MAFIA: Efficient and Scalable Subspace Clustering for Very Large Data Sets

Sanjay Goil *Harsha Nagesh* *Alok Choudhary*

Technical Report No. CPDC-TR-9906-010

June 1999

Center for Parallel and Distributed Computing
Department of Electrical & Computer Engineering
Northwestern University
Technological Institute
2145 Sheridan Road, Evanston IL-60208

MAFIA: Efficient and Scalable Subspace Clustering for Very Large Data Sets*

Sanjay Goil Harsha Nagesh Alok Choudhary
Department of Electrical & Computer Engineering
Northwestern University,
Technological Institute,
2145 Sheridan Road, Evanston, IL-60208
{sgoil,harsha,choudhar}@ece.nwu.edu

Abstract

Clustering techniques are used in database mining for finding interesting patterns in high dimensional data. These are useful in various applications of knowledge discovery in databases. Some challenges in clustering for large data sets in terms of scalability, data distribution, understanding end-results, and sensitivity to input order, have received attention in the recent past. Recent approaches attempt to find clusters embedded in subspaces of high dimensional data. In this paper we propose the use of adaptive grids for efficient and scalable computation of clusters in subspaces for large data sets and large number of dimensions. The bottom-up algorithm for subspace clustering computes the dense units in all dimensions and combines these to generate the dense units in higher dimensions. Computation is heavily dependent on the choice of the partitioning parameter chosen to partition each dimension into intervals (bins) to be tested for density. The number of bins determines the computation requirements and the quality of the clustering results. Hence, it is important to determine the appropriate size and number of the bins. We present MAFIA, which 1) proposes adaptive grids for fast subspace clustering and 2) introduces a scalable parallel framework on a shared-nothing architecture to handle massive data sets. Performance results on very large data sets and a large number of dimensions show very good results, making an order of magnitude improvement in the computation time over current methods and providing much better quality of clustering.

1 Introduction

Clustering is one of the primary data mining tools which helps a user understand the natural grouping of attributes in a data set. Large amount of data are being collected in many business and scientific domains which is being subjected to data analysis for finding interesting and previously unknown patterns. Clustering techniques for large scale data containing a large number of attributes is gaining importance to make the task of knowledge discovery and data mining from various databases much more efficient [1].

*This work was supported in part by NSF Young Investigator Award CCR-9357840, NSF CCR-9509143 and Department of Energy ASCI/ASAP program

Emerging applications of data mining are making refinements to the clustering algorithms that have been studied in the past. 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, clusters can be embedded in some lower dimensional space, a subspace of the total data space. Algorithms for efficient computation of clusters are as important as correctly detecting the clusters in the data. 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.

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. Dimensionality reduction techniques such as principal component analysis and Karhunen-Loève transformation [2], transform the original data space into a lower dimensional space by forming dimensions which are linear combinations of 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. Also, clusters in subspaces of the original space cannot be found using this method. Most techniques in the database community have developed scalable versions of the clustering algorithms used in earlier work in statistics and data mining [3, 4]. These operate and find clusters in the whole data space. We use the method developed in CLIQUE [5] to identify clusters in subspaces which is a density and grid based approach.

1.1 Contributions of the paper

In a grid and density based subspace clustering algorithm, the number of bins in each dimension determine the computation requirements and the quality of the clustering results. In data sets with a large number of dimensions and varying data distributions, a uniform grid size leads to enormous amount of computation for fine grids and very poor cluster quality for coarse grids. Hence, it is important to have the correct size of the grids, embedded by the bins, depending on the data distribution. In this paper we present MAFIA[1], a technique for adaptive computation of the finite intervals (bins) in each dimension, which are merged to explore clusters in higher dimensions. Consequently, adaptive grid sizes are used, which reduce the computation and improves the clustering quality by concentrating on the portions of the data space which have more points and thus more likelihood of having clusters. This also enables minimal length DNF expressions, important for interpreting results by the end-user, to capture the cluster definition without a complicated post-processing step needed in other methods. Performance results show MAFIA to

[1]For Merging of Adaptive Finite Intervals (and is more than a clique)

be 40 to 50 times faster than CLIQUE, due to the use of adaptive grids. Further, to enable scalability, we also introduce parallelism to obtain a highly scalable clustering algorithm for large data sets. This allows us to handle data sets, much larger than the ones reported in the most recent literature in clustering research. To the best of our knowledge, this is one of the first efforts in practical parallel clustering techniques for large data sets.

