attributes need to be maximized. This is opposite to the entropy for categorical attributes where the entropy is to be minimized.

7.3.4 Dissimilarity measure of a cluster

Attributes are homogenized by computing dissimilarity measures for individual attributes based on normalized entropy values. Let, $G_j(C)$ indicates dissimilarity of j -th attribute in the cluster C . It is computed as,

$$G_j(C) = \begin{cases} \frac{H_j(C)}{\log_2(d_j)} & \text{if } A_j \text{ is categorical} \\ 1 - \frac{H_j(C)}{\log_2(n_C)} & \text{if } A_j \text{ is numeric} \end{cases} \quad (7.10)$$

The entropy computed for an attribute is divided by maximum possible entropy so that entropy of each attribute is normalized in the range $[0,1]$. Higher entropy of a categorical attribute indicates that the objects are more dissimilar, whereas higher entropy for numeric attributes indicate similar objects. So, to obtain a dissimilarity measure entropy of numeric attributes are subtracted from one.

Dissimilarity measure for a cluster C , $G(C)$ is obtained by summing the dissimilarity measures for each attribute. That is

$$G(C) = \sum_{j=1}^d G_j(C). \quad (7.11)$$

Dissimilarity measures need to be minimized for better clustering.

7.3.5 Subspace based similarity measure

We define the similarity measure, $S(C)$ of a cluster to be the count of attributes for which the dissimilarity measures are nearly equal to zero :

$$S(C) = |\{j \mid G_j(C) \leq \epsilon, j \in \{1, 2, \dots, d\}\}| \quad (7.12)$$

where, ϵ is a very small quantity (for example 0.0001).

7.3.6 Summary measures

A cluster summary measure,

$$F(C) = \{F_1(C), F_2(C), \dots, F_d(C)\} \quad (7.13)$$

consists of attribute summary measures, $F_j(C)$ for each of the d attributes. Summary measure of a new cluster, obtained by merging two existing clusters, can be computed easily from the summary measures of the existing clusters.

Summary measure for a categorical attribute

Let, $F_j(C)$ represents the summary measure of a categorical attribute A_j for a given cluster C . The attribute value frequencies defined in *Equation 7.5* provide the required summary measure. That is,

$$F_j(C) = \{t_{j1}(C), t_{j2}(C), \dots, t_{jd_j}(C)\}. \quad (7.14)$$

Summary measure $F_j(C_p \cup C_q)$ for the merger of two clusters C_p and C_q is :

$$F_j(C_p \cup C_q) = \{t_{j1}(C_p) + t_{j1}(C_q), t_{j2}(C_p) + t_{j2}(C_q), \dots, t_{jd_j}(C_p) + t_{jd_j}(C_q)\} \quad (7.15)$$

Summary measure for a numeric attribute

Summary measure for a numeric attribute A_j in cluster C contains only two entries which are nothing but the sum and entropy of the attribute values defined by *Equations 7.8* and *7.9* respectively.

$$F_j(C) = \{sum_j(C), H_j(C)\}. \quad (7.16)$$

Given the summary measures for two clusters C_p and C_q , summary measure $F_j(C_p \cup C_q)$ is calculated as shown below.

$$sum_j(C_p \cup C_q) = sum_j(C_p) + sum_j(C_q) \quad (7.17)$$

$$H_j(C_p \cup C_q) = Z_1\{H_j(C_p) - \log_2(Z_1)\} + Z_2\{H_j(C_q) - \log_2(Z_2)\}. \quad (7.18)$$

where, $Z_1 = \frac{sum_j(C_p)}{sum_j(C_p \cup C_q)}$ and $Z_2 = \frac{sum_j(C_q)}{sum_j(C_p \cup C_q)}$. Thus,

$$F_j(C_p \cup C_q) = \{sum_j(C_p \cup C_q), H_j(C_p \cup C_q)\} \quad (7.19)$$

7.3.7 Cluster structure

During the clustering process the cluster summary measure need to be maintained along with the list of objects, $T \subseteq \{1, 2, \dots, n\}$. Total number of objects in the cluster (n_C) and similarity measure (S) can also be stored for computational efficiency. So creation of a new cluster C means the creation of the following structure :

$$C = \{n_C, S, T, F\} \quad (7.20)$$

where S and F are given by *Equations 7.12* and *7.13* respectively.

Example: A sample dataset shown in *Table 7.1* consisting of 8 records defined over five categorical and two numeric attributes. Domains of the categorical attributes are : $D_1 = \{b, a\}$, $D_4 = \{u, y, l, t\}$, $D_5 = \{g, p, gg\}$, $D_6 = \{t, f\}$ and $D_7 = \{g, p, s\}$. Domains of numeric attributes are positive real numbers. Consider a cluster consisting of three records ($n_C = 3$) and $T = \{1, 4, 6\}$. The

Table 7.1: A sample dataset.

A_1	A_2	A_3	A_4	A_5	A_6	A_7
b	19.40	0.75	u	g	t	s
b	21.17	0.25	y	p	f	g
b	17.50	22.00	l	gg	t	p
b	19.17	0.01	y	p	t	s
b	21.25	1.50	u	g	f	g
a	18.78	0.38	l	gg	t	s
b	33.67	1.25	u	g	f	g
b	26.75	4.50	y	p	f	g

cluster structure for this cluster is shown in *Table 7.2*. Summary measures F_1 , F_4 , F_5 , F_6 and F_7 for categorical attributes consists of two, four, three, two and three entries respectively as their domain sizes contain corresponding number of elements. The entries represent frequency of occurrence for each category in the domain. For example, the first entry(3) in F_1 indicates that the first category(b) of domain D_1 occurs three times, while the second entry(0) indicates no occurrence for the second category(a) of the domain. F_2 is a summary measure for numeric attribute. Its first entry stores the sum of the attribute values, $57.35 = 19.40 + 19.17 + 18.78$ and the second entry, 0.99992 represents the corresponding entropy measures for the attribute. Dissimilarity measures for the attributes are computed

to be $G_1 = 0.9182$, $G_2 = 0.00002$, $G_3 = 0.3781$, $G_4 = 0.4591$, $G_5 = 1.0$, $G_6 = 0.0$ and $G_7 = 0.0$. Out of these values G_2 , G_6 and G_7 are less than 0.0001. So, similarity measure for the cluster $S = 3$.

Table 7.2: Summary measure

tobj	S	T	F_1	F_2	F_3	F_4	F_5	F_6	F_7
3	3	1	3	57.35	1.14	2	1	3	0
		4	0	0.99992	0.6219	0	1	0	0
		6				1	1		3
						0	0		

7.4 Proposed Algorithm

The proposed algorithm can cluster large, high dimensional datasets consisting of a mixture of categorical and numeric attributes. For each attribute, the attribute type (*AttType*) and domain size (*Asize*) should be provided as input. Attribute type provides the information for selecting the appropriate method for computing entropy for an attribute. Memory space is reserved for storing a summary measure based on the domain size. Summary measure of a numeric attribute has only two entries. Therefore domain size entered for all numeric attributes should be 2. Two phases of clustering are used. In the first phase, an incremental algorithm places objects, read sequentially from the hard disk, into existing clusters (initially none) based upon subspace similarity. Then, a hierarchical algorithm in the second phase reduces number of clusters by hierarchically merging them until required number of clusters are produced. Outliers handling is done in both the phases. The same incremental algorithm presented in *CatSub* (refer *Chapter 6*) for clustering categorical data is also used here with new similarity measure, summary measure

and cluster structure.

7.4.1 Incremental clustering

The purpose of this step is to form initial clusters with objects which are highly similar over a subset of attributes. Subspace based similarity measure of a cluster, S , introduced in *Equation 7.12* provides the number of attributes which have dissimilarity measure nearly equal to zero. A cluster should contain at least $MinAtt$ number of such attributes, i.e. $S \geq MinAtt$ for any cluster, where $MinAtt$ is an input parameter. To achieve scalability each data object read sequentially from the hard disk is inserted on the fly in an existing cluster or a new cluster is created with the object. Inserting a new object in an existing cluster may decrease its S value, but this decrease should be less than a given threshold, δ , otherwise the object should not be inserted in the cluster. Minimum possible value for δ is zero and maximum possible value is total number of attributes(d) minus $MinAtt$. To determine the cluster, C , where an object x can be inserted, we define the following similarity function :

$$sim(C, x) = \begin{cases} S(C') & \text{if } S(C) - S(C') \leq \delta \text{ and } S(C') \geq MinAtt \\ 0 & \text{otherwise} \end{cases} \quad (7.21)$$

where, C' indicates the cluster obtained if object x is merged with C . The object will be inserted in the cluster returning the maximum nonzero sim value. A sequential search procedure is used to find the best cluster. Search space increases if outlier objects are allowed to create small clusters. To handle outliers three different lists of clusters - *ClusterList*, *CandidateList* and *ExtraList* are used. Initially all the lists are empty. If an object is not inserted in any of the clusters present in any of the three lists a new cluster is created with the object and the cluster is placed in the *CandidateList*. Maximum possible size for *CandidateList* and *ExtraList* are fixed at $maxSize$. When the *Candidate*

list becomes full, the oldest cluster in it is transferred to *ExtraList* to make room for the new cluster. If the *ExtraList* also becomes full with transferred clusters, the oldest cluster in it is removed and merged with *Outliers* cluster. Whenever an object gets inserted in a cluster, included in either *CandidateList* or *ExtraList*, the number of objects present in the cluster should be examined. If it collects *MinObj* (say, 4) objects then the cluster is transferred to *ClusterList*, which is the list of valid clusters. *ClusterList* can grow to any size. The threshold δ defined in Equation 7.21 takes three different forms - δ_1 , δ_2 and δ_3 for inserting an object in a cluster present in *ClusterList*, *CandidateList* and *ExtraList* respectively. The three thresholds take low, medium and high values in the range $[0, d]$, where d is the number of attributes. An object read from the hard disk is first tried for insertion in a cluster in *ClusterList* with a very low value of threshold δ_1 . If it is not inserted, then *CandidateList* is tried with threshold δ_2 assigned a medium value (say, less than 30% of d with minimum value of 1). If not inserted again, *ExtraList* is tried with a very loose threshold δ_3 allowing for much higher decrease in S value. Maximum possible value is $\delta_3=d - MinAtt$. If the object could not be inserted in a cluster in *ExtraList* also, a new cluster structure is created and inserted in *CandidateList*.

7.4.2 Hierarchical clustering

The first step may produce a large number of clusters. One may expect a reduced number of clusters or the number of clusters required (*reqd*) may be specified. Therefore, a bottom up hierarchical clustering technique is used to iteratively reduce number of clusters by merging the two least dissimilar clusters at a time until required number of clusters remain or maximum allowed dissimilarity measure (*Gmax*) of a cluster is crossed. Dissimilarity between a pair of clusters is measured using $G(C)$, presented in Equation (7.11) as shown below.

$$dissimilarity(C_p, C_q) = G(C_p \cup C_q) \quad (7.22)$$

Final set of clusters are obtained when the hierarchical algorithm gets terminated. At this point an attempt is made for merging each of the sub clusters that remain in *CandidateList* and *ExtraList* with any one of the final clusters if maximum allowed dissimilarity threshold(*Gmax*) permits the merger, otherwise they are merged with *Outliers* cluster.

The algorithm is presented below.

