High Performance Subspace Clustering for Massive Data Sets

by

Harsha S Nagesh
Electrical and Computer Engineering

THESIS

Submitted in partial fulfillment of the requirements

for the degree of Master of Science in Graduate School

in the Northwestern University of

1999, Evanston, Illinois

Abstract

Business establishments collect vast amounts of data every day. Leveraging this data for smart decision making is the key to identifying profit opportunities, customer retention and giving a winning touch to the business. The path from large amounts of data to Knowledge Discovery is Information Mining, using a sophisticated set of tools to uncover associations, patterns, and trends; detect deviations; cluster and classify information; and develop predictive models. With the increase in the size of databases parallel processing techniques need to be applied to empower knowledge discovery tools to *dig* information from these data sets and reduce the time for analysis.

In this thesis we focus on clustering techniques for large scale data sets. Clustering is the process of identifying dense regions in a sparse multi-dimensional data set. We have designed and implemented a density and grid based clustering algorithm wherein a multi-dimensional space is divided into finer grids and the dense regions found are merged together to identify the clusters. For large data sets with a large number of dimensions fine division of the multi-dimensional space leads to an enormous amount of computation. We have introduced an adaptive grid framework which not only reduces the computation vastly by forming grids based on the data distribution, but also improves the quality of clustering. Clustering algorithms also need to explore clusters in a subspace of the total data space. We have implemented a new bottom up algorithm which explores all possible subspaces to identify the embedded clusters. Traditional clustering algorithms require several user inputs which are not only hard to determine but are also not practical. We have introduced a framework which requires no user input, making our algorithm, **pMAFIA**, a completely unsupervised data mining algorithm. Finally we have introduced parallelism in the clustering process, which enables our data mining tool to scale up to massive data sets.

Acknowledgements

I would like to first thank Alok Choudhary without whose constant guidance, support and encouragement, this work would not have been possible. I would also like to thank Sanjay Goil who gladly discussed various issues related to my work with me. This work is the result of many insightful discussions that I have had with Alok Choudhary who inspired me all through and also guided me as and when required. I would also like to thank my committee members Prithviraj Banerjee and Peter Scheuermann for their help and guidance.

Further, I would like to thank Venkatesh Karnam and Pramod Joisha for their suggestions and comments. Finally I would like to dedicate this thesis to my parents Nagesh K. and Annapurna S. who have guided me all through my life. I would like to thank them for all the love, encouragement and virtues that I have received while I was growing.

Contents

1 Introduction **1**

1.1 Motivation 1

1.1.1 Data mining techniques 2

1.2 Contributions 4

1.3 Organization 5

2 Related Work **7**

2.1 Clustering: As a Data Mining Tool 8

2.1.1 Hierarchical Clustering 9

2.1.2 Partition based Clustering 10

2.1.3 Categorical Clustering 10

2.1.4 Density based Clustering 11

2.1.5 Parallel Clustering 12

3 Density and Grid based Clustering **13**

3.1 Subspace Clustering 13

3.1.1 Subspace Clustering in CLIQUE 14

3.1.2 Analysis 16

3.1.2.1 pMAFIA: Candidate Dense Unit Generation Process 16

3.1.2.2 CLIQUE: Candidate Dense Unit Generation Process 17

3.2 Adaptive Grids 19

3.2.1 Effect of Grids on Cluster Quality 21

4 pMAFIA: Parallel Subspace Clustering 24

4.1 Algorithm 25

4.1.1 Data Parallelism 27

4.1.2 Task Parallelism 27

4.1.2.1 Building Candidate Dense Units 28

4.1.2.2 Identification of Dense Units and Building Data Structures . . . 31

4.2 Analysis 35

5 Performance Experiments 36

5.1 Synthetic Data Generation 36

5.2 Experimental Results 38

5.2.1 Synthetic Data Results 38

5.2.1.1 Effect of using Adaptive Grids 38

5.2.1.2 Effect of Adaptive Grids on Computational Complexity 40

5.2.1.3 Parallel Performance 41

5.2.1.4 Scalability with Database Size 42

5.2.1.5 Scalability with Dimensionality of Data and Cluster Dimensionality 42

5.2.1.6 Quality of Results 43

5.2.2 Real Data Sets 45

5.2.2.1 One Day Ahead Prediction of DAX 45

5.2.2.2 Ionosphere Data 46

5.2.2.3 EachMovie Recommendation Data 46

6 Summary and Future Work **48**

6.1 Conclusion 48

6.2 Future Work 49

References **50**

List of Figures

3.1 Building A Candidate Dense Unit. (Current Dimension, k = 4) 20
3.2 (a) Uniform grid size (b) Adaptive grid size 22
3.3 (a) Cluster discovered by CLIQUE (b) Cluster discovered by MAFIA 23

4.1 A shared-nothing architecture, P:Processor, M:Memory, D:Disk 25
4.2 Building Candidate Dense Units. (Current Dimension, k = 3) 28
4.3 Parallel Building of Candidate Dense Units 29

5.1 (a) Speedup of pMAFIA over CLIQUE (b) Comparative execution times (in seconds) for 300,000 records in a 15 dimensional space with 1 cluster in 5 dimensions 39
5.2 (a) Parallel run times of pMAFIA (b) Speedup observed 41
5.3 Scalability with increasing database size 42
5.4 Scalability with increasing (a) Data Dimensionality (b) Cluster Dimensionality 43

List of Tables

5.1 Candidate Dense Units generated by pMAFIA and CLIQUE 40
5.2 Comparison of the quality of results obtained by pMAFIA and CLIQUE . . . 44
5.3 Clusters Discovered in the DAX Data Set 46
5.4 Parallel Performance on the Movie Data Set. 47

Chapter 1

Introduction

Large amounts of data are generated every day in business and scientific domains. Sophisticated tools that can intelligently and automatically transform data into information adds value to the business and gives a competitive advantage. The process of Data Mining as one pundit characterized is, the potential to transform the zeroes and ones of business data into the ones and several trailing zeroes of dollars in large profits.

1.1 Motivation

Data Mining, also referred as *knowledge discovery in databases*, is the nontrivial process of extraction of implicit, previously unknown and potentially useful information from data in databases [FPSSU94]. Data mining is a multi disciplinary field with applications of techniques ranging from artificial intelligence [MS83], pattern recognition [Fuk90] to statistics [AA96] and databases [CHY96]. Applications of data mining include the broad areas of telecommunications, financial services and insurance, health-care, database marketing, scientific knowledge discovery, etc. Data mining techniques help us find answers to questions on data residing in conventional databases which database queries fail to produce. Financial business establishments will be able to find what debt pattern characterizes pending bankruptcy, chain stores will be able to find what kind of customers reject mail solicitations, health-care industry would be able to identify fraudulent billing patterns, online stores will be able to characterize the web surfing patterns which actually translate to revenue, etc.

1.1.1 Data mining techniques

Various classification schemes can be used to categorize data mining techniques. Schemes based on the kinds of databases used, based on the kind of information that needs to be mined and based on the techniques used are found in the literature. Application of specific data mining techniques greatly depends on the kind of data we have, the questions that we need to answer and also on the computation power at hand.

Mining *association rules* in relational databases [AIS93] is a data mining technique which derives a set of strong association rules, in a given set of transactions, of the form $X \Rightarrow Y$, where X and Y are sets of items and each transaction is a set of literals (called items). For example, from a large set of transaction data one may find an association rule such as a customer who purchased milk of a particular brand also purchased bread of another particular brand. Mining association rules involve a large number of repeated passes over the data to find different association patterns, thus requiring a large amount of processing power and efficient mining algorithms. A rule $X \Rightarrow Y$ holds in the transaction set D with *confidence* c if $c\%$ of transactions in D contain X also contain Y. The rule $X \Rightarrow Y$ has *support* s in the transaction set D if $s\%$ of transactions in D contain $X \cup Y$. Confidence is a measure of the strength of implication and support indicates the frequencies of occurring patterns in the rule. Rules with high confidence and strong support are often more interesting and are referred to as *strong rules* [AIS93]. The Apriori algorithm [AS94a] and several modifications of this work have explored mining association rules in greater depth.

Another technique that has found several applications is the process of *data classification*. Classification is the process of finding common properties among a set of objects in a database and classifying them into various *classes* based on a classification scheme. Classification models are first trained on a training data set which is representative of the real data set. The training data is used to evolve classification rules for each class such that they best capture the features and traits of each class. Rules evolved on the training data are applied to the main database and data is partitioned into classes based on the rules. Also classification rules are modified as we add data incrementally over time. Data classification techniques are widely applied in the areas of selective marketing, medical analysis, information retrieval to mention a few. Several classification techniques found in the literature are based on neural networks [MST94] , genetic algorithms [Gol89], decision trees [BFOS84] and other methods.

Sequential pattern mining [AS95] is a technique to detect patterns in a series of transactions. Each data sequence is a list of transactions, where each transaction is a set of items (literals). This technique is used to find all sequential patterns which have a strong support and occur a significant many times in the database. This analysis helps to discover buying patterns such as people buying a set of books in a particular order, events that occur more often in a particular order, etc. Time-Series clustering [ALSS95] is another data mining technique used to detect similarities in different time series. Applications of time-series clustering include discovering stocks with similar price movements, determining portions of seismic waves that are not similar to spot geological irregularities, identifying corporations with similar growth patterns, etc.

The process of grouping physical or abstract objects into classes or regions with similar properties is clustering. Clustering has long been studied as part of statistics [AA96], image processing [Fuk90], etc and is a field with a great amount of work done. The process of clustering identifies densely populated regions or clusters in a large, sparsely populated, multi-dimensional space. Clustering has traditionally been considered as a problem of partitioning the data space with applications in the areas of physical circuit design, data compression, customer segmentation, etc in mind. Most of the clustering algorithms in literature require several user inputs to control the quality of clustering. These inputs not only influence the end results greatly but are also very hard to determine. Further, clustering algorithms need to make several passes over large amounts of data to scan and cluster all the points. We shall look into clustering as a data mining technique in a much greater detail in the remaining part of this thesis.

Most of the data mining algorithms are extremely compute intensive. These techniques need to examine massive amounts of data of the order of several giga bytes and perform several computations over them in order to extract useful information. Further, information needs to be extracted in real time without disturbing the state of the transactional database systems. Parallel processing adds the essential processing power to analyze such vast amounts of data. Several incremental data mining algorithms are being designed to mine data that is incrementally generated as in sales records in a consumer market, etc. Data mining techniques can be applied to the incremental data gathered periodically and patterns obtained incrementally could be merged with existing patterns in order to obtain the evolving knowledge embedded in the database.

1.2 Contributions

In this thesis we present a novel clustering algorithm, **pMAFIA**, for massive data sets. pMAFIA is a grid and density based disk-based clustering algorithm. In a density and grid based clustering approach a large multi-dimensional data space is divided into fine grids forming a large number of hyper-rectangular regions in the multi-dimensional space. A histogram count of the number of points in each hyper-rectangular region is found and regions which have a count greater than a specified threshold are identified as *dense*. Dense regions found are merged with other connected dense regions and cluster regions are identified. For large data sets, with a large number of dimensions and large number of records, dividing the multi-dimensional space into a fine grid results in a huge number of regions. Also with the increase in the database size this approach will no longer be scalable with the database size and data dimensionality. Further, fine grids lead to a lot of computation and noise regions also have a potential to be combined into cluster regions, while coarse grids fail to capture the clusters correctly and result in a great loss of quality. We have introduced the framework of adaptive grids where in we divide the multi-dimensional space into an optimal set of regions based on the data distribution. This way we not only minimize the number of regions generated and hence the computation requirements, but also greatly improve in the quality of clustering as we capture the data distribution precisely. pMAFIA also does not require any user inputs making it a completely unsupervised data mining algorithm. Once the grids are automatically determined we compute the required thresholds in each region and determine dense regions using these thresholds. Clusters may also be embedded in a subspace of the total data space. This facet of the clustering problem demands that all possible subspaces of the total data space need to be explored in order to discover all the clusters. However the number of subspaces for a given dimensionality of the data set is exponential in the data dimension. In order to explore all these subspaces we have developed a new bottom up algorithm, a modification of an existing scheme, which builds clusters of lower dimensionality and gradually merge them to form higher dimensional clusters. Merging of lower dimensional clusters to form higher dimensional clusters requires multiple passes over the data set. This implies that for massive data sets the computational requirements also grow rapidly. The increase in the number of clusters in each dimension further adds to the complexity of the computation and hence parallel processing provides the much needed computational muscle for pMAFIA to be a practical clustering algorithm for real world problems. We incorporate both task and data parallelism and have achieved near liner speedups with negligible communication overheads in our implementation on the IBM SP2 machine.