We describe recent work on clustering techniques in databases in Section 2. Density and grid based clustering is presented in Section 3 where we describe subspace clustering as introduced by [5]. Section 4 introduces our approach of using an adaptive grid and an algorithm for calculating bin boundaries for dimensions automatically. Section 5 describes the parallel algorithms for scalable parallel computation on a shared-nothing architecture, the IBM-SP2 machine. Section 6 presents the performance evaluation results on a wide variety of synthetic data sets, for large data sets and large number of dimensions, which highlight both the scalability of the algorithms and the quality of the clustering. Section 7 concludes the paper.

2 Related Work

Clustering has been extensively studied in statistics [6], machine learning [7], pattern recognition [2] and image processing . 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 [8]. 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 [9]. Recently, CURE [10] 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.

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. CLARANS [11], BIRCH [4] can be considered to be extensions of this method for databases. The k-modes algorithm extends the k-means paradigm to categorical domains [12]. Density based 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 [8]. 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. A non-grid based density based approach is presented in DBSCAN [3]. For each object of a cluster the neighborhood of a given radius has to contain at least a minimum number of points. Here both the radius and minimum number of points are input parameters which can significantly affect the quality of clustering. WaveCluster [13] 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. CLIQUE is another density and grid based approach for high dimensional data sets [5], which also detects clusters in the highest dimensional subspaces. Density based algorithms which use grids take 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. Another algorithm that finds clusters in subspaces has been reported in [14]. It relies on a statistical definition of clustering, that clustering has to partition the points in disjoint groups. This makes its appeal limited to the task of data mining and knowledge discovery in which interesting clusters can be overlapping. Further, it uses input parameters k, the number of clusters to be found, and l, the average dimensionality of the cluster, which are not possible to know apriori for real data sets. In our view this makes it unsuitable for the task of unsupervised learning to discover unknown and interesting clusters in data. A survey of parallel algorithms for hierarchical clustering using distance based metrics is given in [15]. These are more theoretical, PRAM algorithms, now more of academic interest than practical parallel algorithms for large data sets.

In the next few sections we describe the density and grid based algorithm and introduce the use of adaptive grids which greatly reduces the computation time and improve the quality of the clustering.

3 Density and Grid based Clustering

A cluster is a region that has a higher density of points than its surrounding region. A common way to find regions of high-density in the dataspace is based on grid cell densities [8]. In this approach a histogram is constructed by partitioning the data space into a number of non-overlapping regions or grids. The data space is partitioned and the number of points are found that map to each cell in the grid. This approximates the density of the data points. Equal length intervals are used in [5] 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.

Let $\mathcal{A} = \{A_1, A_2, \ldots A_d\}$ be a set of bounded, totally ordered domains. Let $\mathcal{S} = A_1 \times A_2 \times \ldots \times A_d$ be a d-dimensional numerical space, where $A_1, \ldots, A_d$ are attributes (dimensions) of $\mathcal{S}$. The input is a set of d-dimensional records. The space $\mathcal{S}$ is partitioned into a grid consisting of non-overlapping rectangular *units*. The units are obtained by partitioning every dimension into intervals of equal length in [5], which is given as an input parameter. A *cell*, c, is the intersection of one interval from each dimension, having a form $\{c_1, \ldots, c_d\}$, where $c_i = [l_i, u_i)$ is the interval in the partitioning of A_i. A record $r = (r_1, \ldots, r_d)$ is contained in the cell c, if $l_i \leq r_i < h_i$ for all c_i. A cell c is dense if the fraction of the total data points contained in the cell is greater than a user-defined threshold. We define θ as the threshold, which is a fraction of the total number of records present in the data set.