Algorithm SMIC

Inputs: $X, n, d, MinAtt, MinObj, MaxSize, \delta_1, \delta_2, \delta_3, \{AttType_i, i = 1, 2, \dots, d\}, \{A_{size}_i, i = 1, 2, \dots, d\}$;

Outputs: The list of valid clusters found in *ClusterList* and *Outliers* cluster;

Steps:

01. Set $MaxSize=100, Gmax=0.5*d$;
02. Initialize *ClusterList*, *CandidateList* and *ExtraList* to NULL;
03. FOR $i=1$ to n DO
04. $\{x = X.getNextObject();$
05. *ClusterList.findBestCluster*($x, MinAtt, \delta_1, index, maxSim$);
06. IF ($index \neq NULL$) THEN
07. *ClusterList[index].merge*($x, i, maxSim$);
08. ELSE
09. $\{CandidateList.findBestCluster(x, MinAtt, \delta_2, index, maxSim);$
10. IF ($index \neq NULL$) THEN
11. $\{CandidateList[index].merge(x, i, maxSim);$
12. IF ($CandidateList[index].nC == MinObj$) THEN
13. Transfer cluster *CandidateList[index]* to *ClusterList*;
14. $\}$
15. ELSE

```

16.      {ExtraList.findBestCluster(x, MinAtt, δ3, index, maxSim)};
17.      IF (index != NULL) THEN
18.          {ExtraList[index].merge(x, i, maxSim)};
19.          IF (ExtraList[index].nC == MinObj) THEN
20.              Transfer cluster ExtraList[index] to ClusterList;
21.          }
22.      ELSE
23.          {C = createClusterStructure(x, i)};
24.          Insert cluster C in CandidateList;
25.          IF (sizeof(CandidateList) == MaxSize) THEN
26.              {transfer the oldest cluster in CandidateList to ExtraList;
27.              IF (sizeof(ExtraList) == MaxSize) THEN
28.                  Delete the oldest cluster in ExtraList and merge it to
Outliers;
29.              }
30.          }
31.      }
32.  }
33. } //end FOR

```

// Hierarchical Clustering

```

34. NoOfClusters = sizeof(ClusterList);
35. WHILE (NoOfClusters > reqd) DO
36.     {ClusterList.findMergePair(index1, index2, minDissimilarity)};
37.     IF (minDissimilarity >= Gmax) BREAK;
38.     ClusterList[index1].merge(ClusterList[index2]);
39.     NoOfClusters = NoOfClusters - 1;
40. }

```



```

41. FOR each cluster  $C$  in CandidateList DO
42.   {ClusterList.findMinDissimilar( $C$ , index, minDissimilarity);
43.   IF (minDissimilarity  $\leq$   $G_{max}$  ) THEN
44.     ClusterList[index].merge( $C$ );
45.   ELSE Outliers.merge( $C$ );
46.   }

47. FOR each cluster  $C$  in ExtraList DO
48.   {ClusterList.findMinDissimilar( $C$ , index, minDissimilarity);
49.   IF (MinDissimilarity  $\leq$   $G_{max}$  ) THEN
50.     ClusterList[index].merge( $C$ );
51.   ELSE Outliers.merge( $C$ );
52.   }
53. FOR each cluster  $C$  present in ClusterList DO
54.   OUTPUT C.T;
55. END SMIC.

```

Details of the functions used can be derived from the description presented in *Section 7.3*. The function *findBestCluster*(x , *MinAtt*, δ , *index*, *maxSim*) returns the *index* of the best cluster where an object x should be inserted. It also returns *maxSim*, the similarity value computed using *Equation 7.21* when the object gets inserted in the best cluster. The computation is based upon first computing subspace based similarity measure S as described in *Equation 7.12*. Actual insertion of the object into the cluster is done by the function *merge*(x , i , *maxSim*) which takes as input the object x , its serial number i and the computed *maxSim* so that the cluster summary measure can be updated.

7.4.3 Complexity analysis

The algorithm requires only one pass through the dataset to produce a set of initial clusters. Number of comparisons required for each object depends upon the number of initial clusters (c) created. It is expected that large datasets also possess large clusters causing c to be smaller, as outliers are removed during clustering. Efficient implementation of the hierarchical clustering phase, as suggested in [And73], makes its complexity to be $O(c^2)$. Therefore, the overall complexity becomes, $O(nc + c^2)$, where n is the total number of objects in the dataset. For each cluster the cluster structure is to be stored in main memory. Size of the cluster structure depends upon the number of attributes (d). Therefore, space complexity is $O(cd)$.

7.5 Experimental Validation

We perform experimental evaluation of the *SMIC* algorithm using some datasets available in the UCI Machine Learning Repository [BM98]. The selected datasets have labeled objects i.e. they are already classified into some classes(clusters). Accuracy of our algorithm is calculated with respect to those known clusters. Besides mixed-type datasets categorical datasets are also used for evaluating the algorithm. Experiments are conducted by implementing the algorithm in C++ on a 1.66 GHz HCL laptop with 512 MB RAM running LINUX operating system.

7.5.1 Accuracy calculation of clustering result

We use the clustering accuracy measure, r to evaluate the quality of the clustering algorithm. The clustering accuracy(exactness) measure is defined in [Hua98] as

follows:

$$r = \frac{1}{n} \sum_{i=1}^k a_i \quad (7.23)$$

where a_i is the number of data objects occurring in both cluster i and its corresponding class, k is the number of clusters and n is the number of objects in the dataset. Further, the clustering error e is defined as: $e = 1 - r$.

7.5.2 Data sets

The real life datasets used to evaluate performance of our algorithm on clustering datasets with mixed categorical and numeric attributes are described below. The categorical datasets used in *Chapter 6* for testing *CatSub* algorithm are also used here to test the performance of the algorithm on clustering datasets with categorical attributes alone. Accuracy obtained by *CatSub* for each dataset is also reported along with accuracies obtained by *SMIC* so that the two algorithms can be compared.

- **The credit approval dataset** contains mixed data. It has 690 instances each described by six numeric and nine categorical attributes. The instances are classified into two classes, approved(A) labeled as + and rejected(R) labeled as -. There are 37 instances having missing values on seven attributes. The dataset contains 307 approved instances and 383 rejected instances. We have removed 24 instances having missing values in numeric attributes as we have not used any method to deal with missing values in numeric attributes.
- **KDD CUP 1999 Corrected Network Intrusion Data.** The dataset [UoC99] contains 311029 data records, each representing a connection between two network hosts according to some well defined

network protocol and is described by 41 attributes (38 continuous or discrete numerical attributes and 3 categorical attributes) such as duration of connection, number of bytes transferred, number of failed login attempts etc. Each record was labeled as either normal or one specific kind of attack. There are 37 different attacks present in the dataset. Total number of normal records is 60593, rest are attacks.

7.5.3 Result on credit approval dataset

The credit approval dataset contains both categorical and numeric attributes. Clustering result and accuracy obtained for this dataset are shown in *Tables 7.3* and *7.4*. The result is obtained with parameter values: $MinObj=4$, $MinAtt = 4$, $\delta_1=1$, $\delta_2=4$, $\delta_3=5$, $reqd=2$. Accuracies reported by two other algorithms *k-sets* [LH03] and *k-prototypes* [Hua98] are also included for comparison.

Table 7.3: Clustering result on Credit approval dataset.

Cluster	Approved	Rejected
C_1	277	92
C_2	22	275

Table 7.4: Accuracy on credit approval dataset.

Algorithm	Accuracy
<i>SMIC</i>	0.83
<i>k-sets</i>	0.83
<i>k-prototypes</i>	0.81

7.5.4 Result on KDD CUP Corrected dataset

The algorithm is used to extract 40 clusters using the parameter values: $MinObj=20$, $MinAtt=35$, $\delta_1=1$, $\delta_2=2$, $\delta_3=5$, $reqd=39$). The result is shown in *Table 7.5*. Our aim is to extract pure clusters that contain either attack records or normal records. It can be seen from the table that most of the clusters are pure clusters. Some attack records in the dataset are so similar to normal records that it is very hard to separate them. The 40-th cluster consists of outliers. We have set $MinAtt$ value to be 35 which means that objects in a cluster are similar over as many as 35 attributes out of 41 attributes. As most of the normal records are not similar over so many attributes they are separated into the *Outliers* cluster. Some attack records also remains merged with the *outliers* cluster. To separate them the *outliers* cluster may be clustered again with a lower $MinAtt$ value. The clustering accuracy becomes $r=0.94$.

7.5.5 Result on soybean dataset

We used the *SMIC* algorithm to find four clusters in the soybean small disease dataset. The result obtained is shown in *Table 7.6*, where C_i s are the cluster names produced by our algorithm, and $D1$, $D2$, $D3$, and $D4$ are the names of the original classes. There is one to one correspondence between the disease classes and the clusters obtained, which means that the *SMIC* algorithm is able to cluster the dataset with 100% accuracy. Accuracies reported by some other algorithms, k -sets [LH03] and k -modes [Hua98] and SUBCAD [GW04] for the same dataset are also shown in *Table 7.7*. Parameter values used to run the program are: $MinObj=4$, $MinAtt=18$, $\delta_1=1$, $\delta_2=5$, $\delta_3=10$, $reqd=4$.

7.5.6 Result on zoo dataset

The misclassification matrix of the clustering result obtained with $MinObj=3$, $MinAtt=10$, $\delta_1=1$, $\delta_2=3$, $\delta_3=4$, $reqd=7$ is shown in *Table 7.8*. The column headings label the existing clusters in the dataset. A row-heading, C_i indicates a cluster obtained by our algorithm with row sum giving the total number of objects present in the cluster. The accuracy achieved is $r = \frac{37+20+0+13+4+8+9}{101} = 0.90$. Accuracy figure obtained by *CatSub* was 0.89. Li et al. [LMO04] reported an accuracy of 0.82 for their method.

7.5.7 Result on congressional voting dataset

We have used $reqd=2$ in order to extract the two known clusters in the congressional voting dataset. The result obtained by clustering all the 435 records of the dataset is presented in *Table 7.9*. Other parameter values used are: $MinObj=3$, $MinAtt=3$, $\delta_1=1$, $\delta_2=5$, $\delta_3=8$. The third cluster consists of outliers. Accuracy obtained is $r = \frac{155+218}{425} = 0.88$.

7.5.8 Result on Wisconsin breast cancer dataset

Result obtained for the breast cancer dataset is presented in *Table 7.10*. Corresponding parameter values used are: $MinObj=8$, $MinAtt=4$, $\delta_1=1$, $\delta_2=2$, $\delta_3=2$, $reqd=2$. Similar to the result obtained for *CatSub* (refer *Chapter 6*) the third cluster consist of outliers. We have considered it as a valid cluster because it signifies that the objects in Malignant class differ widely from one another, which separates them from the Benign class with somewhat similar objects. The dataset is clustered with an accuracy of 0.92 which is better than reported by other algorithms as shown in *Table 7.11*.

7.5.9 Result on mushroom dataset

The dataset is clustered in order to obtain minimum number of clusters such that all clusters are pure. The result (parameter values used: $MinObj=8$, $MinAtt=14$, $\delta_1=1$, $\delta_2=4$, $\delta_3=8$, $reqd=19$) with 19 clusters is shown in *Table 7.12* that includes one impure cluster containing 32 edible and 72 poisonous mushrooms. The *ROCK* algorithm had reported 21 clusters shown in *Table 7.13* with the same impure cluster. All pure clusters are produced by *SMIC* algorithm with 27 clusters (the result is not shown here).