1.3 Organization

The remaining part of the thesis is organized as follows. Chapter 2 explores in great depth work done in the area of clustering. Chapter 3 elaborates on the grid and density based clustering. We also introduce the problem of subspace clustering and describe our pMAFIA, the bottom up algorithm for building higher dimensional clusters by merging clusters in lower dimensions. In chapter 4 details of the parallel algorithm are explained. A theoretical analysis of the algorithm is also performed. Chapter 5 presents the results on a wide variety of synthetic benchmarks as well as real world data sets. Results on data sets with a varying number of dimensions and data set sizes are presented. To the best of our knowledge this is the first practical parallel subspace clustering algorithm which is scalable to data sets with a dimensionality of more than 50 and also database size of more than 12 million records in such high dimensional sizes. Finally chapter 6 concludes the thesis with future work and a summary of contributions.

Chapter 2

Related Work

Clustering problems have been studied in various disciplines with applications ranging from feature recognition, data compression, trend analysis, political change analysis [SG96], data mining, etc. Traditionally clustering has been studied as a problem of grouping a given set S of n objects into several groups of objects such that objects within the same group are more similar as compared to objects in the other clusters. One of the earliest survey in this field is by Duran and Odell [DO74]. Clustering is covered more as a geometric problem and solutions based on enumeration are discussed. An excellent survey of geometric clustering algorithms has been compiled by Procopiuc [Pro96]. Applications of clustering algorithms in the field of graph theory and physical circuit design tend to look clustering as more of a partitioning problem. In two dimensions, optimal clustering of a given set of points has been proved to be NP-complete [MS84]. Vector quantization is another related approach in the field of data compression and speech processing where in we try to map a state space S to a countable set of vectors A using a mapping function, q, called the quantizer. Clustering techniques are used to find the elements of A, also known as the code book, such that it minimizes the quantization noise, $D(q)$. Most of the clustering algorithms based on computational geometry approach work well for lower dimension data and also when the the database size is limited. As the dimensionality of the datasets increase the search space explodes and conventional clustering algorithms break down. The problem of high dimensionality has been either addressed by using user-defined subset of dimensions to apply the clustering algorithm to, or use a dimensionality reduction method. The former is not very effective for using data mining as an unsupervised learning technique. Also selecting a subset of the total dimensions is a non-trivial process. Dimensionality reduction techniques such as principal component analysis and Karhunen-Loève transformation [Fuk90], transform the original data space into a lower dimensional space by forming dimensions which are linear combinations of the given dimensions. However, the effectiveness of clustering in this transformed space is limited by the difficulty in interpreting the clusters in relation to the original data space. Dimensionality reduction techniques, however, have been quite effectively used in the field of document clustering and information retrieval. *Latent Semantic Indexing* is a technique which has a vector space representation of a set of documents and the terms frequently occurring in the documents. It further uses singular value decomposition to project the vectors to a new lower dimensional space. A good reference for intelligent information retrieval using liner algebra is [BDO95].

Clustering has received great interest in the AI community and also in the field of statistics [AA96]. An important approach in this direction has been the problem of expectation maximization. Probabilistic modeling approaches and Bayesian classification techniques try to find the best fit probability density function for a given set of data points. However these approaches require key input parameters which are hard to determine.

2.1 Clustering: As a Data Mining Tool

As a data mining task clustering algorithms discover densely populated regions translating to some useful patterns in the data set. Clustering algorithms from a database perspective need to address several issues. The presence of data distributed in a large multi-dimensional space, due to a large number of attributes makes it difficult to detect clusters. Noise in data owing to uniform distributions in some data dimensions further complicate the cluster detection process. Further, clustering algorithms need to be insensitive to the order of input data points.

Cluster descriptions of the data have to be used by the end-user to leverage the benefits of data mining. Simplicity in reporting clusters among those found improves the usability of the algorithm. This can help to interpret the results better and lead to the expected value-adding that data mining provides to the end-user. Scalability of the algorithm with data sets and the number of dimensions is another important feature for current applications of clustering. These algorithms are to be applied to large repositories of business data (data warehouses) and large scale scientific data (satellite images, large scientific simulations) to be effective. Further, clusters can be embedded in some lower dimensional space, a subspace of the total data space. This results in an explosion of the search space for the clusters as the number of subspaces is exponential in the dimensionality of the data set.

2.1.1 Hierarchical Clustering

Existing clustering algorithms can be classified into *hierarchical* and *partitioning* algorithms. A hierarchical method uses a nested sequence of partitions. This can either be done by starting with each object in its own cluster and then merging these to form larger clusters, or all objects are considered in one cluster and the process starts by subdividing it into smaller clusters [JD88]. Hierarchical clustering has been used in the single-link method, which starts with placing each object in its own cluster and in every step the two closest clusters are merged until all points are in one cluster [Sib73]. Recently, CURE [GRS98] has been proposed which uses the hierarchical clustering method. Creation of cluster hierarchy is stopped if a level consists of k clusters, where k is one of several input parameters. Distance between clusters is evaluated by multiple representative points which is well-suited for arbitrary shaped clusters. CHAMELEON is a recently proposed hierarchical clustering algorithm which performs well for arbitrary shaped clusters in 2 dimensional spaces [KHK99].

2.1.2 Partition based Clustering

Partitioning algorithms construct a partition of the objects in the database into clusters such that objects in a cluster are more similar to each other than to the objects in different clusters. k-means and k-medoid methods determine k cluster representatives and assign each object to the cluster with its representative closest to the object such that the sum of the distances squared between the objects and their representatives is minimized. The k-modes algorithm extends the k-means paradigm to categorical domains [Hua97]. CLARANS [NH94], BIRCH [ZRL96] can be considered to be extensions of partition based methods for databases. The clustering process in [NH94] is analogous to searching a graph for finding an optimum value of a cost function, wherein each node is a potential solution (a set of k mediods). BIRCH (Balanced Iterative Reducing and Clustering) is a clustering algorithm with emphasis for out of core data sets. Their approach introduced the concept of a *CF tree* (Clustering Feature), a balanced tree, whose maximum number of children can be controlled by a branching factor, B. Also the maximum diameter of sub-clusters at each leaf-node can be controlled by a threshold parameter. The CF tree built dynamically as the data points are inserted, ensures that a balanced tree is formed. The I/O costs of the BIRCH algorithm have been shown to be $O(N)$ and the algorithm also produces good quality of clustering with insensitivity to the input order of points as reported in [ZRL96]

2.1.3 Categorical Clustering

Clustering categorical data often needs a different approach as compared to clustering numerical data points. This difference is due to the absence of any inherent ordering among the values of any particular attribute. For example consider the sales transaction of an automobile company. Each transaction contains complete information of every sales transaction like the model of the car, color of the car, brand of the car, etc. Each attribute can take several attribute values. The model of a car can be either Toyota, Mercedez, Audi or other brands and similarly each car sold could take any color from among the set of available colors. An interesting information that can be mined from this data is a particular clustering of customer preferences for any given model, color, brand, etc. However, since the attribute values do not have any imposed order on them, distance based clustering techniques fail to be applicable. k-modes [Hua97] is one of the earlier works in categorical clustering. STIRR [GKR98] is a categorical clustering algorithm which applies the generalized spectral partitioning technique to the problem of hyper-graph clustering. The categorical data set is modeled as a hyper-graph and a clustering method based on non-linear dynamical systems is applied to the hyper-graph. A hyper-graph partitioning algorithm is used to cluster a categorical data set in [HKKM97]. ROCK [GRS99] is a categorical clustering algorithm which clusters a sampled data set using a novel concept called the *links* between the data points. Data points are considered as *neighbors* if their similarity exceeds a threshold and the number of links between two data points is the number of common neighbors that they share. CACTUS [GGR99] is a recent summarization-based clustering algorithm. It has minimum I/O requirements and is based on finding inter-attribute and intra-attribute summaries.

2.1.4 Density based Clustering

DBSCAN is a density based clustering algorithm [EKSX96]. For each object of a cluster the neighborhood in a given radius, r, has to contain at least a minimum number of points, ϵ. Here both the radius and minimum number of points are input parameters which can significantly affect the quality of clustering. However, clusters of arbitrary shape can be discovered in this approach and is well suited for spatial databases. DBCLASD a distribution based clustering clustering algorithm [XEKS98], is based on the assumption that points inside a cluster are uniformly distributed. DBCLASD does not require any user input, but however has a lower efficiency than the DBSCAN algorithm. OPTICS is a recent density based clustering algorithm [ABKS99]. OPTICS however does not produce a clustering of a data set explicitly, but instead creates an augmented ordering of the database representing its density-based clustering structure.

2.1.5 Parallel Clustering

A survey of parallel algorithms for hierarchical clustering using distance based metrics is given in [Ols95]. These are more theoretical, PRAM algorithms, now more of academic interest than practical parallel algorithms for large data sets. Recently, k-means algorithm has been parallelized [DM99], but is limited however in its applicability, as it requires the user to specify k, the number of clusters, and also does not find clusters in subspaces. To the best of our knowledge, pMAFIA is one of the first efforts in practical parallel clustering techniques for large data sets.

Chapter 3

Density and Grid based Clustering

Density based clustering approaches apply a local cluster criterion, in which clusters are regarded as regions in the data space in which the objects are dense, and which are separated by regions of low object density (noise). A common way to find regions of high-density in the data space is based on grid cell densities [JD88]. A histogram is constructed by partitioning the data space into a number of non-overlapping regions or cells. Cells containing a relatively large number of objects are potential cluster centers and the boundaries between clusters fall in the "valleys" of the histogram. The size of the cells determines the computations and the quality of the clustering. Cells of small volume will give a very "noisy" estimate of the density, whereas large cells tend to overly smooth the density estimate. WaveCluster [SCZ98] is a density and grid based approach which applies wavelet transforms to the feature space. It is computationally very efficient but is applicable to only low dimensional data. WaveCluster also finds cluster description at different resolution levels by applying the multi-resolution wavelet transform. Further, it requires an involved post-processing step in order to describe the found clusters.

3.1 Subspace Clustering

None of the clustering algorithms described so far find clusters in a subspace of the total data space. Clusters may be embedded in a lower dimensional space compared to the data space.

The number of subspaces is exponential in the data set dimensionality. No obvious extension of existing algorithms can explore subspace clusters. A brute force algorithm to explore clusters in all possible subspaces breaks down with the increase in the data set dimensionality as is the case with real world data sets.

PROCLUS [APW$^+$99], a subspace clustering algorithm finds representative cluster centers in an appropriate set of cluster dimensions. It needs the number of clusters, k, and the average cluster dimensionality, l, as input parameters, both of which are not possible to be known apriori for real data sets. ENCLUS [CFZ99], an entropy based subspace clustering algorithm requires a prohibitive amount of time to just discover interesting subspaces in which clusters are embedded. Once the subspaces are discovered any clustering algorithm is applied to discover the actual clusters. It also requires input of entropy thresholds which is not intuitive for the user.