The clusters are unions of connected high density cells. Two k-dimensional cells are connected if they have a common face or if there is another cell which is connected to both. Clusters are expressed as a DNF expression on intervals of the domains A_i. For ease of understanding and interpretability of results by the end-user these expressions should be as simple as possible. Using this method to obtain clusters in subspaces, a histogram is created in all subspaces that counts the points contained in each unit. This approach is infeasible in high dimensional data as the number of subspaces is prohibitive.

3.1 Subspace Clustering

A bottom-up approach of finding dense units and merging them to find dense clusters in higher dimensional subspaces has been proposed in CLIQUE [5]. This algorithm is similar to the Apriori algorithm [16] 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" [5]. The algorithm starts by determining 1-dimensional dense units by making a pass over the data. 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 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. It is shown in [5] that the running time of this algorithm is exponential in the highest dimensionality of any dense cell. 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. The running time of the algorithm is $O(c^k + mk)$, where k is the highest dimensionality of any dense cell, m is the number of input records and c is a constant.

A pruning technique is used to decrease the number of candidate dense units in CLIQUE. This is done by using a minimum description length technique to find the dense units in interesting subspaces. It groups dense units that lie in the same subspace and finds out the coverage of the database that the dense units provide in each subspace. Subspaces which provide larger coverage are selected and the rest are pruned. According to [5], there is a caveat of missing some clusters by using this technique. If a cluster exists in k dimensions, then all of its projections in k dimensions are also clusters. In the bottom-up approach, all of these have to be considered if the cluster in k dimensions has to be found. Some of these may be in the pruned subspaces and hence the cluster will never get discovered. To maintain the high quality of clustering we do not use this pruning technique in our implementation of CLIQUE.

4 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 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.

We propose an adaptive interval size to partition the dimension depending on the distribution of data in the dimension. Using a histogram constructed by one pass of the data initially, we determine the minimum number of bins for a dimension. Contiguous bins with similar histogram values are combined to form larger bins. This partitions the dimension based on the data distribution. The bins and cells that have low density of data will be pruned limiting the eligible candidate dense units, thereby reducing the computation. The boundaries of the bins will also not be rigid, as in the uniform bins case and will be able to delineate cluster boundaries more accurately in each dimension. This improves the quality of the clustering result. Algorithm 1 describes the adaptive grid algorithm. First, we set the maximum value of the histogram in a dimension within a small window to reflect the window value. Adjacent windows are merged to form larger windows within a certain threshold. Window sizes of 5 are shown to work well in our experiments. A threshold of 20% is used to merge adjacent windows. A dimension containing a cluster will have a differential greater than 20% between the values of A_i which have the likelihood of being included in a cluster description and the ones which represent noise. Hence, we will get them in separate bins, and try to merge contiguous sections with relatively low densities in one bin. However, in dimensions where data is equi-distributed, this will result in one bin. We need to examine this further to be sure that this dimension does not contribute to clusters. Hence, we split such a dimension into a small fixed number of partitions, and collect statistics for these bins. This allows us to set a higher threshold since the likelihood of a cluster in this dimension is low, and limit the bins which contribute to the candidate dense units. This greatly reduces the computation time since we have been able to limit the bins from non-cluster dimensions, and eliminating them in some cases, from contributing to the computation. In the variable sized bins we use a threshold value for checking density set to be a percentage of the total number of records. Since the interval lengths of the dimension are such that a region with clusters will fall in a bin, a high value can eliminate the bins which have noise or low density.

4.1 Effect of Grids on Cluster Quality

Figure 1(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 1(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.

Algorithm 1 *Adaptive Grid Computation*

$|A_i|$ - Cardinality of A_i
for each dimension $A_i, i \in d$
    Divide $|A_i|$ 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 β
    /* 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 and set a threshold β' for it
end

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 clusters.