7.5.10 Scalability test

Execution times of *SMIC* for each of the datasets used in the experiments are shown in *Table 7.14*. To ascertain the nature of scalability, execution times needed to extract 20 cluster in first 50000, 100000, 150000, 200000, 250000, 300000 and 311029 records of the KDD CUP Corrected dataset are evaluated and plotted in *Figure 7.1*. The graph shows that the execution time tend to increase almost linearly. The scalability of *SMIC* can be compared with that of *CatSub* presented in *Figure 6.1*. It is seen that *CatSub* is faster than *SMIC*. It is due to the fact that *SMIC* performs more computation in order to calculate the similarity measure and also it includes a hierarchical clustering phase.

7.5.11 Order dependency and parameter sensitivity

Incremental algorithms generally become order dependent. To test the order dependency of the algorithm we have repeated the above mentioned experiments several times, each time the records are shuffled randomly. Small difference in the results are noticed. It indicates minor order dependency for the algorithm.

The main input parameters are $MinAtt$ and δ_2 . $MinAtt$ has a wide range of input values. For example, $MinAtt$ has the possible range [2, 22] for the

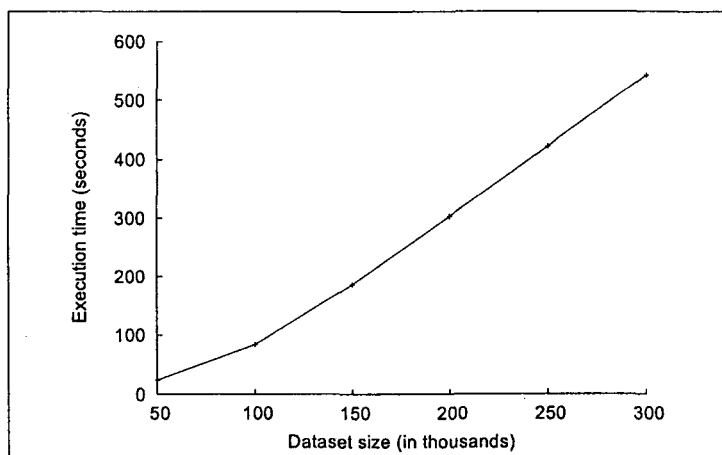


Figure 7.1: Scalability of *SMIC* to the no. of records.

Mushroom dataset with 22 attributes. The same clustering result reported in *Table 7.12* is obtained for any value of *MinAtt* in the range [2,16], with the other parameter δ_2 set to 4. Acceptable results are obtained for δ_2 in the range [2, 6]. It was observed in all the experiments that δ_2 should be provided with a value which is less than 30% of the dimensionality of the dataset, minimum possible value being 1. In general value of δ_1 should be set to 1. δ_3 should be greater than or equal to δ_2 . A value greater than 2 can be used for *MinObj*, typical values are 4, 8. Larger values can be used if small clusters are not acceptable.

It is clear from the results presented that the proposed algorithm can produce good quality results for small or large datasets with categorical or mixed type attributes.

Table 7.5: Clustering result on KDD CUP Corrected dataset.

Cluster-no	Normal	Attack	Cluster-no	Normal	Attack
1	11625	9571	21	8851	0
2	4	28910	22	0	249
3	36	131178	23	0	3812
4	0	1451	24	0	119
5	0	61	25	0	259
6	0	239	26	0	143
7	0	359	27	0	124
8	2	281	28	0	37957
9	0	20	29	0	44
10	2	5279	30	0	337
11	53	107	31	614	0
12	457	0	32	0	525
13	0	22	33	0	180
14	0	2403	34	0	84
15	0	138	35	1888	0
16	3369	0	36	0	92
17	0	22	37	413	0
18	155	0	38	0	195
19	0	70	39	0	21
20	0	16164	40	33124	10020

Table 7.6: Clustering result on soybean dataset.

Clusters	D1	D2	D3	D4
C_1	10	0	0	0
C_2	0	10	0	0
C_3	0	0	10	0
C_4	0	10	0	17

Table 7.7: Accuracy on soybean dataset.

Algorithm	Accuracy
<i>SMIC</i>	1.00
<i>k-sets</i>	1.00
<i>CatSub</i>	0.96
<i>k-modes</i>	0.95
<i>SUBCAD</i>	0.93

Table 7.8: The misclassification matrix of result on zoo dataset.

Clusters	1	2	3	4	5	6	7
C_1	37	0	0	0	0	0	0
C_2	0	20	1	0	0	0	0
C_3	4	0	0	0	0	0	0
C_4	0	0	3	13	0	0	0
C_5	0	0	1	0	4	0	0
C_6	0	0	0	0	0	8	1
C_7	0	0	0	0	0	0	9

Table 7.9: Clustering result on Congressional Voting dataset.

Cluster	Republican	Democrat
C_1	155	43
C_1	9	218
C_2	4	6

Table 7.10: Clustering result on breast cancer dataset.

Cluster	Benign	Malignant
C_1	409	2
C_2	0	10
C_3	49	229

Table 7.11: Accuracy on breast cancer dataset.

Algorithm	Accuracy
<i>SMIC</i>	0.92
<i>CatSub</i>	0.91
<i>SUBCAD</i>	0.87

Table 7.12: Clustering result on Mushroom dataset.

Clusters	1	2	3	4	5	6	7	8	9	10
<i>Edible</i>	192	0	288	0	32	1744	0	0	192	48
<i>Poisonous</i>	0	1722	0	36	72	0	1304	288	0	0
Clusters	11	12	13	14	15	16	17	18	19	
<i>Edible</i>	48	0	0	0	765	518	96	186	99	
<i>Poisonous</i>	0	32	198	264	0	0	0	0	0	

Table 7.13: Clustering result by ROCK on Mushroom dataset.

Clusters	1	2	3	4	5	6	7	8	9	10	11
<i>Edible</i>	96	0	704	96	768	0	1728	0	0	0	48
<i>Poisonous</i>	0	256	0	0	0	192	0	32	1296	8	0
Clusters	12	13	14	15	16	17	18	19	20	21	
<i>Edible</i>	48	0	192	32	0	288	0	192	16	0	
<i>Poisonous</i>	0	288	0	72	1728	0	8	0	0	36	

Table 7.14: Execution time for the datasets.

Datasets	Objects	Attributes	Classes	Accuracy	Time(s)
Soybean small	47	35	4	1.00	0.01
Zoo	101	16	7	0.90	0.02
Congressional Votes	435	16	2	0.89	0.2
Credit approval	690	15	2	0.86	0.18
Wisconsin breast cancer	699	10	2	0.92	0.27
Mushroom	8124	22	2	0.99	4
KDD CUP Corrected	311029	41	2	0.93	575

Part V

Application Development

Our techniques for solving two real life problems are the content of this part. *Chapter 8* contains biclustering of gene expression data. A technique for network intrusion detection is presented in *Chapter 9*.

Chapter 8

Biclustering Gene Expression Data Using A Node Addition Algorithm

8.1 Introduction

DNA microarray technology has made it possible to simultaneously monitor the expression level of thousands of genes during important biological processes and across collection of related samples. The samples may come from different tissues, organs, or individuals. The samples may also correspond to different time points or different environmental conditions. This kind of data, are arranged in a data matrix, where each row represents a gene and each column a condition. Each element of this matrix is a real number, which represents the expression level of a gene under a specific condition.

It is not easy to interpret the meaning of huge amount of gene expression data. A first step towards addressing this challenge is the use of clustering techniques, which are essential for extracting correlated patterns and natural classes present in the data. Gene expression data clustering can be performed in two ways - (1) grouping of genes according to their expression under multiple conditions;

(2) grouping of conditions based on the expression of a number of genes. However, many activation patterns are common to a group of genes only under specific experimental conditions. It is therefore desirable to develop a distinct type of clustering algorithm, known as biclustering, that performs simultaneous grouping of genes and conditions. Biclustering was first described in the literature by Hartigan [Har72]. Cheng and Church [CC00], were the first to introduce biclustering in gene expression data analysis. They introduced the concept of mean squared residue score to capture the coherence of a subset of genes under a subset of conditions. The goal of biclustering is to find biggest volume (the size of the bicluster in terms of number of entries) biclusters with lowest mean squared residues. Moreover, the row variance should be large enough to eliminate the trivial biclusters where the subset of genes do not have any fluctuation or have very little fluctuations. It has been proved that the problem of finding biclusters satisfying these criteria is NP-hard in general.

A set of heuristic algorithms using the concept of node deletion/addition were designed by Cheng and Church [CC00] to either find one bicluster or a set of biclusters. The main drawback of their method is that discovered biclusters need to be masked with random values so that successive runs discover new biclusters. There exists substantial risks that these random numbers will interfere with the future discovery of biclusters, especially those cases that have overlap with discovered ones. Yang et al. [YWWY03] further developed the ideas of Cheng and Church by dealing with missing values in the bicluster. They introduced the FLOC algorithm to find several biclusters simultaneously. Many other biclustering algorithms [LO02, KBCG03, ZTOT04] are proposed in the literature to perform gene expression data analysis. A survey of biclustering algorithms for biological data can be found in [MO04].

We provide an efficient node addition algorithm to find a set of biclusters without the need of masking discovered biclusters. Use of incremental method of computing score makes the algorithm faster. Initialized with a gene and a

subset of conditions, a bicluster is extended by adding more genes and conditions until its score approaches *delta* or no more additions are possible. The method provides facility to study individual genes, besides generating a large number of biclusters with different initializations. Biclusters with lower or higher scores within a specified limit can be generated by parameter setting, thus increasing the scope for detecting interesting biclusters. If some smaller biclusters are found to be interesting they can be extended again and studied further.

8.2 Problem Formulation

The gene expression matrix with N rows and M columns is represented as:

$$\begin{aligned} A &= \{a_{ij}, i = 1, 2, \dots, N, j = 1, 2, \dots, M\} \\ &= \{X_i, i = 1, 2, \dots, N\} = \{Y_j, j = 1, 2, \dots, M\} \end{aligned}$$

Each entry a_{ij} in this matrix corresponds to the logarithm of the relative abundance of *mRNA* of gene X_i under a specific condition Y_j . The i -th gene and the j -th condition are given by $X_i = \{a_{i1}, a_{i2}, \dots, a_{iM}\}$ and $Y_j = \{a_{1j}, a_{2j}, \dots, a_{Nj}\}$. The i -th gene X_i can be denoted by its label i only. Let, $I \subseteq \{1, 2, \dots, N\}$ is a subset of genes and $J \subseteq \{1, 2, \dots, M\}$ be a subset of conditions. We use A_{IJ} or simply (I, J) to denote the submatrix of A that contains only the elements a_{ij} belonging to set of rows I and set of columns J . For example, consider a sample expression matrix shown in Equation 8.1. For subset of rows $I = \{1, 3, 5\}$ and subset of columns $J = \{2, 4, 5, 7\}$ the submatrix A_{IJ} is shown in Equation 8.2.

$$A = \left\{ \begin{array}{cccccccc} 36 & 70 & 87 & 40 & 55 & 24 & 50 & 54 \\ 52 & 14 & 56 & 19 & 22 & 57 & 89 & 99 \\ 84 & 80 & 76 & 50 & 65 & 71 & 60 & 30 \\ 64 & 98 & 35 & 46 & 25 & 33 & 32 & 91 \\ 27 & 70 & 93 & 30 & 45 & 66 & 40 & 40 \end{array} \right\} \quad (8.1)$$

$$A_{IJ} = \begin{pmatrix} 70 & 40 & 55 & 50 \\ 80 & 50 & 65 & 60 \\ 70 & 30 & 45 & 40 \end{pmatrix} \quad (8.2)$$