3.1.1 Subspace Clustering in CLIQUE

In a grid and density based approach the grid size heavily determines the computations and the quality of the clustering. CLIQUE, a density and grid based approach for high dimensional data sets [AGGR98], detects clusters in the highest dimensional subspaces. Density based approaches regard clusters as high density regions than their surroundings. A common way of finding high-density regions in the data space is based on the grid cell densities [JD88]. CLIQUE takes the size of the grid and a global density threshold for clusters as input parameters. The computation complexity and the quality of clustering is heavily dependent on these parameters. A histogram is constructed by partitioning the data space into a number of non-overlapping regions and then mapping the data points to each cell in the grid. Equal length intervals are used in [AGGR98] to partition each dimension, which results in uniform volume cells. The number of points inside the cell with respect to the volume of the cell can be used to determine the density of the cell.

Clusters are unions of connected high density cells. Two k-dimensional cells are connected if they have a common face in the k-dimensional space or if they are connected by a common cell. Creating a histogram that counts the points contained in each unit is infeasible in high dimensional data. Subspace clustering further complicates the problem as it results in an explosion of such units as the number of subspaces is exponential in the dimension of the data set. A bottom-up approach of finding dense units and merging them to find dense clusters in higher dimensional subspaces has been proposed in CLIQUE [AGGR98]. This algorithm is similar to the Apriori algorithm [AS94b] used in mining association rules. It exploits the monotonicity of the clustering criterion with respect to dimensionality to restrict the search space. Formally, it states that "If a collection of points S is a cluster in a k-dimensional space, then S is also a part of a cluster in any $(k-1)$-dimensional projection of the space" [AGGR98]. Each dimension is divided into a user specified number of intervals, ϵ. The algorithm starts by determining 1-dimensional dense units by making a pass over the data. In [AGGR98] candidate dense cells in any k dimensions are obtained by merging the dense cells in $(k-1)$ dimensions which share the *first* $(k-2)$ dimensions. A pass over the data is made to find which of the candidate dense cells are actually dense. The algorithm terminates when no more candidate dense cells are generated. In [AGGR98] candidate dense units are pruned based on a minimum description length technique to find the dense units only in *interesting* subspaces. However, as noted in [AGGR98] this could result in missing some dense units in the pruned subspaces. In order to maintain the high quality of clustering we do not use this pruning technique in our implementation of pMAFIA.

The method for generation of candidate dense units proposed in [AGGR98] does not explore all possible combinations of the dense units of a lower dimension. For example, consider two 3-dimensional dense units $\{a_1, b_7, c_8\}$ and $\{b_7, c_8, d_9\}$, where (a, b, c, d) are the bins in the dimensions indicated by their subscripts. Let the numerical space be defined by an ordered set of dimensions $\{1, \ldots, 10\}$. We can easily see that the two dense units results in a 4-dimensional candidate dense unit $\{a_1, b_7, c_8, d_9\}$ which is not formed by the approach in [AGGR98]. Thus, in our approach, candidate dense cells in k dimensions, are obtained by merging any two dense cells, represented by an ordered set of $(k-1)$ dimensions, such that they share *any* of the $(k-2)$ dimensions. We shall now present a detailed analysis to derive the probability of error that results by using the candidate dense unit (CDU) generation technique proposed in [AGGR98].

3.1.2 Analysis

In the discussions that follow let Ndu be the number of dense units in dimension $k-1$ which combine to form the candidate dense units in dimension k. Let d be the dimension of the data used in the clustering process. We shall evaluate the total number of candidate dense units that could be generated by the pMAFIA algorithm followed by the number of CDUs generated using the approach in [AGGR98]. Finally we shall calculate the probability of error in the candidate dense unit generation process using the algorithm in [AGGR98] as compared to the one in pMAFIA.

Each dense unit in dimension k is completely represented by the k dimensions in which the dense unit is present and by the k bin indices in their respective dimensions. For example, $(1_1, 3_2, 4_5, 7_2)$ represents a 4 dimensional dense unit in the dimensions $(1, 3, 4, 7)$ and their subscripts indicate the bin indices in the respective dimensions.

3.1.2.1 pMAFIA: Candidate Dense Unit Generation Process

Candidate dense units in k dimensions, are obtained by merging any two dense cells, represented by an ordered set of $(k-1)$ dimensions, such that they share **any** *of the $(k-2)$ dimensions.* Each dense unit needs to be compared with every other dense unit to identify the CDUs, resulting in an $O(Ndu^2)$ algorithm. In case of clusters having a large subspace coverage, dense units in dimension k occur in every k dimensional subspace of total data space of dimension d. In order to perform a worst case analysis let us assume that the given data has clusters embedded in all possible subspaces resulting in dense units of dimension $k-1$ in ${}^dC_{k-1}$ subspaces. Consider the generation of a single candidate dense unit by combining two dense units. The two dense units can be combined together if they share any of their $k-2$ dimensions and further, if the bins in the matched dimensions are the same. The $k-1$ dimensions of each dense unit can be looked as a sequence of $k-1$ distinct integers, all of which are less than d. Thus the generation of a k dimensional candidate dense unit is equivalent to extending a $k-1$ sequence to a k sequence of integers. If d is the maximum dimensionality of data, the number of $k-1$ subsequences is given by ${}^{d}C_{k-1}$. Each $k-1$ subsequence can be extended into a k sequence in $\{d-(k-1)\}$ ways. Hence a particular set of $\{d-(k-1)\}$ sequences each of length k contain a common $k-1$ length subsequence. Any two dense units belonging to this set can be combined to form a candidate dense unit. Thus for a given $k-1$ subsequence, we can form ${}^{\{d-(k-1)\}}C_2$ candidate dense units. If N_{pMAFIA} represents the total number of possible candidate dense units that can be generated, we have

$$N_{\text{pMAFIA}} = {}^{d}C_{k-1} \times {}^{\{d-(k-1)\}}C_2 \tag{3.1}$$

as there are ${}^{d}C_{k-1}$ subsequences each of length $k-1$.

3.1.2.2 CLIQUE: Candidate Dense Unit Generation Process

Candidate dense units in k dimensions, are obtained by merging any two dense cells, represented by an ordered set of $(k-1)$ dimensions, such that they share **first** *of the $(k-2)$ dimensions.* We shall perform a worst case analysis as before and find the total number of candidate dense units, N_{CLIQUE}, that can be generated for a data set containing clusters with a maximum subspace coverage. This problem can be solved using an exhaustive enumeration technique. As before we shall look at the problem as extending a $k-1$ sequence to a k sequence. Consider a $k-1$ sequence formed by placing the number i in the i^{th} position, $\{1, 2, \ldots, (k-1)\}$. A k sequence can be formed from this $k-1$ sequence by selecting any one integer between k and d and placing it in the k^{th} position. Thus there are $d-(k-1)$ sequences of length k which have the given subsequence in common. It follows that from this set of $d-(k-1)$ sequences, each of length $(k-1)$, we can combine any two of them to form a k sequence. The total number of such k

sequences is ${}^{d-(k-1)}C_2$. Now consider the $k-1$ subsequence $\{1, 2, \ldots, k\}$, where the $(k-1)^{th}$ position contains the integer k. It can be easily seen that we have $(d-k)$ sequences each of length k which have the given subsequence in common. Thus the set of $(d-k)$ sequences can be combined among themselves to give a total number of ${}^{d-k}C_2$ sequences of length k. In an analogous manner when we vary the integer present in the $(k-1)^{th}$ position from $k-1$ to d we can form a total of

$$ {}^{d-(k-1)}C_2 + {}^{d-k}C_2 + \ldots + {}^{2}C_2 \tag{3.2}$$

sequences of length k.

We shall now vary the integer present in the $(k-2)^{nd}$ position and perform a similar analysis. If the integer $k-1$ occupies the $(k-2)^{nd}$ position and we vary the the integer present in the $k-1$ position from k to d we can form a total of

$$ {}^{d-k}C_2 + {}^{d-(k+1)}C_2 + \ldots + {}^{2}C_2 \tag{3.3}$$

candidate dense units (sequences of length k). Similarly, if k occupies the $(k-2)^{nd}$ position we can form an additional

$$ {}^{d-(k+1)}C_2 + {}^{d-(k+2)}C_2 + \ldots + {}^{2}C_2 \tag{3.4}$$

sequences of length k. Thus, by varying the integer in the $(k-2)^{nd}$ position we can form a total of

$$ \{{}^{d-k}C_2 + {}^{d-(k+1)}C_2 + \ldots + {}^{2}C_2\} + \{{}^{d-(k+1)}C_2 + {}^{d-(k+2)}C_2 + \ldots + {}^{2}C_2\} + \ldots + {}^{2}C_2 \tag{3.5}$$

sequences of length k. In a similar manner we vary the integer present in each position and by an enumerative technique find that

$$\begin{aligned} N_{\text{CLIQUE}} &= [{}^{d-(k-1)}C_2 + {}^{d-k}C_2 + \ldots + {}^{2}C_2] + \\ &\quad [\{{}^{d-k}C_2 + {}^{d-(k+1)}C_2 + \ldots + {}^{2}C_2\} + \{{}^{d-(k+1)}C_2 + {}^{d-(k+2)}C_2 + \ldots + {}^{2}C_2\} + \ldots + \{{}^{2}C_2\}] + \\ &\quad \vdots \end{aligned}$$

In order to simplify the above expression to a closed form which can be used to evaluate N_{CLIQUE}, we observe that

$$^{n}C_2 + {}^{n-1}C_2 + \ldots + {}^{3}C_2 + {}^{2}C_2 = \frac{n(n^2-1)}{6} \tag{3.6}$$

Thus, we have

$$N_{\text{CLIQUE}} = \sum_{n_{k-3}=2}^{n-(k-2)} \cdots \sum_{n_2=2}^{n_3-3} \sum_{n_1=2}^{n_2-2} \sum_{i=2}^{n_1-1} \frac{i(i^2-1)}{6} \tag{3.7}$$

where, $n_1, n_2, \ldots, n_{k-3}$ are all equal to $d-(k-1)$.

Given a value for k and d we can evaluate the value N_{CLIQUE} using a tool like *Mathematica*. From the above results we can see that N_{CLIQUE} is the number of candidate dense units that is going to be formed in dimension k for a data set which has a maximum subspace coverage by the approach in CLIQUE. And N_{pMAFIA} is the number of candidate dense units formed by pMAFIA in dimension k for a similar data set. Thus the probability of error,P_e, of missing out the formation of candidate dense units by the approach in CLIQUE is given by,

$$P_e = 1 - \frac{N_{\text{CLIQUE}}}{N_{\text{pMAFIA}}} \tag{3.8}$$

When the dimension of the data set, d, is 10 and the dimension k in which we find the candidate dense units is 5 we have N_{CLIQUE} to be 210 and N_{pMAFIA} to be 3150 and hence P_e evaluates to 0.933. For large data sets with a large subspace coverage such high probability of error can be observed.

3.2 Adaptive Grids

The computation cost of the algorithm depends on the identification and propagation to higher dimension of dense cells. The number of candidate dense cells generated and processed depend on the size of the unit in each dimension. A user defined constant size unit disregards the distribution of data with respect to the dimension. Dense regions and sparse regions are both represented by the same grid sizes. For regions with clusters this might leads to a high number of candidate dense cells and regions with noise data might also get propagated as dense cells since their densities are above a certain threshold. Further, user defined threshold levels may not be able to identify all the embedded dense regions. Figure 3.1 illustrates the building of a candidate dense unit in dimension 4 using the technique in pMAFIA.