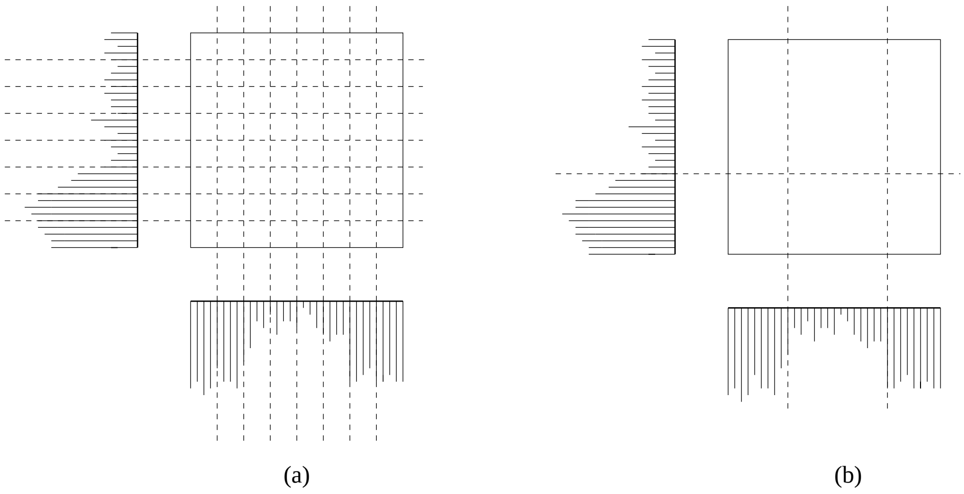


Figure 1: (a) Uniform grid size (b) Adaptive grid size

On the other hand, since MAFIA uses adaptive grid boundaries its cluster definitions are minimal DNF expressions and report the boundaries of clusters far more accurately. Figure 2 shows a cluster definition in two dimensions. The cluster reported by CLIQUE, $pqrs$, shown in Figure 2(a), loses the boundaries of the cluster. MAFIA 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 2(b) is $(l, y) \wedge (m, z) \wedge (n, y) \wedge (m, x) \wedge (m, y)$.

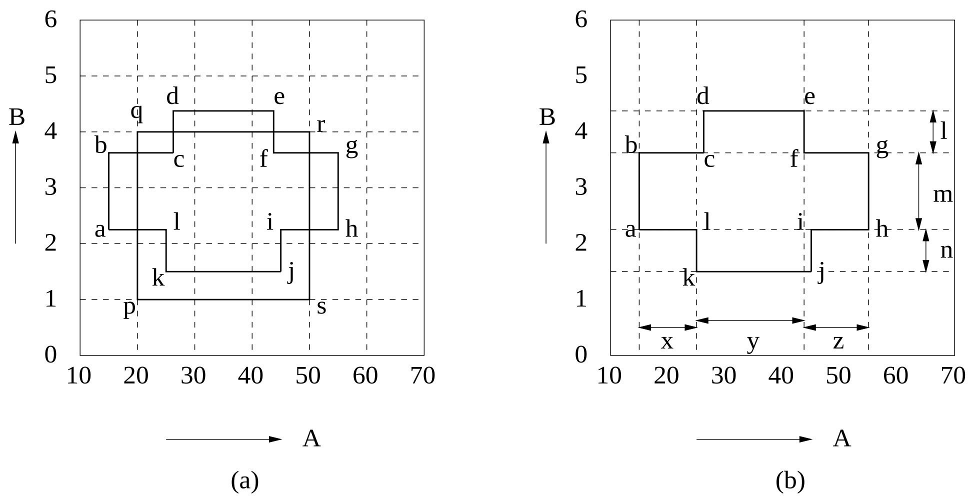


Figure 2: (a) Cluster discovered by CLIQUE (b) Cluster discovered by MAFIA

4.2 Performance of Adaptive Grids

The gains of using adaptive finite intervals are very impressive. As can be seen from the table in Figure 4(b) the performance of MAFIA due to the use of adaptive grids reduces the clustering time to 55.61 seconds from 2469.12 seconds that CLIQUE takes, which is 44 times faster. This result is for a 15 dimensional data set with 300,000 records with a 5 dimensional cluster. For larger data sets and higher dimensions, the effect of performance enhancements by using adaptive grids are more pronounced. Parallelism further enhances the appeal of MAFIA adding scalability for handling massive data sets, making it a platform for emerging clustering applications. Experimental results are presented in detail in Section 6.