Such a submatrix A_{IJ} is called a bicluster. Given a gene expression data set, the problem is to extract biclusters that exhibit some form of homogeneity. The values of each row of the bicluster shown in Equation 8.2 are plotted using line graphs in Figure 8.1. It can be seen from the figure that values of the first and the second row rise and fall with complete coherence. All the rows are not completely coherent. Still it can be said that the bicluster is homogeneous to some extent. To measure this homogeneity we use *MeanSquareResidue*, which was originally used by Cheng and Church [CC00]. The following notations concerning a bicluster A_{IJ}

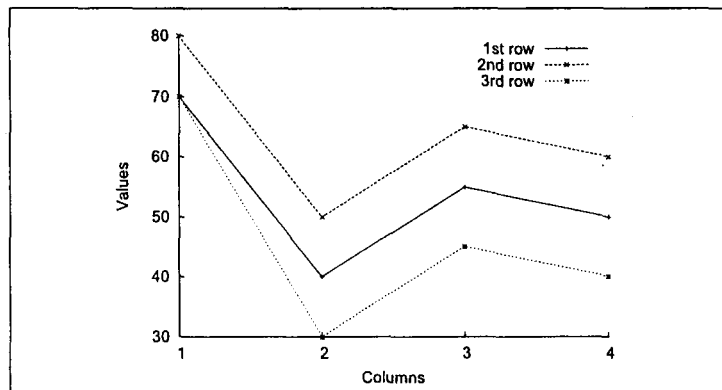


Figure 8.1: Coherence patterns in submatrix A_{IJ}

are used here :

a_{iJ} is the mean of the i -th row,

a_{Ij} is the mean of the j -th column,

$a_{IJ} = \frac{1}{|I||J|} \sum_{i \in I, j \in J} a_{ij} = \frac{1}{|I|} \sum_{i \in I} a_{iJ} = \frac{1}{|J|} \sum_{j \in J} a_{Ij}$, is the mean of the submatrix,

$X_i^J = \{a_{ij}, j \in J\}$ is a row vector defined over set of columns J ,

$CM^J = \{a_{Ij}, j \in J\}$ is the vector of column means,

$RM^I = \{a_{iJ}, i \in I\}$ is the vector of row means.

Let $t(X_i^J)$ represents transformed row X_i^J obtained by subtracting the row mean from the value of each element of the row, so that

$$t(X_i^J) = \{a_{ij} - a_{iJ}, j \in J\} \quad (8.3)$$

The coherence score between two rows X_i^J and X_k^J is defined as :

$$\begin{aligned} C(X_i^J, X_k^J) &= \frac{1}{|J|} (t(X_i^J) - t(X_k^J))^2 \\ &= \frac{1}{|J|} \sum_{j \in J} (a_{ij} - a_{iJ} - a_{kj} + a_{kJ})^2. \end{aligned} \quad (8.4)$$

A score of zero indicates complete coherence. Higher scores indicate more incoherence. For a bicluster A_{IJ} consider a fixed row that may or may not belong to the bicluster. Coherence score of each row of the bicluster is the coherence score between the row and the fixed row. Let this fixed row be the column means vector, CM^J . The mean squared residue score for the i -th row X_i , denoted by $H(i)$ is nothing but the coherence score between X_i and the column means vector CM^J , so that :

$$\begin{aligned} H(i) &= C(X_i^J, CM^J) = \frac{1}{|J|} (t(X_i^J) - t(CM^J))^2 \\ &= \frac{1}{|J|} \sum_{j \in J} (a_{ij} - a_{iJ} - a_{Ij} + a_{IJ})^2 \\ &= \frac{1}{|J|} \sum_{j \in J} (r_{ij})^2 = \frac{1}{|J|} (R_i)^2 \end{aligned} \quad (8.5)$$

where, $r_{ij} = a_{ij} - a_{iJ} - a_{Ij} + a_{IJ}$, $i \in I$, $j \in J$ is the residue of each element of the i -th row and $R_i = \{r_{ij}, j \in J\}$ is the row vector of residues. Note that $R_i = t(X_i^J) - t(CM^J)$. It can be shown easily that,

$$\sum_{i \in I} r_{ij} = 0 \text{ and } \sum_{j \in J} r_{ij} = 0 \quad (8.6)$$

The mean squared residue score (or simply, score) $H(I, J)$ of a bicluster (I, J) is computed by taking the average of mean squared residue score of all rows of the bicluster.

$$\begin{aligned} H(I, J) &= \frac{1}{|I|} \sum_{i \in I} H(i) \\ &= \frac{1}{|I||J|} \sum_{i \in I, j \in J} (a_{ij} - a_{iJ} - a_{Ij} + a_{IJ})^2 \end{aligned} \quad (8.7)$$

A bicluster (I, J) with mean squared residue score $H(I, J) < \delta$ for some $\delta > 0$ is called a δ -bicluster. The row variance of a bicluster (I, J) is defined as

$$V(I, J) = \frac{1}{|I||J|} \sum_{i \in I, j \in J} (a_{ij} - a_{iJ})^2 \quad (8.8)$$

Let $H'(I, J)$ be the mean squared residue score of bicluster (I, J) computed with respect to a row vector $Z^J \neq CM^J$, that may or may not belong to the bicluster.

Lemma 7 $H'(I, J) > H(I, J)$.

Proof : Let, $R'_i = \{r'_{ij}, j \in J\}$ be the row vector of residues corresponding to i -th row computed with respect to $Z^J \neq CM^J$ and $R_i = \{r_{ij}, j \in J\}$ be the corresponding residue vector computed with respect to $CM^J = \{a_{Ij}, j \in J\}$. Let, $W = \{w_i, j \in J\}$ be a row vector such that :

$$W = t(CM^J) - t(Z^J) \quad (8.9)$$

We have,

$$R'_i = t(X_i^J) - t(Z^J) \text{ and } R_i = t(X_i^J) - t(CM^J)$$

So,

$$\begin{aligned} R'_i - R_i &= t(X_i^J) - t(Z^J) - t(X_i^J) + t(CM^J) \\ &= t(CM^J) - t(Z^J) \\ &= W \end{aligned}$$

Hence,

$$R'_i = R_i + W \text{ with } \{r'_{ij} = r_{ij} + w_j, j \in J\}$$

Now,

$$\begin{aligned}
H'(I, J) &= \frac{1}{|I|} \sum_{i \in I} H'(i) \\
&= \frac{1}{|I|} \sum_{i \in I} (R'_i)^2 \\
&= \frac{1}{|I|} \sum_{i \in I} (R_i + W)^2 \\
&= \frac{1}{|I||J|} \sum_{i \in I} \sum_{j \in J} (r_{ij} + w_j)^2 \\
&= \frac{1}{|I||J|} \sum_{i \in I} \sum_{j \in J} ((r_{ij})^2 + 2r_{ij}w_j + (w_j)^2) \\
&= \frac{1}{|I||J|} \sum_{i \in I} \sum_{j \in J} (r_{ij})^2 + \frac{1}{|I||J|} \sum_{i \in I} \sum_{j \in J} 2r_{ij}w_j + \frac{1}{|I||J|} \sum_{i \in I} \sum_{j \in J} (w_j)^2 \\
&= H(I, J) + \frac{2}{|I||J|} \sum_{i \in I} r_{ij} \sum_{j \in J} w_j + \frac{1}{|J|} \sum_{j \in J} (w_j)^2 \\
&= H(I, J) + \frac{1}{|J|} \sum_{j \in J} (w_j)^2 \text{ [since } \sum_{j \in J} r_{ij} = 0\text{]} \\
&= H(I, J) + \text{a positive quantity} \\
&\Rightarrow H'(I, J) > H(I, J)
\end{aligned}$$

□

8.2.1 Incremental computation of score

Calculation of the mean squared residue score or simply the score of a bicluster, $H(I, J)$ requires $O(nm)$ effort, where $n = |I|$ and $m = |J|$. After adding a row

or a column to a bicluster its new score can be computed from scratch requiring $O(nm)$ effort again. But storing the present score, row means and column means, the new score can be computed easily by spending only $O(m)$ or $O(n)$ effort depending upon whether a new row or a new column is added. The procedure is described below.

Consider a bicluster $A_{IJ} = (I, J)$ with mean squared residue score $H(I, J)$, $n = |I|$, $m = |J|$. Let after adding a new row $X_k (k \notin I)$, the new set of rows becomes $I' = I \cup k$. New column means vector $CM'^J = \{a_{I'j}, j \in J\}$ is computed from CM as:

$$a_{I'j} = \frac{na_{Ij} + a_{kj}}{n+1}, j \in J$$

Bicluster mean for (I', J) is given by

$$a_{I'J} = \frac{1}{m} \sum_{j \in J} a_{I'j}$$

Score for the k -th row is computed as

$$H(k) = \frac{1}{m} \sum_{j \in J} (a_{kj} - a_{kJ} - a_{I'j} + a_{I'J})$$

The new score of the bicluster is computed by summing the score contributed by the new row a_{kj} , $j \in J$, and the new score contributed by the old matrix (I, J) , and then dividing the sum by $(n+1)m$. That is

$$H(I', J) = \frac{mH(k) + mnH'(I, J)}{(n+1)m}$$

where,

$$H'(I, J) = H(I, J) + \frac{1}{m} \sum_{j \in J} (w_j)^2$$

$$w_j = a_{ij} - a_{iJ} - a_{Ij} + a_{I'J}$$

New score after adding a column can be computed in an analogous way.

8.3 Proposed Algorithm

Given the expression matrix A and a threshold δ , a node addition algorithm is provided here to extract biclusters (δ -clusters) with mean squared residue score less than δ . It finds one bicluster at a time starting with an initial bicluster which is extended by adding more rows and columns. Different initialization will create possibly different biclusters. An incremental method of computing score is used. This requires the row means (RM^J), column means (CM^I) and the score ($H(I, J)$) to be stored. In the algorithm presented below, it is assumed that these parameters along with the given expression matrix, a_{ij} , $i \in \{1, 2, \dots, N\}$, $j \in \{1, 2, \dots, M\}$, the size of the bicluster (I, J) in terms of number of rows (n) and columns (m) are global values.

8.3.1 Initializing the cluster

The initial bicluster (I, J) may be as small as a single row containing a subset of columns or it may be a large bicluster with the possibility of further extension. If not given as input, the initial biclusters may be created by randomly selecting a row $t \in \{1, 2, \dots, N\}$, and a subset of columns J , such that $I = \{t\}$, $J \subseteq \{1, 2, \dots, M\}$, $score = 0$, $CM^J = \{a_{tj}, j \in J\}$, $RM^I = \{a_{iJ}\}$.

8.3.2 Extending the cluster

An initial bicluster (I, J) with $|I| = n$, $|J| = m$ and $score \ll \delta$ is given. More rows and columns can be added to the cluster allowing the $score$ to increase up to δ . First, row extension is performed keeping the set of columns fixed. Then column extension is tried, keeping set of rows fixed. Repetition of the two steps are needed to extract a bicluster.

8.3.3 Row extension

A simple procedure for adding rows is to visit each row of the data matrix A , include it in the bicluster if the row is not included already and the score after inclusion still remains less than δ . This simple procedure may greedily insert some less coherent rows which greatly increase the *score*, thus preventing later inclusion of more rows. As a result biclusters with lesser volume will be obtained. To avoid this problem we can make one scan over A , compute new *score* for every possible insertion of a row that is not already included in the cluster and insert that particular row which causes the least increase in the *score* of the bicluster. This procedure will be very costly, since it inserts a single row in a complete scan over the data matrix. We adopt an intermediate procedure that adds several rows in a single scan so that fewer scans are needed to get a bicluster. A row passes through two screenings before being inserted in a cluster.