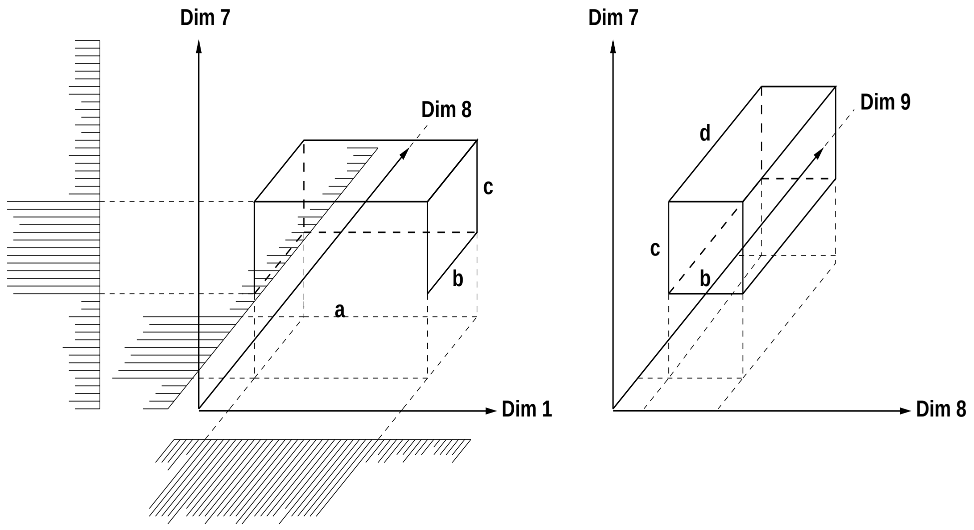


Figure 3.1: Building A Candidate Dense Unit. (Current Dimension, k = 4)

We shall first define some notations before introducing the concept of adaptive grids. Let $\mathcal{A} = \{A_1, A_2, \ldots, A_d\}$ be a set of attributes with domains $\{D_1, D_2, \ldots, D_d\}$ defining $\mathcal{S} = A_1 \times A_2 \times \ldots \times A_d$, a d-dimensional numerical space. Let $r = (r_1, \ldots, r_d)$ be a d-dimensional input record. The space $\mathcal{S}$ is partitioned into a grid consisting of non-overlapping rectangular *units*. Let $\mathcal{C} = c_{1k'} \times c_{2k''} \times \ldots \times c_{dk'\ldots'}$ be a *cell* (hyper-rectangle) if for all $i \in \{1, \ldots, d\}, c_{ik'} \subseteq D_i$. $c_{ik'} = [l_{ik'}, u_{ik'})$ is the interval in the partitioning of A_i such that $\bigcup_{allk} c_{ik} = D_i$. In [AGGR98] each dimension, i, is partitioned into ϵ equal intervals such that $c_{ik} = \frac{D_i}{\epsilon}$ for all $k = 1, \ldots, \epsilon$. A record $r = (r_1, \ldots, r_d)$ is contained in the cell C, if $l_{ik} \leq r_i < u_{ik}$ for all c_{ik}. A cell C is dense if the fraction of the total data points contained in the cell is significantly greater (by some factor α) than the value expected if data were uniformly distributed in the data space. A significant deviation from uniform distribution can be characterized by a value of α greater than 1.5.

A single pass over the data is done in order to construct a histogram in every dimension.

Algorithm 1 describes the steps of the adaptive grid technique. The domain of each dimension is divided into fine intervals, each of size x. The maximum of the histogram value within a window is taken to reflect the window value. Adjacent windows whose values differ by less than a threshold percentage are merged together to form larger windows ensuring that we divide the dimensions into variable sized bins which capture the data distribution. In essence, we fit the best rectangular wave which matches the data distribution. However, in dimensions where data is uniformly distributed this results in a single bin and indicates much less likelihood of the presence of a cluster. In order to examine these dimensions further, we split the domain into a small fixed number of partitions and collect statistics for these bins. This also allows us to set a high threshold as this dimension is less likely to be part of a cluster. This technique greatly reduces the computation time as we are able to limit the degree to which the bins from non-cluster dimensions contribute to the computation. In the dimensions with variable sized bins we set a variable threshold for each bin in that dimension. A bin in such a dimension is likely to be part of a cluster if it has a significantly (by a factor of α) greater number of points than it would have had, had the data been uniformly distributed in that dimension. Thus, for a bin of size a in a dimension of size D_i we set its threshold to be $\frac{\alpha a N}{D_i}$, where N is the total number of data points. A value of α greater than 1.5 has worked well in our experiments.

3.2.1 Effect of Grids on Cluster Quality

Figure 3.2(a) illustrates the uniform grid used in CLIQUE. This is not cognizant of the data distribution and generates many more candidate dense units for processing at each step than an adaptive grid illustrated in Figure 3.2(b). CLIQUE needs a post-processing phase to generate the minimal description length of the clusters to make the cluster definitions more amenable to the end-user. This is solved by using a greedy algorithm in CLIQUE which covers the found grids in clusters by maximal rectangles that provide coverage. Since this is an approximation of the cluster, it further adds to the complexity and reduces the correctness of the reported

Algorithm 1 *Adaptive Grid Computation*

```
$D_i$ - Domain of $A_i$
$N$ - Total number of data points in the data set
for each dimension $A_i, i \in (1, \ldots d)$
    Divide $D_i$ into windows of some small size $x$
    Compute the histogram for each unit of $A_i$, and set the value of the window to the maximum in the
    window
    From left to right merge two adjacent units if they are within a threshold $\beta$
    /* If number of bins is one, we have an equi-distributed dimension */
    if(number of bins == 1)
        Divide the dimension $A_i$ into a fixed number of equal partitions.
    Compute the threshold of each bin of size $a$ as $\frac{\alpha a N}{D_i}$
end
```

clusters.

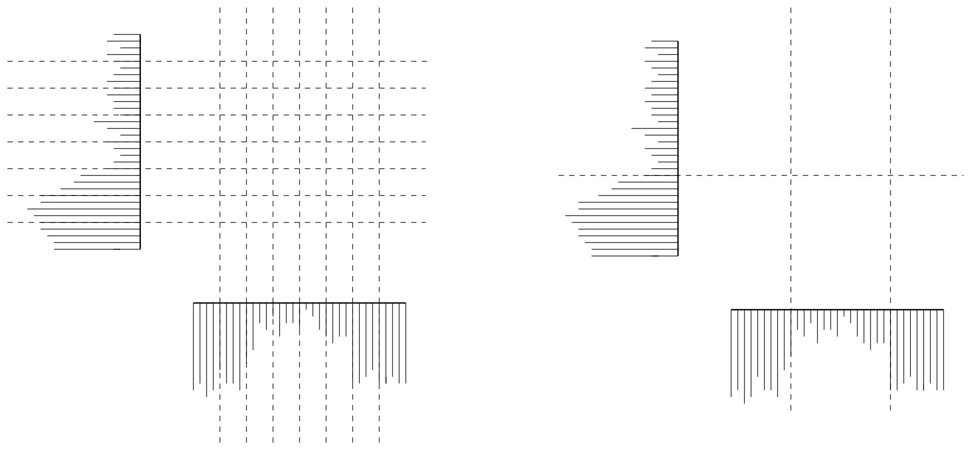


Figure 3.2: (a) Uniform grid size (b) Adaptive grid size

On the other hand, since pMAFIA uses adaptive grid boundaries its cluster definitions are minimal DNF (disjunctive normal form) expressions and report the boundaries of clusters far more accurately. Figure 3.3 shows a cluster definition in two dimensions. The cluster reported by CLIQUE, *pqrs*, shown in Figure 3.3(a), loses the boundaries of the original cluster *abcdefghijkl*. pMAFIA develops grid boundaries very close to the boundaries of the cluster and hence can incorporate most of the cluster in its definition. The DNF expression for the cluster $abcdefghijkl$ shown in Figure 3.3(b) is $(l, y) \wedge (m, z) \wedge (n, y) \wedge (m, x) \wedge (m, y)$.

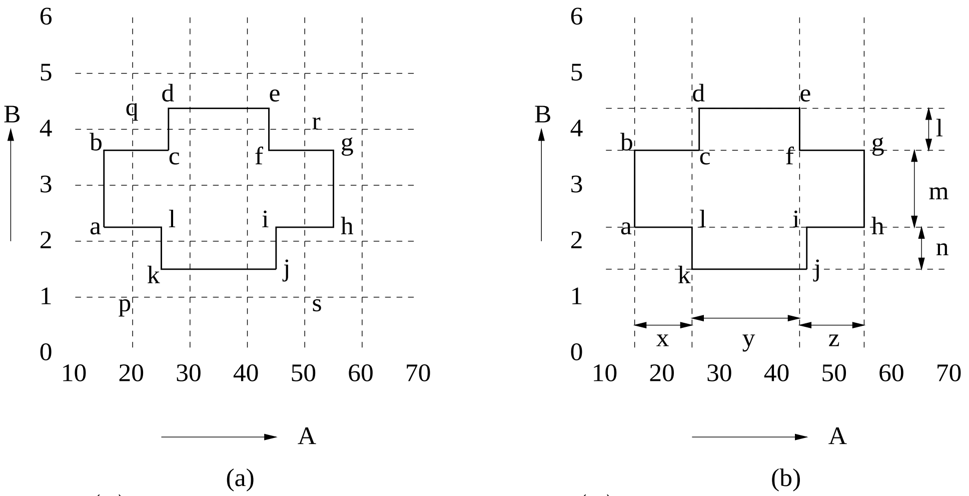


Figure 3.3: (a) Cluster discovered by CLIQUE (b) Cluster discovered by MAFIA

CHAPTER 4

pMAFIA: Parallel Subspace Clustering

We introduce a scalable parallel formulation of the subspace clustering algorithm incorporating both data and task parallelism. The underlying parallel machine is a shared-nothing architecture. Several processor-memory-disk unit are attached on a communication network as shown in Figure 4.1. Programs run in the Single Program Multiple Data (SPMD) mode, where the same program runs on multiple processors but uses portions of the data assigned to the processor. This leads to a natural data parallel mechanism for parallelism. Task parallelism is achieved by portions of the task at hand assigned to each processor. Processors communicate by exchanging messages. In contemporary parallel architectures which use this paradigm (e.g IBM SP2), the communication latencies are more than an order of magnitude larger than computation time. The communication time is comprised of a connection setup component and per message word latency, the former being the larger component. To achieve good parallelization, the overhead of communication has to be reduced by allocating tasks to processors such that relatively few number of messages are exchanged. Further, exchanges of a few large messages is better than many exchanges of small messages. A **Reduce** communication primitive using sum as its operand is used for communication to gather global statistics after statistics are gathered locally.