8.3.4 First screening

The column means of the initial bicluster before extension is stored in the row vector CM^J . Coherence score of each row i with respect to CM^J , i.e. $C(X_i^J, CM^J)$ is computed. If the $C(X_i^J, CM^J)$ value is greater than a threshold θ the row is not considered for addition. The value of θ is computed from the value of δ as $\theta = \alpha\delta$, where $\alpha > 0$ is another constant. Lower value of α leads to detection of more coherent biclusters with lower *score*, while higher value of α finds biclusters with higher *score*.

8.3.5 Second screening

In the second screening the amount of increase in score is considered. A row is added only if increase in score satisfies the following condition :

$$\begin{aligned} \text{newscore} - \text{score} &\leq \delta - \text{newscore} \\ \Rightarrow \text{newscore} &\leq (\text{score} + \delta)/2 \end{aligned} \quad (8.10)$$

Here, *score* and *newscore* respectively represent the present score and the new score obtained after including the row in the bicluster. The row extension procedure is given below.

Procedure extendRow(θ, δ)

Inputs : Data matrix, $A = a_{ij}$, $i = 1..N$, $j = 1..M$, bicluster (I, J) with $\text{score} = H(I, J)$, column means $CM^J = \{a_{Ij}, j \in J\}$, row means $RM^I = \{a_{iJ}, i \in I\}$.

Outputs: New bicluster (I', J) with $I' \supseteq I$.

Steps:

1. for each row $i \in \{1..N\}$, $i \notin I$ repeat steps 2-10;
2. if $C(X_i^J, CM^J) \geq \theta$ skip row i ;
3. $I' = I \cup \{i\}$;
4. incrementally compute, $\text{newscore} = H(I', J)$;
5. if $(\text{newscore} > (\text{score} + \delta)/2)$ skip steps 6-10;
6. $CM^J = \{(n * a_{Ij} + a_{ij})/(n + 1), j \in J\}$;
7. $RM^I = RM^I \cup \{a_{iJ}\}$;
8. $I = I'$;
9. $n = n + 1$;
10. $\text{score} = \text{newscore}$;
11. return;

8.3.6 Column extension

A random subset of columns is used in the initial bicluster. So, $|J|$ may be as big as M . If $|J| < M$ column extension need to be performed. Column extension should not increase the *score* in greater amounts as number of columns is less compared to number of rows. Otherwise lesser scope will remain for adding more rows. A threshold $\delta' \leq \delta$ is used for column extension. The procedure is given below.

Procedure extendColumn(δ')

Inputs : Data matrix, $A = a_{ij}$, $i = 1..N$, $j = 1..M$, bicluster (I, J) with *score* = $H(I, J)$, column means $CM^J = \{a_{Ij}, j \in J\}$, row means $RM^I = \{a_{iJ}, i \in I\}$.

Outputs: New bicluster (I, J') with $J' \supseteq J$.

Steps:

1. for each column $j \in \{1..M\}$, $j \notin J$ repeat steps 2-9;
2. $J' = J \cup \{j\}$;
3. incrementally compute, *newscore* = $H(I, J')$;
4. if (*newscore* > (*score* + δ')/2) skip steps 5-9;
5. $RM^I = \{(m * a_{iJ} + a_{ij}) / (m + 1), i \in I\}$;
6. $CM^J = CM^J \cup \{a_{Ij}\}$;
7. $J = J'$;
8. $m = m + 1$;
9. *score* = *newscore*;
10. return;

To extract a bicluster a few calls to the *extendRow()* and *extendColumn()* procedures should be made after initializing it. First *extendRow()* procedure is called. If no extension is resulted a new initialization should be tried. A single

call to *extendRow()* procedure will detect a bicluster whose score may be far less than δ . It is extended further by performing a column extension. Then repeated calls to *extendRow()* procedure can be made till there is no addition of rows. Practical execution shows that number of new rows added during later calls goes on diminishing and after a few calls row extension stops completely. Leaving a little scope for further expansion we may use only a single call to *extendRow()* procedure at this place. It need to be ensured that the cluster can not be extended without allowing the *score* to increase. So, calls to *extendColumn()* and *extendRow()* procedures should be made once again with δ set to present *score*. According to lemma 7, these last calls will add columns or rows (if any) that will not increase the present score but will possibly decrease it. The complete procedure for extracting $T(\leq N)$ number of biclusters is given below. In the algorithm, the step 8 can be repeated until no addition of rows is possible. In that case biclusters with score nearly equal to δ will be obtained.

Procedure findBiclusters()

Inputs : Data matrix, $A = a_{ij}$, $i = 1..N$, $j = 1..M$, score limit δ , constant α , constant T .

Outputs: A set of T biclusters with scores $\leq \delta$.

Steps :

1. repeat steps 2-12 for T times;
2. randomly select $J \subseteq \{1, 2, \dots, M\}$, randomly select $i \in \{1, 2, \dots, N\}$, set $I = \{i\}$, $score = 0$, $CM^J = \{a_{ij}, j \in J\}$, $RM^I = \{a_{iJ}, i \in I\}$;
3. $\theta = \alpha\delta$;
4. *extendRow*(θ, δ);
5. if no extension is achieved go to step 2;
6. $\delta' = (score + \delta)/2$;
7. *extendColumn*(δ');

8. *extendRows*(θ , δ);
9. $\delta' = score$;
- 10 *extendColumn*(δ');
11. *extendRows*(θ , δ');
12. print the bicluster (I , J);
13. return;

8.4 Experimental Results

The biclustering algorithm is tested on two gene expression datasets downloaded from <http://arep.med.harvard.edu/biclustering>. The first dataset is the yeast data containing 2884 genes and 17 conditions. Integer valued elements range between 0 and 600 with 34 missing values. There are 4026 genes and 96 conditions in the second dataset, human lymphoma data. This matrix of integers range between -750 and 650, with 47639 missing values. We replaced missing values with uniformly distributed random numbers within the data range. These are the datasets used by Cheng and Church [CC00] to test their biclustering algorithm. We use the same δ values (300 for yeast dataset and 1200 for lymphoma dataset) as used by Cheng and Church so that the results become comparable.

The proposed algorithm can detect biclusters with lower or higher score within the given limit of δ by using different values for the parameter α . Lower values of α are used to detect small biclusters with lower scores while larger values of α cause detection of bigger biclusters whose scores approach the given limit of δ . This is demonstrated with the three biclusters shown in *Table 8.1* extracted from yeast dataset using three different values of α . Those three biclusters having increasingly bigger sizes are plotted in *Figure 8.2– 8.4* in order to visualize the coherence of the genes. The serial numbers of genes present in the first bicluster

are - (216, 217, 616, 1022, 1623, 1795, 2375, 2538), which are also present in the second bicluster -(216, 217, 526, 616, 1022, 1184, 1476, 1623, 1795, 2086, 2278, 2375, 2538) and the third bicluster - (58, 59, 105, 216, 217, 424, 448, 453, 457, 526, 616, 862, 972, 1022, 1184, 1286, 1320, 1417, 1475, 1476, 1590, 1623, 1725, 1795, 1981, 2086, 2087, 2278, 2375, 2467, 2538) along with other genes. These biclusters include more genes than bicluster no. 66 (score=164.06, genes : 217 616 1476 1623 1795 2086) reported by Cheng and Church in [CC00]. All these biclusters contains the full set of 17 conditions. It means that our method can detect bigger volume biclusters than Cheng and Church method.

Table 8.1: Sample biclusters in yeast dataset

α	δ	score	no. of genes	no. of conditions
1.0	300	144.15	8	17
1.2	300	198.93	13	17
2.0	300	295.03	31	17

Sample biclusters extracted from the human lymphoma dataset are shown in *Table 8.2* and are plotted in *Figures 8.5– 8.7*. All the 96 conditions are present in each of the three biclusters, genes present in the biclusters are respectively : (143, 144, 145), (124, 126, 143, 144, 145, 195, 2371) and (124, 126, 131, 136, 143, 144, 145, 158, 170, 172, 195, 205, ...).

We have extracted 100 biclusters from each of the yeast and lymphoma datasets by random row initialization. The results are presented in *Table 8.3* and *8.4*. Results of Cheng and Church(CC) method are also included for comparison. Again average volume of biclusters extracted by our algorithm are found to be bigger.

All the biclusters extracted from the datasets may not be biologically

Table 8.2: Sample biclusters in human dataset

α	δ	score	no. of genes	no. of conditions
1.0	1200	132.97	3	96
2.5	1200	1003.26	7	96
5.0	1200	1196.01	53	96

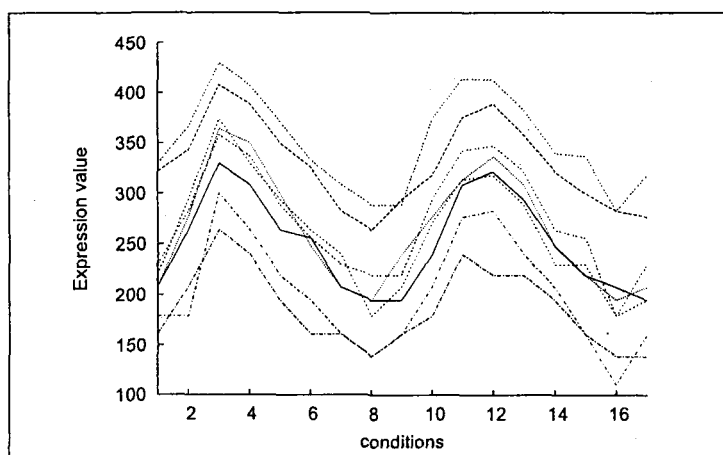


Figure 8.2: First bicluster in *Table 8.1*

interesting. We have not analyzed the biological significance of the biclusters. Instead of generating just 100 biclusters, our algorithm can generate a large number of biclusters. A set of biclusters with smaller residues and smaller volumes may be extracted for significance analysis. For example we got 208 biclusters with average score less than 100 from the yeast dataset. Each bicluster contains at least one gene not contained in other clusters. *Table 8.5* gives a summary of the biclusters. If some of them are found to be biologically interesting, they can be extended again and studied further. We may also filter out small biclusters with higher scores for study.