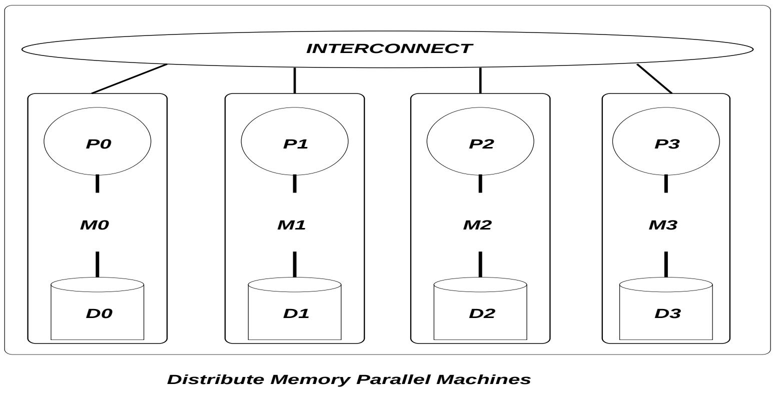


Figure 4.1: A shared-nothing architecture, P:Processor, M:Memory, D:Disk

4.1 Algorithm

pMAFIA is a disk-based parallel and scalable algorithm which can handle massive data sets with a large number of dimensions. The algorithm can also run on a single processor in which the communication steps will be ignored. Algorithm 2 shows the steps of the algorithm. In our set up on the IBM SP2, each processor reads a portion of the data from a shared disk initially and keeps it on the local disk. The bandwidth seen by a processor of an I/O access from the local disk is much higher than an access to a shared disk.

pMAFIA consists mainly of the following steps. Candidate dense units in dimension k are built by combining dense units of dimension $k-1$ such that they share *any* of the $k-2$ dimensions. The parallel algorithm *Find-candidate-dense-units()* elaborates the steps involved in this process. The algorithm spends most of its time in making a pass over the data and finding out the dense units among the candidate dense units formed in every dimension. Repeated passes over the data need to be done as the algorithm progresses building dense units of higher dimensions. After finding the histogram count of the candidate dense units in a particular dimension, dense units are identified and dense unit data structures are built for the next higher dimension. The algorithms *Identify-dense-units()* and *Build-dense-unit-data-structures()* explain these in detail. We have introduced both data parallelism for the I/O intensive phase of building the histogram count of the candidate dense units and task parallelism for all the

Algorithm 2 *pMAFIA Algorithm*

```
$N$ - Number of records; $p$ - Number of processors; $d$ - Dimensionality of data
$A_i$ - $i^{th}$ attribute $i \in d$; $B$ - Number of records that fit in memory buffer allocated at each processor
/* Each processor reads $\frac{N}{p}$ data from its local disk */
On each processor
    Read $\frac{N}{pB}$ chunks of $B$ records from local disk and build a histogram in each dimension $A_i, i \in (1, \ldots, d)$
    **Reduce** communication to get the global histogram
    Determine adaptive intervals using the histogram in each dimension $A_i, i \in d$ and also fix the threshold level
    Set candidate dense units to the bins found in each dimension
    Set current dimensionality, $k$ to 1
    while (no more dense units are found)
        if ($k > 1$)
            Find-candidate-dense-units();
        Read $\frac{N}{pB}$ chunks of $B$ records from local disk and for every record populate the candidate dense units
        **Reduce** communication to get the global candidate dense unit population
        Identify-dense-units();
        Register non dense units with the print data structures on the parent processor.
        Build-dense-unit-data-structures();
if ( Parent Processor )
    print-clusters();
end
```

remaining tasks in the algorithm. The algorithm terminates when no more dense units exists. Clusters are finally printed by the parent processor at the end of the program.

4.1.1 Data Parallelism

Each processor starts by building a histogram for all dimensions with the data of size $\frac{N}{p}$, where N is the total number of records and p the number of processors. It is during this process that the data is read from the shared disk and written to the local disks of an IBM SP2 so that subsequent data set accesses can see a much larger bandwidth. A *Reduce* communication primitive with sum as its operand gathers the global histogram on each processor. Given a vector of size m on each processor and a binary associative operation, the *Reduce* operation computes a resultant vector of size m and stores it on every processor. The i^{th} element of the resultant vector is the result of combining the i^{th} element of the vectors stored on all the processors using the binary associative operation. Each processor now determines the adaptive finite intervals for every dimension and fixes the bin sizes and thresholds for every bin formed as elaborated in Algorithm 1. Each bin thus found is considered to be a candidate dense unit. Candidate dense units are populated by a pass on local data, which is read in chunks of B records. This enables our algorithm to be applicable to out of core data sets. Since each processor accesses only the local data, of size $\frac{N}{p}$, data parallelism can be obtained. A *Reduce* operation gets the global histogram count of the candidate dense units on all processors. This is followed by the task parallel algorithms which identify the dense units and build their data structures.

4.1.2 Task Parallelism

The task of finding the candidate dense units and identifying the dense units among the candidate dense units is divided among the processors such that each processor gets an equal amount of work. Optimal task partitioning is important in order to ensure that processors do not wait for other processors to finish their tasks. After completion of the job at hand, processors exchange messages in order to obtain the global information. The benefits of task parallelism are obtained only when the computation time is much more than the communication overheads. Each candidate dense unit (CDU) and, similarly a dense unit, in the d^{th}-dimension is completely specified by the d dimensions of the unit and their corresponding d bin indices. In our implementation we store this information in the form of an array of bytes, one array for the bin indices of all the CDUs and one for the CDU dimensions. Similarly an array each is used to store the dimensions and the bin indices of the dense units. By storing the information in the form of a linear array of bytes we not only optimize for space, but also gain enormously while communicating. This helps in communicating information in a single step with the use of much smaller message buffers.

4.1.2.1 Building Candidate Dense Units

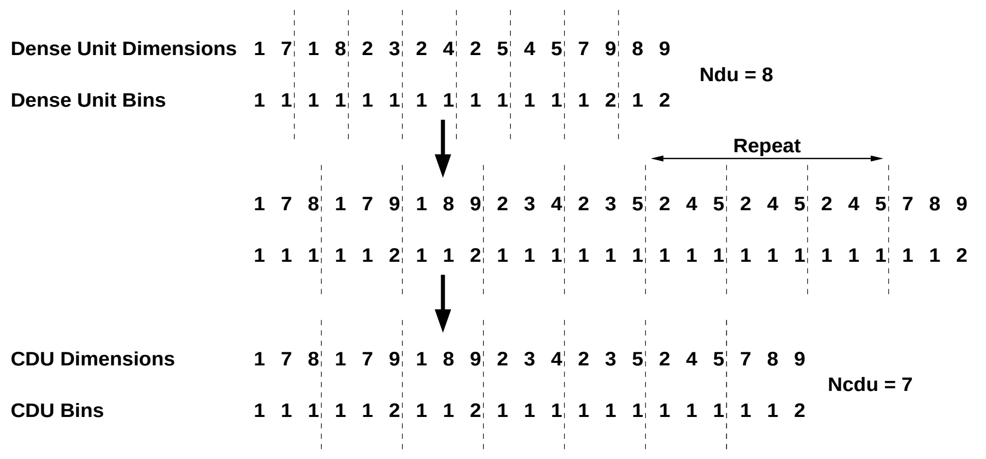

Dense Unit Dimensions 1 7 1 8 2 3 2 4 2 5 4 5 7 9 8 9
Dense Unit Bins 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2
Ndu = 8
Repeat
1 7 8 1 7 9 1 8 9 2 3 4 2 3 5 2 4 5 2 4 5 2 4 5 7 8 9
1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2
CDU Dimensions 1 7 8 1 7 9 1 8 9 2 3 4 2 3 5 2 4 5 7 8 9
CDU Bins 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2
Ncdu = 7

Figure 4.2: Building Candidate Dense Units. (Current Dimension, k = 3)

Figure 4.2 illustrates the process of building candidate dense units in dimension 3 for a data set in 10 dimensions. Candidate dense units in any dimension k is formed by combining dense units of dimension $k-1$ such that they share *any* $k-2$ dimensions. The steps of this algorithm is shown in Algorithm 3. Let the number of dense units be denoted by Ndu and the number of

CDUs by $Ncdu$. It is easy to see that each dense unit needs to be examined with every other dense unit to form candidate dense units and this would lead to a $O(Ndu^2)$ algorithm. This would amount to a huge computation when Ndu is large. One can thus generate these CDUs in parallel and speedup the whole process. We shall now derive formulations which achieve *optimal* task partitioning of the candidate dense unit generation process and result in good speedups. Let k be the dimension in which we build candidate dense units. If Ndu is the number of dense units, it can be easily seen that the total amount of computation performed is $\frac{Ndu(Ndu+1)}{2}$ when each processor works on all the dense units. Let $n_1, n_2, \ldots, n_{p-1}$ be real numbers such that $0 < n_1 < n_2 < \ldots < n_{p-1} < Ndu$ and divide the Ndu dense units into p parts such that each part of the dense unit array be processed by one of the p processors. The division of Ndu into p parts is such that each processor does an equal amount of work, equal to $\frac{Ndu(Ndu+1)}{2p}$. If processor of rank i operates in the region between n_i and n_{i+1} comparing the dense units from n_i to n_{i+1} with all the other dense units greater than n_i, as shown in figure 4.3, we have

$$Ndu * (n_{i+1} - n_i) - (\sum_{j=n_i}^{n_{i+1}-1} j) = \frac{Ndu(Ndu+1)}{2p} \tag{4.1}$$

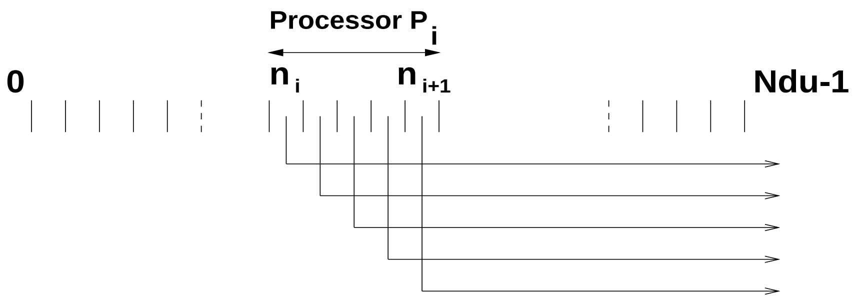


Figure 4.3: Parallel Building of Candidate Dense Units

Solving the $p-1$ equations iteratively for $n_1, \ldots, n_{p-1}$, starting from n_1, one can obtain an optimal task partition for finding the candidate dense units. Having obtained the solution for n_i, the value of n_{i+1} can be obtained by solving the above quadratic equation. CDUs generated by the processors are communicated to the parent processor which concatenates the CDU dimension and bin arrays in the rank order of the processors. This information is *broadcast* to all the processors. Dense units which could not be combined with any other dense units are registered with the parent processor as a potential cluster in dimension $k-1$. Candidate dense units are generated in parallel only when each processor is guaranteed to have a minimal amount of work. If $Ncdu$ is less than a constant τ, CDUs are generated by all processors by processing all the dense units.

Algorithm 3 *Find-candidate-dense-units()*

```
$Ncdu$ - Number of candidate dense units; CDU - Candidate dense unit
$Ndu$ - Number of dense units; $p$ - Number of Processors; $\tau$ - A constant
On each Processor
    if ( $Ndu > \tau$ )
        Find the start and end indices of the portion of the dense unit array it has to process.
        Build CDUs from its portion of the dense units.
        Send the information of local CDU count, CDU dimensions and Bins to the parent processor.
        Send non combinable dense unit information to the parent processor.
        if ( Parent Processor )
            Recv local CDU count, CDU dimension, CDU bin indices from all processors.
            Compute total number of CDUs, $Ncdu$ and concatenate the CDU dimension and CDU bin
            indices in their rank order.
            Broadcast $Ncdu$ and the concatenated CDU dimension and bin information.
            Recv non combinable dense unit information and update the data structures used for printing
            clusters.
        Eliminate-repeat-CDUs();
    else
        Build candidate dense units of dimension $k$ from all the dense units of dimension $(k-1)$.
        if ( Parent Processor )
            Register non combinable dense units with the data structures used for printing clusters.
        Eliminate-repeat-CDUs();
end
```

It can be seen from Figure 4.2 that the process of CDU generation may lead to identical candidate dense units being formed. We need to identify the repeated CDUs and retain only the unique elements. Elimination of identical CDUs is not only necessary but also greatly reduces the time during the task parallel part of the algorithm. One needs to compare each CDU with every other CDU to identify the repeated elements, which results in an $O(Ncdu^2)$ complexity algorithm. Thus, when $Ncdu$ is large we identify the repeated candidate dense units in parallel. This is similar to the generation of the candidate dense units and task parallelism is obtained by solving the above $(p-1)$ equations iteratively and with Ndu being substituted by $Ncdu$. Further, if the number of unique candidate dense units identified is still large, $(> \tau)$, we construct the data structures of the CDUs in parallel and finally exchange messages to obtain the global information. The steps of the algorithm are elaborated in Algorithm 4.

4.1.2.2 Identification of Dense Units and Building Data Structures

Candidate dense units have to be examined to determine which of them are actually dense. Each processor selects $\frac{1}{p}^{th}$ of the candidate dense units to determine the dense units. The histogram count of each CDU is compared against the threshold of all the bins which form the CDU. A local count of the number of dense units found is maintained on each processor. A *Reduce* communication operation is now performed so that all processors have the information of the dense units from the set of candidate dense units. Another *Reduce* communication operation is performed to obtain the global count of the number of dense units (Ndu) on all processors. If the number of candidate dense units is less than τ, each processor works on all the candidate dense units to determine the dense units. Algorithm 5 shows the steps involved. As before, if the number of dense units is greater than τ, the data structures of the dense units are constructed in parallel. It is important to note that we need to carefully divide the task of building the data structures as the dense units would not be distributed evenly. A linear search over the dense unit array is required to determine the start and end indices between which each processor operates for equal task distribution. Data structures of dense unit bin indices

Algorithm 4 *Eliminate-repeat-CDUs()*

```
$Nrepeat$ - repeated CDUs;
On each Processor
    if ( $Ncdu > \tau$ )
        Find the start and end indices of the portion of the CDU array it has to process.
        Identify repeated CDUs in the entire CDU array as compared to the CDUs of its portion of the array.
        **Reduce** communication to obtain the global information of the repeated CDUs.
        Set the value of $Nrepeat$ and $Ncdu = Ncdu - Nrepeat$.
        build-cdu-with-unique-elements();
    else
        Identify the repeated CDUs. Set the value of $Nrepeat$ and $Ncdu = Ncdu - Nrepeat$.
        build-cdu-with-unique-elements();
end
build-cdu-with-unique-elements(){
if ( $Ncdu > \tau$ )
    Divide $Ncdu$ by $p$ and find the start and end indices of the CDU array it has to process.
    Build the $(\frac{1}{p})^{th}$ CDU dimension and Bin arrays removing the repeated CDU elements.
    **Send** the $(\frac{1}{p})^{th}$ CDU information formed to the parent processor.
    if ( Parent Processor )
        **Recv** the $(\frac{1}{p})^{th}$ CDU information from all the processors. Concatenate them in their rank order.
        **Broadcast** the global CDU information to all processors.
else
    Build CDU data structures of the CDU dimensions and bins removing the repeated CDU elements.
}
```

and their dimensions constructed in parallel are now merged with a *Reduce* communication operation. The steps of the algorithm is shown in Algorithm 6. The algorithm then proceeds to a higher dimension and starts building the candidate dense units. It terminates when there are no more candidate dense units.

After the cluster detection process is completed, the parent processor processes all the entries registered in its data structures for printing clusters. Clusters which are a proper subset of a higher dimension cluster are eliminated and only unique clusters of the highest dimensionality are presented to the end user. This increases the usability of the algorithm to a much greater extent. pMAFIA is an unsupervised clustering algorithm. The clusters discovered by the algorithm are dependent on two parameters, α and β. α indicates the magnitude of deviation of the histogram values from that of equidistribution. A value of α greater than 1.5 has been accepted to be sufficient deviation to be considered significant in the field of statistics and data mining. Discovering clusters with higher values of α yields clusters in the data set which are more dominant than the others in terms of the number of data points contained in the cluster. Hence, choosing a suitable value of α is straightforward. The parameter β controls the process of finding adaptive grids. β controls the number of bins formed in each dimension. A low value of β results in merging adjacent bins which have nearly identical histogram values. However, histogram values of adjacent bins are rarely the same. Thus a low value of β results in a large number of bins in each dimension. This results in a greater time for computation but will yield better quality clusters. High values of β results in merging all the bins in a given dimension and will yield poor quality clusters. Our algorithm is not very sensitive to the value of β. Clusters of high quality are discovered efficiently by pMAFIA when too low or too high values of β are avoided. A value of β in the range of 25% to 75% has worked well in our experiments. Specifically, the results reported in the paper are with β equal to 35%.

Algorithm 5 *Identify-dense-units()*

```
On each Processor
    if ( $Ncdu > \tau$ )
        Divide $Ncdu$ by $p$. Find the start and end indices of the portion of the CDU array which it has
        to process.
        For each CDU in its portion of the array, compare th CDU histogram count with the thresholds
        of the bins which form the CDU.
        Determine if the CDU is *dense* or not. Maintain a local count of the number of dense units
        detected.
        **Reduce** communication to get the dense unit information of all the CDUs followed by another
        **Reduce** communication to get the total number of dense units $Ndu$.
    else
        For all CDUs, $Ncdu$, compare the CDU histogram count with the thresholds of the CDU bins.
        Determine if the CDU is *dense* or not. Find the total number of dense units $Ndu$.
end
```

Algorithm 6 *Build-dense-unit-data-structures()*

```
On each Processor
    if ( $Ndu > \tau$ )
        Divide $Ndu$ by $p$. Find the start and end indices of the CDU array on which it has to process.
        Construct the data structures related to the dimension and bin indices of the dense units from its
        portion of the CDU array.
        **Reduce** communication to get the global information of the data structures of the dense units.
    else
        From all CDUs, construct data structures related to the dimension and bin indices of the dense
        units.
end
```

4.2 Analysis

Let k represent the highest dimensionality of any dense unit in the data set. The running time of the algorithm is exponential in k. This is due to the fact that if a dense cell exists in k dimensions, then all its projections in a subset of k dimensions, $O(2^k)$ combinations, are also dense. Thus one needs to check for clusters in all possible subspaces among the k cluster dimensions. However, with the use of adaptive grids the number of candidate dense units is greatly reduced and thus enables pMAFIA to scale gracefully with the dimensionality of the data set and the data set size. Let α be the constant for communication and S be the size of messages exchanged among processors and N, the total number of records. Also, let B be the number of records that fit in memory buffer on each processor and let γ be the I/O access time for a block of B records from the local disk. The computation time complexity of the algorithm is then $O(c^k)$, where c is a constant. The total I/O time on each processor is $O(\frac{N}{pB}k\gamma)$ as each processor has to read just $\frac{N}{p}$ part of the data in chunks of B records. The factor of k is due to the k passes required over the database before the algorithm terminates. The communication time is $O(\alpha Spk)$. The total time complexity of the algorithm is then $O(c^k + \frac{N}{pB}k\gamma + \alpha Spk)$. The running time on a single processor can simply be obtained by substituting $p = 1$ and $S = 0$, as there will be no communication.

CHAPTER 5

Performance Experiments

In this chapter we shall discuss the performance of pMAFIA with respect to various issues. Our implementation of pMAFIA is on an IBM SP2, which is a collection of IBM RS/6000 workstations connected with a fast communication switch. In our setup of 16 processors, each processor is a 120MHz thin node with 128MB main memory, and a 2GB disk on each node of which 1GB is available as scratch space. The communication switch has a latency of 29.3 milliseconds and the bandwidth is 102 MB/sec (uni-directional) and 113 MB/sec (bi-directional) [May98]. We use Message Passing Interface (MPI) [MPI98] for communication between processors.

We report results of our experiments with MAFIA on a variety of synthetic and real world data sets and show the performance and scalability of the algorithm in terms of - size of database, dimensionality of data, dimensionality of clusters, performance improvement over CLIQUE, parallel speedup as number of processors are increased, and quality of the reported clusters.

5.1 Synthetic Data Generation

We created a data generator to produce the data sets used in our experimental results. The data generator takes from the user a definition of cluster(s) in different subspaces. The extents of the cluster are specified by the user in every dimension of the subspace in which it is embedded. Data can vary between any user specified maximum and minimum values for all attributes. Unlike CLIQUE, our data sets can have arbitrary cluster shapes instead of just hyper-rectangular regions.

The attribute values in the dimensions of the subspace in which the cluster is defined is generated as follows. We scale all the dimensions to lie in the range $[0 \ldots 100]$. We generate the data points such that each unit cube (which is a part of the user defined cluster) in this scaled space contains *at least* one point. The data so generated is scaled back appropriately into the user specified attribute ranges. This method, as against randomly populating the user defined cluster region, ensures that we have a cluster exactly as defined by the user and helps us to validate the results easily. Randomly populating points in the cluster region, as used in CLIQUE, may not form the user required cluster strictly. However, since CLIQUE uses fixed sized bins, the cluster boundaries reported are discretized and is an approximation of the cluster. For the remaining attributes we select a value at random from a uniform distribution over the entire range of the attribute. It is to be noted that long sequences of uniform random numbers generated by typical Unix random number generators (Linear Congruential Generators) have been shown to exhibit regular behavior by falling into specific planes [Knu80]. To avoid this, we use a random number generator called the Inversive Congruential Generator which has better statistical properties [EHG92]. After populating the user defined cluster we add an additional 10% noise records to the data set. Values for all the attributes in these noise records are independently drawn at random over the entire range of the attribute. It is also important to mention here that we split the user defined cluster ranges into parts and permute them before we generate the synthetic data. This will ensure that there is no specific behavior with respect to the input order of the data records.

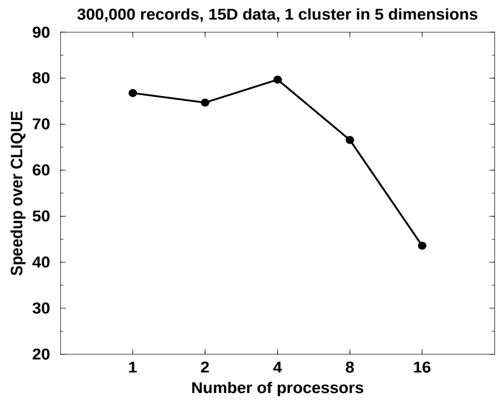
5.2 Experimental Results

We present performance results of our adaptive grid technique in pMAFIA and compare it with our implementation of CLIQUE. The results presented here are on our synthetic data sets generated using our data generator. Further, we show results for scalability of pMAFIA with data dimension, cluster dimensionality and data records, followed by results of the improvement in quality of the clustering obtained. Finally, we show results of the parallelization of pMAFIA observing both scale up and speedup. In the results we report below time taken for data to be read from the shared disk onto the local disks of the processors is not included as data is read over an NFS and this time varies greatly based on the network activity. We first report the results on the synthetic data sets followed by the results on real data.

5.2.1 Synthetic Data Results

5.2.1.1 Effect of using Adaptive Grids

Figure 5.1(a) shows the speedup obtained over CLIQUE on different number of processors. The results are for a database with 300,000 records in 15 dimensions with one cluster embedded in a 5 dimensional subspace. We set the threshold percentages for the bins in various dimensions depending on the bin size based on Algorithm 1. However, while running CLIQUE we set the threshold τ to a uniform high value of 2% in all dimensions so that it could discard a larger number of candidate dense units in every pass over the data. Each dimension is divided into 10 bins for the results reported for CLIQUE. We see good parallelization for both CLIQUE and pMAFIA from the table in Figure 5.1(b). Figure 5.1(a) shows that pMAFIA performs 40 to 80 times better than CLIQUE for this data set. It is very important to note here that the results presented for CLIQUE are by using the algorithm presented in [AGGR98], where in k dimensional CDUs are formed by combining $k-1$ dimensional dense units which share the *first* $k-2$ dimensions. However when we tried to run the corrected version of CLIQUE in which we combine two $k-1$ dimensional dense units which share *any* $k-2$ dimensions we could not complete the run for this data set even with in 2 hours on 8 processors. Two orders of magnitude speedup have been observed for data sets with much larger dimensionality and database size. This is due to the very nature of our algorithm which decides a minimum set of bins in a dimension based on its *interestingness* as observed from the data histogram in every dimension. This results in a set of candidate dense units much lower than the one obtained by equal number of bins in all dimensions. The latter results in an explosion of candidate dense units in a very high dimension space. Figure 5.1(b) shows the comparative execution times of pMAFIA and CLIQUE for the data set used in Figure 5.1(a).



	Number of Processors				
	1	2	4	8	16
pMAFIA	32.15	17.73	8.34	5.08	4.51
CLIQUE	2469.12	1324.51	664.65	338.19	184.36

(a) (b)

Figure 5.1: (a) Speedup of pMAFIA over CLIQUE (b) Comparative execution times (in seconds) for 300,000 records in a 15 dimensional space with 1 cluster in 5 dimensions

Subspace overlap of clusters heavily affects the computation time. Candidate dense units explodes with increase in the number of distinct cluster dimensions as we consider more combinations of distinct dimensions. This effect would be even more pronounced in CLIQUE as it treats all dimensions equally without exploiting the data distribution in each dimension. A direct comparison to the results reported in [AGGR98] is not possible since they do not mention the subspace overlap of the clusters.