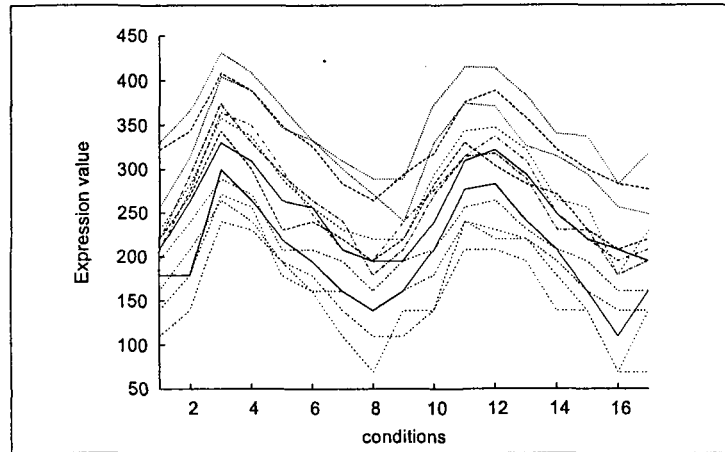


Figure 8.3: Second bicluster in *Table 8.1*

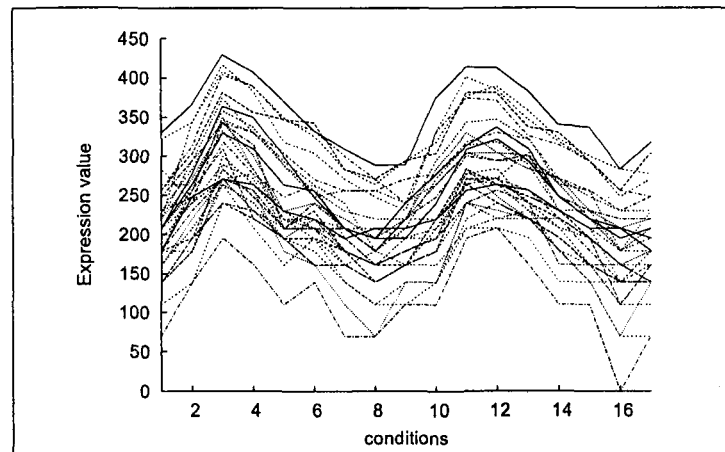


Figure 8.4: Third bicluster in *Table 8.1*

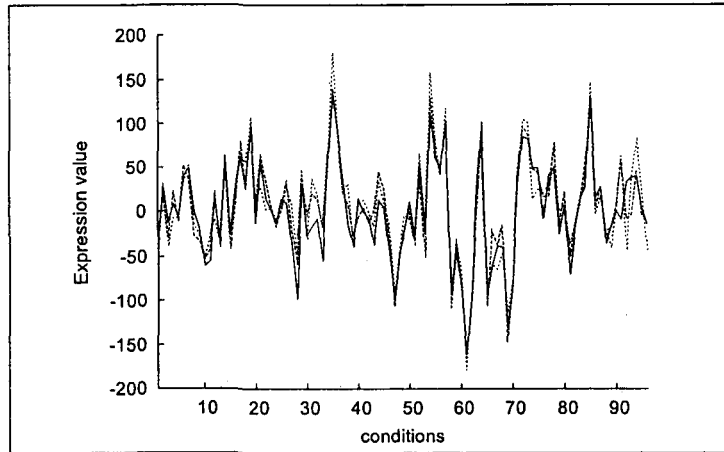


Figure 8.5: First bicluster in *Table 8.2*

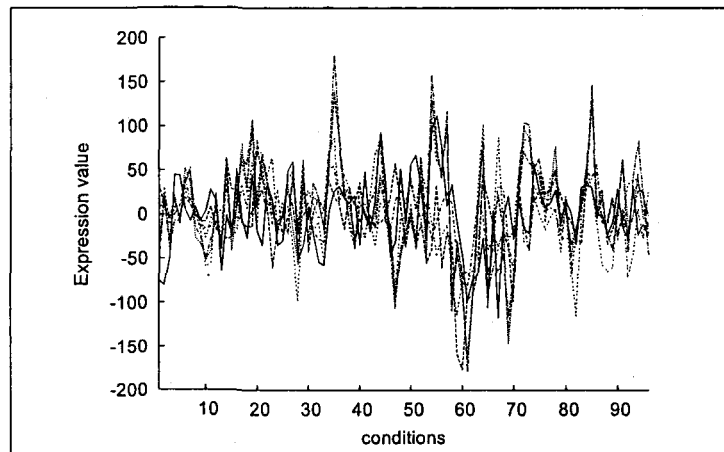


Figure 8.6: Second bicluster in *Table 8.2*

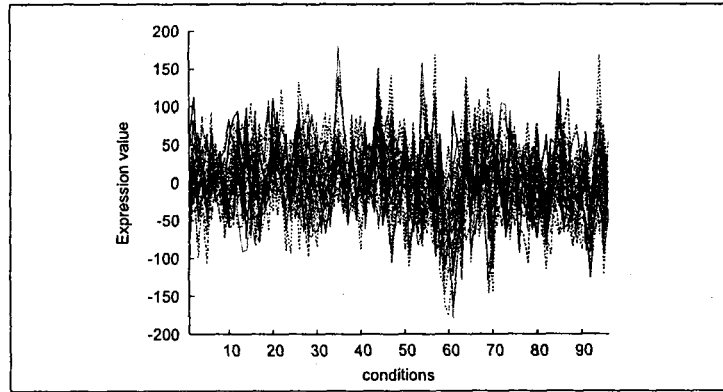


Figure 8.7: Third bicluster in *Table 8.2*

Table 8.3: Performance of proposed algorithm on yeast data set

Algo	Avg. score	Avg. genes	Avg. conds.	Avg. Vol.	row cov.	Cell cov.
CC	204.3	166.8	12.1	1577.0	0.91	0.81
Our	199.0	195.3	11.9	1773.2	0.96	0.79

Table 8.4: Performance of proposed algorithm on lymphoma data set

Algo	Avg. score	Avg. genes	Avg. conds.	Avg. vol.	row cov.	Cell cov.
CC	850.1	269.2	24.5	4596.0	0.92	0.37
Our	831.9	374.3	25.2	5615.0	0.93	0.44

Table 8.5: Summary of biclusters with score less than 100 from yeast data set

Avg. score	Avg. genes	Avg. cond.	Avg. Vol.	Row cov.	Cell cov.
74.6	39.9	11.9	354.7	0.73	0.49

Chapter 9

A Clustering Based Technique For Network Intrusion Detection

9.1 Introduction

Security of the information stored on computers connected to publicly accessible networks has become critical issue due to different types of intrusion attacks. Intrusion detection allows organizations to protect their systems from the threats that come with increasing network connectivity and reliance on information systems. According to [BM01] intrusion detection is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusions, defined as attempts to compromise the confidentiality, integrity, availability, or to bypass the security mechanisms of a computer or network. Intrusions are caused by attackers accessing the systems from the Internet, authorized users of the systems who attempt to gain additional privileges for which they are not authorized, and authorized users who misuse the privileges given to them. Intrusion Detection Systems (IDSs) are software or hardware products that automate this monitoring and analysis process.

Based upon information source intrusion detection systems are classified into network-based and host-based. Network-based IDSs analyze network packets, captured from network backbones or LAN segments, to find attackers. Host-based IDSs operate on information such as operating system audit trails, and system logs collected from within an individual computer system.

Different approaches are used in intrusion detection such as machine learning, pattern matching, neural networks, data mining, etc. Some approaches detect attacks in progress in real time while others provide after-the-fact information about the attacks providing help to reduce the possibilities of future attacks of the same type. In general there are two types of approaches for network intrusion detection : misuse detection and anomaly detection. Misuse detection searches for specific pattern (attack signature) in the data. Previously known attacks are effectively detected without generating large number of false alarms. Such methods can not detect new types of attacks because their signatures are not known. Anomaly detection builds models for normal behaviour and automatically detects significant deviations from it. Supervised or unsupervised learning can be used. In a supervised approach, the model is developed based on training data that are known to be normal. Unsupervised approaches work without any training data. The main advantage of anomaly detection is that it can detect previously unknown attacks, since no knowledge of attacks is needed to train the normality model. But, it may fail to detect some known attacks if the behaviour of them are not significantly different from what is considered to be normal. Moreover, the false alarm rate tends to be higher.

In recent years considerable attention has been given to data mining approaches for intrusion detection. Anomaly detection often tries to cluster test dataset into groups of similar instances - either attacks or normal data. Intrusion detection problem is then reduced to the problem of labeling the clusters as intrusive or normal traffic. For labeling, unsupervised anomaly detection algorithms model normal behaviour by using the following two assumptions - i)

the number of normal instances vastly outnumber the number of anomalies and ii) anomalies themselves are qualitatively different than the normal instances. If the assumptions hold attacks can be detected based on cluster sizes. Larger clusters correspond to normal data, and smaller clusters correspond to attacks. But this simple method is likely to produce lower detection rate as the assumptions are not always true. For example in denial of service attacks a large number of very similar instances are generated that may form larger clusters. On the others hand some less frequently used protocols such as *ftp* may generate few records that may be wrongly classified as an attack cluster. Some attacks such as *R2L* and *U2R* are qualitatively very similar to normal instances which means that such attacks may remain mixed with normal records creating impure clusters. Again, test data instances to be clustered contain a large number of features. All features are not equally important for distinguishing between different normal and attack instances. Therefore subspace based algorithm should produce better clustering result in this case.

We address these issues in the proposed unsupervised anomaly detection technique that uses the *CatSub* algorithm presented in *Chapter 6* to cluster the test dataset based on subspaces over which data records are highly similar. Anomalies are then detected on the basis of subspace sizes and also on cluster cardinality.

9.2 Related Works

Clustering is a widely used technique in anomaly detection. Some algorithms use training datasets and others work in unsupervised manner. Portnoy et al. [PES01] presented an unsupervised anomaly detection algorithm which train on unlabeled data in order to detect new intrusions. The training dataset is clustered using a modified incremental *k-means* algorithm. The algorithm starts with an empty set of clusters, and generates the clusters with a single pass through the dataset. For

each data instance in the dataset, it computes the distance between it and each of the centroids of the clusters in the cluster set so far. The cluster with the shortest distance is selected, and if that distance is less than some constant W (cluster width) then the instance is assigned to that cluster. Otherwise a new cluster is created with the instance as its center. This algorithm does not use any outlier handling method and consequently a large number of very small clusters may be created causing increase in execution time of the algorithm. After clustering, each cluster is labeled as normal or intrusive based on the number of instances in the cluster. Some percentage of the clusters containing the largest number of instances were labeled as normal and the rest of the clusters were labeled as anomalous. The labeled clusters were then used to detect intrusions in test datasets. A test instance is given the cluster label of the cluster which is closest to the instance.

Yu Guan et al. [GGB03] presented the *Y-means* algorithm which is an improved *k-means* algorithm. The algorithm handles outliers by splitting and merging clusters that automatically adjust the number of clusters k . No training data is used. Clusters are labeled according to their population, that is, if the population ratio of one cluster is above a given threshold, all the instances in the cluster will be classified as normal; otherwise they are labeled intrusive.

Most anomaly detection algorithms require a set of purely normal data to train the model, and they implicitly assume that anomalies are outliers i.e. patterns not observed before. Lazarevic et al. [LEK⁺03] focus on several outlier detection schemes in order to see how efficiently these schemes may deal with the problem of anomaly detection.

ADWICE (Anomaly Detection With fast Incremental Clustering) [BNT04] uses the first phase of the existing BIRCH clustering framework to implement fast, scalable and adaptive anomaly detection. It uses training data assumed to consist only of normal data to construct the CF tree. After being trained, it is used to detect anomalies in unknown data. When a new data point arrives detection starts with a top down search from the root to find the closest cluster feature. When

search is done, the distance from the centroid of the cluster to the new data point is computed. The new data point is considered normal if the distance is lower than a limit otherwise it is an anomaly. The number of alarms is then further reduced by application of an aggregation technique.

In [LL05] Leung et al. proposed a density based and grid based clustering algorithm, named as *fpMAFIA*, that uses adaptive grid algorithm adopted from *pMAFIA* and *FP-tree* growth method for frequent itemset mining. They aim to discover clusters from large volume of high dimensional input data. Any point that falls inside the clusters are labeled as normal. The small percentage of points that do not belong to any clusters are labeled as abnormal.

S. Petrovic et al. [PAOC06] used the *k-means* algorithm for clustering and proposed a cluster labeling strategy based on a combination of clustering evaluation techniques. The Davis Bouldin clustering evaluation index and the comparison of centroid diameters of the clusters are combined in order to respond adequately to the properties of attack vectors. Compactness of the corresponding clusters and separation between them distinguish between normal from abnormal behaviour in the analyzed network.

9.3 Methodology

Our algorithm works based on two principles. First, the clustering algorithm should be such that it is able to distinguish minor differences between normal and attack instances so that as far as possible pure clusters are formed with only one kind of instances - either attack or normal. Secondly, besides cluster sizes some other criteria need to be used for labeling clusters.

9.3.1 Clustering

The dataset to be clustered is high dimensional containing many attributes. For example KDD CUP 1999 dataset contains 41 numeric and categorical attributes. All attributes are not equally important for distinguishing between different normal and attack records. Therefore we use a subspace based algorithm for clustering the dataset. Certainly the mixed-type data clustering algorithm, *SMIC*, proposed in *Chapter 7* would have been a better choice. But, we want to utilize another strategy - conversion of attributes (refer *Chapter 7*) for dealing with mixed-type data. Continuous attributes can be easily converted to categorical type by discretization (taking logarithm to the base 2). Therefore, the clustering algorithm used here is our categorical clustering algorithm, *CatSub* presented in *Chapter 6*. The assumptions used for detecting anomalies are :

1. Some attacks are similar over very large subspaces, other attacks are similar over smaller subspaces or have lower occurrences.
2. Normal records are similar over medium sized subspaces.

9.3.2 Detection

Detection is done in two phases. In the first phase clustering is performed by tuning the *MinAtt* parameter of the *CatSub* algorithm so that clusters produced are of larger subspaces only. The clusters so produced are labeled as attacks based upon our first assumption. As normal records are not similar over very large subspaces they will be separated by the clustering algorithm into a group of outliers. Attacks that form smaller or medium subspaces also become outliers.

In the second phase the cluster containing outliers is clustered again. This time the parameter indicating minimum subspace size (*MinAtt*) is set to a low value. Outliers, if found, are labeled as attacks. The clusters with cardinalities

below a specified threshold are also labeled as attacks. Remaining clusters contain normal records.

9.4 Experimental Results

In this section we perform performance evaluation of the proposed anomaly detection algorithm. The experiments were done in a 1.66 GHz HCL laptop with 512 MB RAM. C++ programs were used in LINUX environment.

9.4.1 Dataset description

We tested the algorithm on the *Corrected* dataset available in KDD Cup 1999 intrusion detection benchmark datasets [UoC99] containing a wide variety of intrusions simulated in a military network environment. The dataset contains 311029 data records, each represents a connection between two network hosts according to some well defined network protocol and is described by 41 attributes (38 continuous or discrete numeric attributes and 3 categorical attributes) such as duration of connection, number of bytes transferred, number of failed login attempts, etc. Each record is labeled as either normal or one specific kind of attack. There are 37 different attacks present in the dataset. The attacks fall in one of the four categories : User to Root (*U2R*), Remote to local (*R2L*), Denial of Service (*DOS*) and *PROBE*.

- Denial of Service(*DOS*) : Attackers try to prevent legitimate users from using a service. For example, SYN flood, smurf, teardrop etc.
- Remote to Local (*R2L*) : Attackers do not have an account on the victim machine, hence try to gain access. For example, guessing password.
- User to Root (*U2R*) : Attackers have local access to the victim machine and

try to gain super user privilege. For example, buffer overflow attacks.

- *PROBE* : Attacker tries to gain information about the target host. For example, Port-scan, ping-sweep etc.

Number of samples of each category of attack in *Corrected* KDD dataset is shown in *Table 9.1*. It can be noticed that number of *DOS* attacks far more exceeds the

Table 9.1: Attacks distribution in *Corrected* KDD dataset

<i>DOS</i>	<i>U2R</i>	<i>R2L</i>	<i>PROBE</i>	<i>Normal</i>	<i>Total</i>
229853	70	16347	4166	60593	311029

normal instances which is not expected in practice. It is because the goal of the KDD datasets was to produce good training sets for learning methods that use labeled data. The labels are not used during the clustering process, but are used for evaluating the detection performance of the algorithm.

9.4.2 Performance measures

We report the *detection rate (DR)* and the *false positive rate(FPR)* for evaluating the performance of the proposed anomaly detection algorithm. The *detection rate* is defined as the number of intrusion instances successfully detected divided by the total number of intrusion instances present in the dataset. The *false positive rate* is defined as the number of normal instances incorrectly labeled as intrusion divided by the total number of normal instances. A good method should provide high *detection rate* together with low *false positive rate*. The trade-off between the *detection rate* and *false positive rate* is reported by using Receiver Operating Characteristic(*ROC*) curves. An intrusion detection system can operate at any point on the *ROC*

curve. To prepare the *ROC* curve different values of *detection rates* and *false positive rates* are obtained by varying one parameter (*MinObj*) in the clustering algorithm.

Performance of the first phase of the algorithm is shown in *Table 9.2*. Detection rate of the individual attack classes are also shown in *Table 9.3*. It can be seen that the first phase has difficulty in detecting *U2R* and *PROBE* attacks but it detects *DOS* attacks with higher accuracy. Performance of the algorithm as a whole (including both first and second phases) is shown in *Table 9.4*. Performance of the algorithm in detecting individual attack categories is shown in *Table 9.5*. The *detection rate* becomes higher as the second phase is able to detect some attacks that could not be detected in the first phase. *Figure 9.1* shows the *ROC* curve for the algorithm. It can be seen that detection rate remains higher than 90%. The area under the *ROC* curve is more than many other anomaly detection methods reported in the literature, which indicates that our method is very promising.

Table 9.2: Performance of first phase.

<i>MinObj</i>	<i>False positive rate</i>	<i>Detection rate</i>
4	0.0	0.662
40	0.042	0.723
100	0.128	0.844
600	0.159	0.909
1000	0.315	0.966

Table 9.3: *Detection rate* of individual attack classes after the first phase

<i>MinObj</i>	<i>U2R</i>	<i>R2L</i>	<i>DOS</i>	<i>PROBE</i>
4	0	0.8999	0.9771	0.6306
40	0	0.6847	0.9353	0.3802
100	0	0.5053	0.8799	0.2259
600	0	0.1982	0.7742	0.0007
1000	0	0.0000	0.7214	0.0007

Table 9.4: Performance of the algorithm.

<i>MinObj</i>	<i>False positive rate</i>	<i>Detection rate</i>
4	0.015	0.917
40	0.057	0.936
100	0.143	0.961
600	0.174	0.975
1000	0.360	0.982

Table 9.5: *Detection rate* of individual attack classes after the second phase

<i>MinObj</i>	<i>U2R</i>	<i>R2L</i>	<i>DOS</i>	<i>PROBE</i>
4	0.5000	0.9279	0.9862	0.9606
40	0.6714	0.7545	0.9931	0.8502
100	0.6000	0.5980	0.9898	0.7991
600	0.6143	0.2909	0.9845	0.7736
1000	0.6142	0.0927	0.9786	0.7736

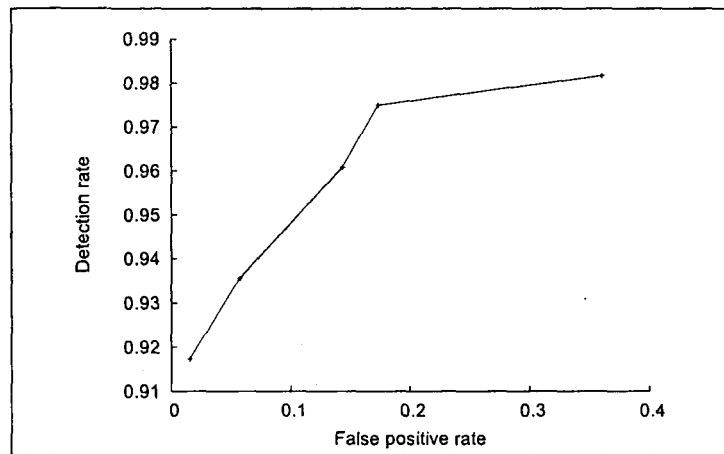


Figure 9.1: ROC curve

Chapter 10

Conclusion

We have attempted to develop clustering algorithms possessing certain requirements that a good clustering algorithm should possess. Our main concern is developing faster algorithms so that larger datasets can be clustered as demanded by present day data mining applications. To achieve this we have proposed a sampling procedure, a parallel processing technique, and incremental clustering algorithms. Finding clusters with widely varying sizes, shapes and densities is another concern. A solution is provided for this problem. Categorical datasets are generally high dimensional. Clustering high dimensional datasets containing a large number of records is a harder problem, since cluster may be determined by subsets of attributes. A subspace-based technique is proposed for clustering large high dimensional categorical datasets. The problem becomes compounded when the dataset contains a mixture of categorical and numeric attributes. The categorical data clustering algorithm is extended for clustering datasets with mixed categorical and numeric attributes, which is one of our major contributions.

Gene expression data analysis is one of the active fields of research where clustering techniques need to be applied. It requires a specialized type of algorithm known as biclustering. We have proposed an efficient algorithm for

biclustering gene expression data. Network intrusion detection is another area where clustering techniques can be applied. A network intrusion detection technique is also proposed based on one of our clustering algorithms.

Altogether we have developed seven algorithms, which are validated by experimental results.

10.1 Directions for future works

- The query sampling procedure applied in *IDBSCAN* can be used to sample the dataset first followed by applying *DBSCAN* to the reduced dataset.
- The *DDSC* algorithm for detecting clusters with widely varying densities can be extended so that it detects subspace clusters in high dimensional spatial data. Parallel implementation of the algorithm can be considered.
- The *CatSub* algorithm can be made more efficient by employing a tree data structure instead of linear lists of clusters.
- using a density-based approach for mixed-type data clustering technique may be more beneficial.
- The gene expression data clustering technique can be modified to detect several biclusters at a time. Node deletion can also be considered along with node addition. Biological significance analysis of the extracted clusters will be an important task.
- The assumptions made in the network intrusion detection algorithm produced promising results for KDD CUP Corrected dataset. To validate the assumptions more extensive study may be performed with other sources of data including data collected from real networks.

Bibliography

- [AAW04] B Andreopoulos, A An, and X Wang. Mulic : Multi-layer increasing coherence clustering of categorical data sets. Technical report, York University, 2004.
- [ABKS99] M Ankerst, M Breunig, H-P Kriegel, and J Sander. Optics: Ordering objects to identify the clustering structure, proc. acm sigmod. In *International Conference on Management of Data*, pages 49–60, 1999.
- [And73] M R Anderberg. *Cluster Analysis for Applications*. Academic Press, New York, 1973.
- [And02] P Andritsos. Data clustering techniques. Technical report, University of Toronto, Department of Computer Science, Mar 2002.
- [Ash90] R B Ash. *Information Theory*. Dover Publications, Inc, New York, 1990.
- [ATMS04] P Andritsos, P Tsaparas, R J Miller, and K C Sevcik. Limbo : Scalable clustering of categorical data. In *Proceedings of the 9th International Conference on Extending Database Technology(EDBT)*, Mar 2004.