5.2.1.2 Effect of Adaptive Grids on Computational Complexity

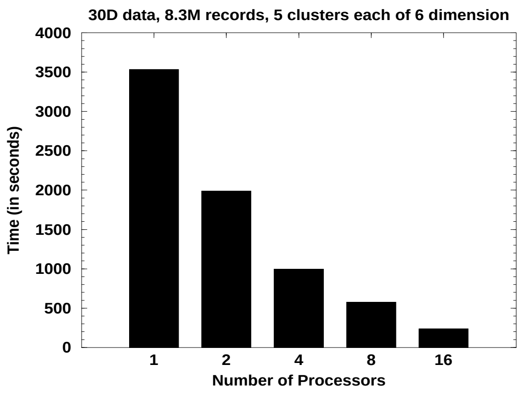
We generated a data set containing a *single* 7 dimensional cluster in a 10 dimensional data space. The data set contained 5.4 million records. In this experiment we ran pMAFIA and compared it with our modified implementation of [AGGR98] where candidate dense units in dimension k are formed by combining $k-1$ dimensional dense units which share *any* $k-2$ dimensions. This modification of the algorithm drastically increases the search space for finding the embedded clusters. Table 5.1 shows the number of CDUs (Ncdu) and the number of dense units (Ndu) generated in this experiment. Results presented for [AGGR98] are with 10 uniform sized bins in each dimension and a threshold percentage of 1% for all dimensions. pMAFIA discovered correctly the single cluster embedded. However, CLIQUE discovered 75 unique clusters each of dimension 6 and 546 unique clusters each of dimension 7. Most of these clusters contained atleast one cluster dimension which was not part of the original defined cluster. The increase in the search space of the algorithm detects all embedded clusters. Since [AGGR98] treats all dimensions of the data set in the same way by forming uniform sized grids, its computation time grows drastically. However, pMAFIA exploits the data distribution in each dimension by forming adaptive grids and thus greatly reduces the computation time by forming as few bins as required in each dimension. This results in very few candidate dense units being generated as seen in Table 5.1. For this experiment on a 400 MHz pentium II processor, pMAFIA took just 691 seconds while the modified implementation of CLIQUE took 79162 seconds resulting in a factor of 114.56 speedup.

Table 5.1: Candidate Dense Units generated by pMAFIA and CLIQUE

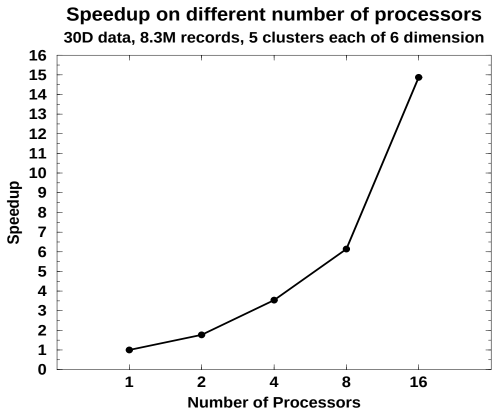
Dimension		2	3	4	5	6	7	8
pMAFIA	Ncdu	21	35	35	21	7	1	0
	Ndu	21	35	35	21	7	1	0
CLIQUE	Ncdu	2313	5739	19215	38484	42836	24804	5820
	Ndu	535	1572	3337	3870	2312	546	0

5.2.1.3 Parallel Performance

Figure 5.2 presents the times obtained by pMAFIA. The results are on a 30 dimensional data set with 8.3 million records containing 5 clusters each in a different 6 dimensional subspace. The threshold percentages were automatically set by the algorithm for bins in all the dimensions based on the data distribution. From the plot it can be seen that we have achieved near linear speedups. This is due to the algorithm being heavily data parallel and time for computation goes down linearly with the increase in the number of processors. We observed that bulk of the time is taken in populating the candidate dense units which is completely data parallel. Communication overhead introduced due to the parallel algorithm is negligible as compared to the total time. The time taken to build the initial histogram is also a very small percentage of the total time. Further, we observed that the I/O time decreases with the increase in the number of processors as expected due to the data parallelism.



(a)



(b)

Figure 5.2: (a) Parallel run times of pMAFIA (b) Speedup observed

5.2.1.4 Scalability with Database Size

Figure 5.3 shows the results for scalability with the database size for a 20 dimensional data with the number of records ranging from 1.45 million to 11.8 million. There were 5 clusters embedded in 5 different 5-dimensional subspaces. The results reported are on 16 processors. The thresholds for different bins were determined automatically by Algorithm 1. The time spent in cluster detection almost shows a direct linear relationship with the database size. The linear behavior is because the number of passes over the database depends only on the dimensionality of the embedded cluster. An increase in the size of the database just means that more data has to be scanned on every pass over the database while finding the dense units resulting in a linear increase in time.

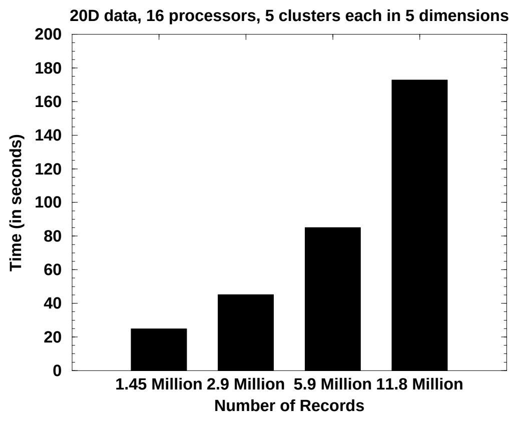
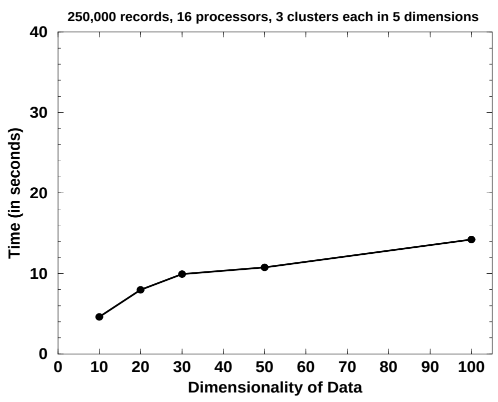


Figure 5.3: Scalability with increasing database size

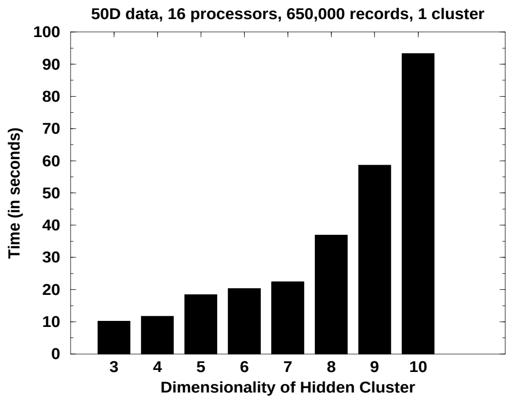
5.2.1.5 Scalability with Dimensionality of Data and Cluster Dimensionality

In Figure 5.4(a) we see that pMAFIA scales very well with the increase in data dimension. The results shown are on a data set of 250,000 records with 3 clusters each in a five dimensional subspace, with a total of 9 distinct dimensions. The results reported are on 16 processors with similar behavior observed on other number of processors. The better than linear behavior is due to the fact that our algorithm makes use of data distribution in every dimension and only depends on the number of distinct cluster dimensions. CLIQUE not only depends on distinct cluster dimensions but also on the data dimensionality. Hence it exhibits a quadratic behavior with respect to the data dimensionality as reported in [AGGR98].

Figure 5.4(b) shows the scalability observed with increasing cluster dimensionality in pMAFIA. The results reported are for a 50-dimensional data set with 650, 000 records containing 1 cluster on 16 processors. Results show that the time increase with cluster dimensionality reflects the time complexity of the algorithm, $O(\frac{N}{pB}k\gamma + \alpha Spk + c^k)$, where N is the number of records, p is the number of processors, k is the maximum dimensionality of the cluster, B is the number of records read in one chunk from disk, γ is the I/O access time for a block of B records, α is the constant for communication, S is the total size of the message communicated, and c is a constant. Similar behavior is observed on other number of processors.



(a)



(b)

Figure 5.4: Scalability with increasing (a) Data Dimensionality (b) Cluster Dimensionality

5.2.1.6 Quality of Results

We compare the quality of the results obtained by pMAFIA with those of CLIQUE, shown in Table 5.2. The results are for a relatively small data set with 400, 000 records in 10 dimensions with 2 clusters each in a different 4 dimensional subspace. We ran our parallelized version of

CLIQUE on 16 processors. In the first case we set the number of bins to be 10 in every dimension and also set a threshold of 1% uniformly in all dimensions (as implemented in CLIQUE). In the second case we set arbitrary number of bins in each dimension (with a minimum of 5 bins to a maximum of 20 bins per dimension). The threshold in each dimension is set to 1%. In the first case CLIQUE reported the correct dimensions of the 2 clusters, however, it detected the 2 clusters only partially and large parts of the clusters were thrown away as outliers. In the second run, with a variable number of bins in each dimension, it completely failed to detect one of the clusters and, as before, the single cluster was partially detected. This is due to the inherent nature of CLIQUE which uses fixed discretization of the dimensions and hence results in a loss in the quality of the cluster obtained. For real life data sets validation of the results obtained would be a very hard task and thus bin selection would be a non trivial problem. When we ran pMAFIA on the same data set on 16 processors, both the clusters and the cluster boundaries in each dimension were accurately reported.

Table 5.2: Comparison of the quality of results obtained by pMAFIA and CLIQUE

	Cluster Dimensions	Clusters Discovered
CLIQUE (fixed 10 bins)	{1,7,8,9} {2,3,4,5}	{1,7,8,9} {2,3,4,5}
CLIQUE (variable bins)	{1,7,8,9} {2,3,4,5}	{2,3,4,5}
pMAFIA	{1,7,8,9} {2,3,4,5}	{1,7,8,9} {2,3,4,5}

We performed another experiment on a data set with 1.6 million records in a 10 dimensional space. The data set consisted of 2 clusters, one in a 4 dimensional subspace and the other in a 6 dimensional subspace with a subspace overlap of 2 dimensions. We set the number of bins to be 10 in all the dimensions while running CLIQUE. Further, a threshold of 1% was set for all the dimensions as used in [AGGR98]. CLIQUE partially reported the two clusters with a mismatch of 1 dimension in both the reported clusters. Also, the boundaries reported by CLIQUE were a crude approximation of the embedded clusters. However, pMAFIA reported both the clusters with their exact cluster dimensions and cluster boundaries as was defined for the data set.

5.2.2 Real Data Sets

We applied pMAFIA on two real world data sets to discover embedded clusters in different subspaces. The data sets used are of very high dimensionality. Although the data set size is small compared to the synthetic data sets used in our experiments, we report the results mainly to illustrate the applicability of pMAFIA, a completely unsupervised data mining algorithm, on real world problems. Our experiments show that for such data sets task parallelism is more effective than data parallelism. However, it is easy to see that one would gain enormously from both task and data parallelism with the increase in the data set size.

5.2.2.1 One Day Ahead Prediction of DAX

The data set is a one day ahead prediction of the German Stock index (DAX, Deutscher Aktien Index) based on twelve input time series which includes different stock indices, bond indices and inflation indicators like the DAX Price Index, DAX Price-Earnings Ratio and DAX Composite. Detailed explanation of the DAX data set and description of the inputs can be found in [WZ98]. The data set is in 22 dimensions with 2757 records. We choose the value of α to be 2 for the results reported. The clusters discovered are given in Table 5.3. The results reported are with 8 processors taking 8.16 seconds. All the clusters reported are unique and no cluster in a lower dimension subspace is a proper subset of a higher dimensional cluster.