- [BCL02] D Barbara, J Couto, and Y Li. Coolcat: An entropy-based algorithm for categorical clustering. In *Proceedings of the 11th ACM CIKM Conference*, pages 582–589, 2002.
- [BKSS90] N Beckmann, H-P Kriegel, R Schneider, and B Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of ACM-SIGMOD International Conference On Management of Data (SIGMOD90)*, pages 322–331, 1990.
- [BM98] C Blage and C Merz. Uci repository of machine learning databases. <http://www.ics.uci.edu/mllearn/MLRepository.html>, 1998.
- [BM01] R Bace and P Mell. Intrusion detection systems. In *NIST Special Publications SP 800-31*, Nov 2001.
- [BNT04] K Burbeck and S Nadjm-Tehrani. Advice - anomaly detection with real-time incremental clustering. In *Information Security and Cryptology - ICISC 2004, Lecture Notes on Computer Science, Volume 3506/2005*, pages 407–424, 2004.
- [CC00] Y Cheng and G M Church. Biclustering of expression data. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology (ISMB'00)*, pages 93–103, 2000.
- [DM00] I S Dhillon and D S Modha. A data-clustering algorithm on distributed memory multiprocessors. In *Large Scale Parallel Data mining, Lecture Notes in Artificial Intelligence, 1759*, pages 245–260, 2000.
- [EKSX96] M Ester, H-P Kriegel, J Sander, and X Xu. A density-based algorithm for discovering clusters in large spatial data sets with noise. In *2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

- [ESK03] L Ertöz, M Steinbach, and V Kumar. Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *Proceedings of Second SIAM International Conference on Data Mining*, January 2003.
- [FLPT00] D Foti, D Lipari, C Pizzuti, and D Talia. Scalable parallel clustering for data mining on multicomputers. In *15 IPDPS 2000 workshops*, pages 390–398, 2000.
- [GGB03] Y Guan, A Ghorbani, and N Belacel. Y-means: A clustering method for intrusion detection. In *Proceedings of the Canadian Conference on Electrical and Computer Engineering, Montreal, Quebec, Canada*, May 2003.
- [GGR99] J Gehrke, V Ganti, and R Ramakrishnan. Cactus: Clustering categorical data using summaries. In *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining(KDD '99)*, pages 73–83, 1999.
- [GKR88] D Gibson, J Kleinberg, and P Raghavan. Clustering categorical data : An approach based on dynamical systems. In *Proceedings of 25th International Conference on Very Large Databases*, 1988.
- [gla] <http://www.glaros.dtc.umn.edu/gkhome/cluto/cluto/download>.
- [GW04] G Gan and J Wu. Subspace clustering for high dimensional categorical data. *ACM SIGKDD Explorations Newsletter*, 6(2):87–94, 2004.
- [GWY06] G Gan, J Wu, and Z Yang. Partcat : A subspace clustering algorithm for high dimensional categorical data. In *Proceedings of 2006 International Joint Conference on Neural Networks, Sheraton*

Vancouver Wall Centre Hotel, Vancouver, BC, Canada, pages 4406–4412, July 2006.

- [Har72] J A Hartigan. Direct clustering of data matrix. *Journal of American Statistical Association*, 67(337):123–129, 1972.
- [HK98] A Hinneburg and D Keim. An efficient approach to clustering in large multimedia data sets with noise. In *4th International Conference on Knowledge Discovery and Data Mining*, pages 58–65, 1998.
- [HK06] J Han and M Kamber. *Data Mining Concepts and Techniques*. Morgan Kaufman, 2006.
- [HMS04] D Hand, H Mannila, and P Smyth. *Principles of Data Mining*. Prentice-Hall of India, New Delhi, 2004.
- [Hua98] Z Huang. Extensions to the k -means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, 2(2):283–304, 1998.
- [HXd] Z He, X Xu, and S deng. Clustering mixed numerical and categorical data : A cluster ensemble approach. <http://arxiv.org/ftp/cs/papers/0509/0509011.pdf>.
- [HXD02] Z He, X Xu, and S Deng. Squeezer : An efficient algorithm for clustering categorical data. *Journal of Computer Science and Technology*, 17(5):611–624, 2002.
- [JK00] E K Johnson and H Kargupta. Collective, hierarchical clustering from distributed, heterogeneous data. In *Large Scale Parallel Data Mining, LNCS 1759*, 2000.

- [KBCG03] Y Kluger, R Basri, J T Chang, and M Gerstein. Spectral biclustering of microarray data : coclustering genes and conditions. *Genome Research*, 13(4):703–716, 2003.
- [KHK99] G Karypis, E H Han, and V Kumar. Chameleon: A hierarchical clustering algorithm using dynamic modeling. *Computer*, 32(8):68–75, 1999.
- [LEK⁺03] A Lazarevic, L Ertöz, V Kumar, A Ozgur, and J Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *SIAM International Conference on Data Mining*, 2003.
- [LH03] S Q Le and T B Ho. A k-sets clustering algorithm for categorical and mixed data. In *Proceedings of the 6th SANKEN (ISIR) International Symposium*, pages 124–128, 2003.
- [LL05] K Leung and C Leckie. Unsupervised anomaly detection in network intrusion detection using clusters. In *Proceedings of the Twenty-eight Australasian Conference on Computer Science, Newcastle, Australia*, pages 333–342, 2005.
- [LMO04] T Li, S Ma, and M Ogihara. Entropy-based criterion in categorical clustering. In *Proceedings of the 21st International Conference on Machine Learning*, 2004.
- [LO02] L Lazzeroni and A Owen. Plaid models for gene expression data. *Statistica Sinica*, 12(1):61–86, 2002.
- [MO04] S C Manderia and A L Oliveira. Biclustering algorithms for biological data analysis : a survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):24–45, 2004.

- [Ols95] C F Olson. Parallel algorithms for hierarchical clustering. In *Parallel Computing*, 21, pages 1313–1325, 1995.
- [PAOC06] S Petrovic, G Alvarez, A Orfila, and J Carbo. Lebellng clusters in an intrusion detection system using a combination of clustering evaluation texchniques. In *Proceedings of the 39th Hawaii International Conference on System Sciences*, 2006.
- [PES01] L Portnoy, E Eskin, and S Stolfo. Intrusion detection with unlabeled data using clustering. In *Proceedings of ACM CSS Workshop on Data Mining Applied to Security(DMSA-2001)*, Nov 2001.
- [PHL04] L Parsons, E Haque, and H Liu. Subspace clustering for high dimensional data: A review. *ACM SIGKDD Explorations Newsletter*, 6(1):90–105, 2004.
- [PRB05] R G Pensa, C Robardet, and J-F Boulicaut. A biclustering framework for categorical data. In *A. Jorge et al.(Eds.): PAKDD 2005, LNAI 3721*, pages 645–650, 2005.
- [PZ04] M Peters and M J Zaki. Click : Clustering categorical data using k-partite maximal cliques. Technical report, CS Department, RPI, 2004.
- [RGS99] R Rastogi, S Guha, and K Shim. Rock: A robust clustering algorithm for categorical attributes. In *Proceedings of the 15th International Conference on Data Engineering*, 1999.
- [SB99] K Stoffel and A Belkoniene. Parallel k-means clustering for large data sets. In *Proceedings Euro-Par99, LNCS 1685*, pages 1451–1454, 1999.
- [TSK06] P-N Tan, M Steinbach, and V Kumar. *Introduction to Data Mining*. Pearson Addison Wesley, 2006.

- [UoC99] Irvine University of California. Kdd cup 1999 data. <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>, 1999.
- [WA03] B Wilkinson and M Allen. *Parallel Programming Techniques and Applications*. Pearson Education, Second Indian Reprint, 2003.
- [Wei] E W Weisstein. Sphere-sphere intersection. <http://www.mathworld.wolfram.com/Circle-CircleIntersection.html>. From MathWorld – A Wolfram Web Resource.
- [XJK99] X Xu, J Jger, and H P Kriegel. A fast parallel clustering algorithm for large spatial databases. In *Data Mining and Knowledge Discovery*, 3, pages 263–290, 1999.
- [YTRC05] J Yin, Z Tan, J Ren, and Y Chen. An efficient clustering algorithm for mixed type attributes in large dataset. In *Proceedings of the 4th International Conference on Machine Learning and Cybernetics*, pages 18–21, Aug 2005.
- [YWWY03] J Yang, H Wang, W Wang, and P S Yu. Enhanced biclustering of expression data. In *proceedings of the 3rd IEEE Symposium on Bioinformatics and Bioengineering (BIBE'03)*, pages 321–327, Mar 2003.
- [Zak00] Mohammed J Zaki. Parallel and distributed data mining. In *Large Scale Parallel Data Mining, Lecture Notes in Artificial Intelligence*, 1759, pages 1–23, 2000.
- [ZSP03] B Zhou, J-Y Shen, and Q-K Peng. Parcle : A parallel clustering algorithm for cluster system. In *Proceedings of 2nd International Conference on Machine Learning and Cybernetics*, 2003.

- [ZTOT04] Z Zhang, A Teo, B C Ooi, and K-L Tan. Mining deterministic biclusters in gene expression data. In *Proceedings of the Fourth IEEE Symposium on Bioinformatics and Bioengineering (BIBE'04)*, 2004.
- [ZZH00] S Zhou, A Zhou, and Y Hu. Combining sampling technique with dbscan algorithm for clustering large spatial databases. In *Knowledge Discovery and Data Mining : Current Issues and New Applications, PAKDD2000*, Apr 2000.

List of Publications:

1. B Borah and D K Bhattacharyya. An Improved Sampling-Based DBSCAN for Large Spatial Databases. In proceedings of International Conference on Intelligent Sensing and Information Processing(ICISIP-2004), January, 2004, pages 92–96.
2. B Borah and R K Das and D K Bhattacharyya. A Parallelization of Density-Based Clustering Technique on A Distributed Memory Multicomputer. In proceedings of the 12-th International Conference on Advanced Computing and Communications(ADCOM-2004), December, 2004, pages 536–541.
3. B Borah and D K Bhattacharyya. Image Retrieval by Content using Segmentation Approach. In proceedings of First International Conference on Pattern Recognition and Machine Intelligence(PReMI 2005), Kolkata, December, 2005, pages 551–556.
4. B Borah and D K Bhattacharyya. An Entropy Based Hierarchical Clustering Algorithm for Categorical and Mixed Type Data. In proceedings of National Workshop in Networks, Data Mining and Artificial Intelligence Trends and Future Directions(NWTAC 2006), January, 2006, Tezpur University, pages 153–162.
5. B Borah and D K Bhattacharyya. A Clustering Technique Using Density Difference. In proceedings of International Conference on Signal Processing, Communications and Networking(ICSCN-2007), March, 2007, pages 585–588.
6. B Borah and D K Bhattacharyya. SCUDD: Spatial Clustering Using Density Difference. In proceedings of National Conference on Trends in Advanced Computing(NCTAC-2007), March, 2007, Tezpur University, pages 138–146.

7. B Borah and D K Bhattacharyya. Biclustering Expression Data Using Node Addition Algorithm. In proceedings of the 15-th International Conference on Advanced Computing and Communications(ADCOM-2007), December, 2007, pages 307–312.
8. B Borah and D K Bhattacharyya. DDSC: A Density Differentiated Spatial Clustering Technique. Journal of Computers, Vol. 3, No. 2, February, 2008, pages 72–79.
9. B Borah and D K Bhattacharyya. CatSub: Clustering Categorical Data Based on Subspace, The Icfai Journal of Computer Sciences, Vol. 2, No. 2, April, 2008, pages 7–20.
10. B Borah and D K Bhattacharyya. SMIC: A Subspace Preferred Mixed Type Data Clustering Technique, International Journal of Reliability, Quality and Safety Engineering(submitted).