Table 5.3: Clusters Discovered in the DAX Data Set

Cluster Dimension	3	4	5	6
Number of Clusters Discovered	161	134	104	24

5.2.2.2 Ionosphere Data

We applied pMAFIA to the radar data that was collected by a system in Goose Bay, Labrador [BM98]. The data set is of 34 dimensions with 351 records. In the results reported on 8 processors we set α to be 2. We discovered 158 unique clusters in 3 dimensional subspaces and 32 unique clusters in 4 dimensional subspaces. However, when we increased α to 3 we discovered one single cluster in a 3 dimensional subspace. PROCLUS [APW+99] has reported two clusters one each in 31 and 33 dimensions for this data set. However, we believe that this could be in part due to an incorrect value of l, the average cluster dimensionality, chosen by the user.

5.2.2.3 EachMovie Recommendation Data

We applied pMAFIA to a massive data set which contained movie ratings. The data set was collected by DEC Systems Research Center over a period of 18 months. During this period 72916 users entered a total of $2,811,983$ numeric ratings for 1628 different movies (films and videos). Each rating is characterized by four numbers. These numbers contain information about the user-id, movie-id, a score $(0-1)$ and a weight $(0-1)$. In this 4 dimensional data set we discovered 7 interesting clusters all of dimension 2 in just about 28 seconds on a 400 MHz Pentium II processor. However, our implementation of CLIQUE took 151 seconds and found clusters in dimensions 2, 3 and 4. For this experiment using CLIQUE, we had 10 uniform sized bins in all dimensions with a uniform threshold of 1% in each dimension. However, when we ran CLIQUE with an arbitrary number of bins in each dimension we discovered a different set of clusters. Thus evaluating the results and finding the correct set of clusters would not be easy. Clusters discovered by pMAFIA revealed interesting information about which set of movies were rated most by which set of users. Since the dimensionality of the data is low (4), the experiment reveals the scalability aspect of pMAFIA on real data. The massive synthetic data sets used in the previous sections demonstrate the performance of pMAFIA both with respect to scalability with data size and also scalability with the dimensionality of the data. Table 5.4 gives the run times of pMAFIA on this data set along with the corresponding speed-ups on different number of processors of an IBM SP2. The near linear speed-ups obtained demonstrate the scalability of pMAFIA on large real data sets too.

Table 5.4: Parallel Performance on the Movie Data Set.

Processors	1	2	4	8	16
Run Times (in sec)	144.86	70.47	36.86	20.35	10.18
Speed Up	1	2.06	3.93	7.11	14.23

Chapter 6

Summary and Future Work

6.1 Conclusion

In this thesis we presented a new clustering algorithm, **pMAFIA**, for massive data sets. Our algorithm is designed to handle massive out of core data sets and mine interesting cluster patterns in the data. pMAFIA is a density and grid based clustering algorithm. Clusters are defined as high density regions than their surroundings. We introduced the concept of *Adaptive Grids* to form a grid in the multi-dimensional data space based on the data distribution in each dimension of the data. Adaptive grids help not only reduce the computation greatly but also improve the quality of clustering. Forming grids based on the data distribution also helps to capture the cluster boundaries effectively. We devised a new bottom up algorithm to form clusters of higher dimensions. Dense units in a lower dimension, $k-1$, which share any of their $k-2$ dimensions are combined to form potential clusters (candidate dense units) in a higher dimension, k.

Further, we have introduced both data parallelism and task parallelism in our approach. Massive data sets can be processed in parallel giving rise to near linear speedups. Tasks of building the candidate dense units and identifying the dense units among those formed is also done in parallel when the number of clusters is large. Communications overheads due to parallelism have been observed to be negligible. Performance results on a large number of synthetic data sets and some real data sets have shown pMAFIA to be scalable with the database size, data dimensionality and cluster dimensionality. Quality of clusters obtained is of high quality capturing complete cluster regions.

6.2 Future Work

Business data is collected continuously and information needs to be extracted from this data regularly to know about the trends in the business. However algorithms require huge amounts of time to process these massive data sets. Thus it is advantageous to develop incremental algorithms to be applied to the incremental data that is generated. We are currently in the process of implementing an incremental subspace clustering algorithm, *iMAFIA*. We would like to cluster the incremental data sets periodically and extract the new cluster information from them. Further, the new clusters formed are analyzed with the cluster information from the base data and new clusters generated are recorded.

Another interesting area is that of categorical clustering. Consumer data is usually in the form of non-numerical data. Examples of such data could be any sort of sales transactions which would enumerate the items purchased in each transaction along with their characteristics. Techniques applied for clustering numerical data cannot be directly applied to cluster categorical data as categorical data do not have any inherent ordering among the values of attributes. Thus new methods to explore clusters in categorical clustering need to be developed. Further, finding clusters in subspaces makes this problem more interesting. Parallelization would speed up the whole process.

References

[AA96] P. Arabie and L.J Arabie. An Overview of Combinatorial Data Analysis. *Clustering and Classification, World Scientific Publication*, pages 5–63, 1996.

[ABKS99] M. Ankerst, M.M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: Ordering Points To Identify the Clustering Structure. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1999.

[AGGR98] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. ACM SIGMOD International Conference on Management of Data*, 1998.

[AIS93] R. Agrawal, T. Imielinski, and A. Swami. Mining Associations between Sets of Items in Massive Databases . In *Proc. of the ACM SIGMOD International Conference on Management of Data*, pages 207–216, May 1993.

[ALSS95] R. Agrawal, K. Lin, H. S. Sawhney, and K. Shim. Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases. In *Proc. of the 21st Int'l Conference on Very Large Databases*, 1995.

[APW$^+$99] C. Aggarwal, C. Procopiuc, J.L. Wolf, P.S. Yu, and J.S. Park. A Framework for Finding Projected Clusters in High Dimensional Spaces. In *Proc. ACM SIGMOD International Conference on Management of Data*, 1999.

[AS94a] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the 20th International Conferance on Very Large Data Bases* , pages 478–499, September 1994.

[AS94b] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 20th International Conference on Very Large Databases*, 1994.

[AS95] R. Agrawal and R. Srikant. Mining Sequential Patterns . In *The Proceedings of the* 11*th International Conferance on Data Engineering*, 1995.

[BDO95] M.W. Berry, S.T. Dumais, and G.W. O'Brien. On the complexity of some common geometric location problems. In *SIAM Review*, volume 37, pages 573–595, 1995.

[BFOS84] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression trees.* Wadswoth, Belmont, 1984.

[BM98] C.L. Blake and C.J. Merz. UCI repository of machine learning databases. 1998.

[CFZ99] C. Cheng, A.W. Fu, and Y. Zhang. Entropy-based subspace clustering for mining numerical data. In *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1999.

[CHY96] M.S. Chen, J. Han, and P.S. Yu. Data Mining: An Overview from a Database Perspective. In *IEEE Transactions on Knowledge and Data Engineering* , volume 8, pages 866–883, 1996.

[DM99] I.S. Dhillon and D.S. Modha. A data-clustering algorithm on distributed memory multiprocessors. *Large-Scale Parallel KDD Systems, ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1999.

[DO74] B.S. Duran and P.L. Odell. *Cluster Analysis, A Survey.* Springer-Verlag, 1974.

[EHG92] Jurgen Eichenauer-Herrmann and Holger Grothe. A new inversive congruential pseudorandom number generator with power of two modulus. *ACM Transactions on Modeling and Computer Simulation*, 2(1):1–11, January 1992.

[EKSX96] M. Ester, H-P. Kriegel, J. Sander, and X. Xu. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proc. of the 2nd International Conference on Knowledge Discovery in Databases and Data Mining*, 1996.

[FPSSU94] U.M. Fayyad, G. Piatesky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in data mining and knowledge discovery.* MIT Press, 1994.

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

[GGR99] V. Ganti, J. Gehrke, and R. Ramakrishnan. CACTUS: Clustering Categorical Data Using Summaries . In *Proc. of ACM SIGKDD, International Conference on Knowledge Discovery and Data Mining*, 1999.

[GKR98] D. Gibson, J. Kleinberg, and P. Raghavan. Clustering categorical data: An approach based on dynamical systems. In *Proc. of the 24th VLDB Conferance, New York*, 1998.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Morgan Kaufmann, 1989.

[GRS98] S. Guha, R. Rastogi, and K Shim. CURE: An Efficient Clustering Algorithm for Large Databases. In *Proc. ACM SIGMOD International Conference on Management of Data*, 1998.

[GRS99] S. Guha, R. Rastogi, and K. Shim. Rock: a robust clustering algorithm for categorical attributes. In *Proceedings of International Conference on Data Engineering*, 1999.

[HKKM97] E.H. Han, G. Karypis, V. Kumar, and B. Mobasher. Clustering in a high-dimensional space using hypergraph models (1997). In *Technical Report, University of Minnesota, Department of Computer Science, 97-019*, 1997.

[Hua97] Z. Huang. A Fast Clustering Algorithm to Cluster Very Large Categorical Data sets in Data Mining. In *Proc. SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, 1997.

[JD88] A.K. Jain and R.C Dubes. *Algorithms for Clustering Data.* Prentice-Hall Inc., 1988.

[KHK99] G. Karyapis, E.H. Han, and V. Kumar. CHAMELEON: A hierarchical clustering algorithm using dynamic modeling. *IEEE Computer, Special Issue on Data Analysis and Mining*, 1999.

[Knu80] Donald E Knuth. *The Art of Computing*, volume 2, Seminumerical Algorithms. Addison Wesley, 3^{rd} edition, 1980.

[May98] F. May. *SP Switch Performance.* IBM Corporation, 1998.

[MPI98] MPI Forum, http://www-unix.mcs.anl.gov/mpi. *MPI: A Message Passing Interface Standard*, 1998.

[MS83] R.S Michalski and R.E. Stepp. Learning from Observation: Conceptual Clustering. *Machine Learning: An Artificial Intelligence Approach*, I:331–363, 1983.

[MS84] N. Megiddo and K.J. Supowit. On the complexity of some common geometric location problems. In *SIAM Journal of Comput.*, volume 13, pages 182–196, 1984.

[MST94] D. Michie, D.J. Spiegelhater, and C.C. Taylor. *Machine Learning, Neural and Statistical Classification* . Ellis Horwood, 1994.

[NH94] R.T. Ng and J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. In *Proc. 20th International Conference on Very Large Databases*, 1994.

[Ols95] C.F. Olson. Parallel algorithms for hierarchical clustering. *Parallel Computing*, 21, 1995.

[Pro96] C.M. Procopiuc. Clustering problems and their applications. In *Computer Science Department, Duke University*, 1996. `http://www.cs.duke.edu/~magda/`.

[SCZ98] G. Sheikholeslami, S. Chatterjee, and A. Zhang. WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases. In *Proc. 24th International Conference on Very Large Databases*, 1998.

[SG96] P.A. Schrodt and D.J. Gerner. Using cluster analysis to derive early warning indicators for political change in the middle east, 1979-1996. *Presented at the American Political Science Association, San Fransisco*, 1996.

[Sib73] R. Sibson. SLINK: An optimally efficient algorithm for the single-link cluster meethod. *The Computer Journal*, 16(1), 1973.

[WZ98] A. S. Weigend and H. G. Zimmermann. Exploiting local relations as soft constraints to improve forecasting. *Journal of Computational Intelligence in Finance*, 6:14–23, 1998.

[XEKS98] X. Xu, M. Ester, H.-P. Kriegel, and J. Sander. A distribution-based clustering algorithm for mining in large spatial databases. In *Proceedings of International Conference on Data Engineering*, 1998.

[ZRL96] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proc. ACM SIGMOD International Conference on Management of Data*, 1996.