5 Parallelism for Large Data Sets

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 3. 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. 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.

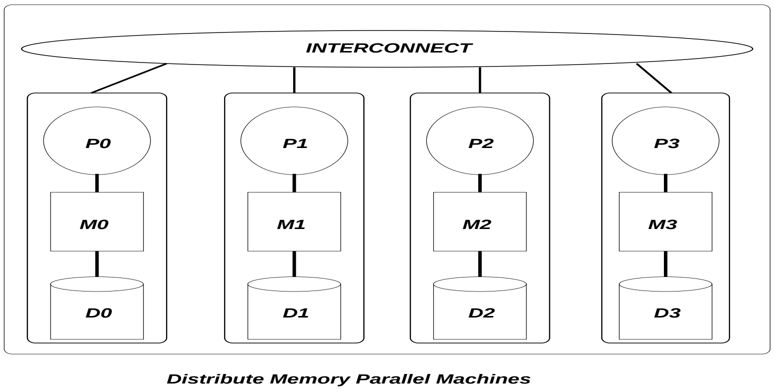


Figure 3: A shared-nothing architecture, P:Processor, M:Memory, D:Disk

5.1 Algorithm and Analysis

We develop a disk-based parallel and scalable algorithm for MAFIA which can handle large data sets and a large number of dimensions. This 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 this discussion we assume that the data is available on the local disk of a processor which it needs to process. In our set up, 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. This can be modified for other disk setups if necessary.

Each processor starts by building the histogram for each dimension with the data of size $\frac{N}{p}$ on its local disk, where N is the total number of records and p is the number of processors. A Reduce operation gathers the global histogram on each processor. Each processor then proceeds to determine the adaptive finite intervals for each dimension to fix their sizes. Each bin is considered to be a candidate dense unit. These are then populated by a pass on local data, which is read in chunks of B records. Another Reduce operation gets the global counts of the candidate dense units on all processors. The phase of populating the candidate dense units is the data parallel phase of the algorithm. The candidate dense units are then examined to determine which of them are actually dense by selecting the ones which have counts greater than all the thresholds of their projected dimensions. Each processor selects $\frac{1}{p}^{th}$ of the candidate dense units to do this in a completely task parallel manner. Having done this, each processor has the partial counts of the actual dense units which is combined to get the global counts on all processors with another Reduce communication

Algorithm 2 *Parallel MAFIA 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 $A_i, i \in d$
    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 ($k > 1$)
            Build candidate dense units in k from the dense units in $(k-1)$ which share the $(k-2)$-dimensional edges
        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
        /* Now divide the task of finding dense units between processors */
        Pick the appropriate $\frac{1}{p}^{th}$ portion of the populated candidate dense units to check if it qualifies to be a dense unit
        Reduce communication to get the global information of identified dense units
        Pick the appropriate $\frac{1}{p}^{th}$ of the dense units to find their bounds and build their data structures for use in the $(k+1)^{th}$ dimension
        Reduce communication to get the global information of the dense unit bounds
end

---

operation. Combinations of the dense units in dimension $(k-1)$ are then formed which share the first $(k-2)$ dimensions to be examined in the next higher dimension (k). The algorithm terminates when there are no more candidate dense units.

The running time of the algorithm is exponential in the distinct number of dimensions represented by the cluster subspaces in the data set, denoted by k'. Let N be the total number of records and p be the number of processors. Let α be the constant for communication and S be the size of messages exchanged among processors. 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 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 $O(\frac{N}{pB}k'\gamma)$. 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 be obtained by substituting $p = 1$ and $S = 0$, as there will be no communication.

In the next section we present performance results of our experiments with a variety of synthetic data sets.

6 Experimental Results

Our implementation of MAFIA 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) [17]. We use Message Passing Interface (MPI) [18] for communication between processors.

We report results of our experiments with MAFIA on a variety of synthetic 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.

6.1 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 [19]. To avoid this, we use a random number generator called the Inversive Congruential Generator which has better statistical properties [20]. 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.

6.2 Synthetic Data Results

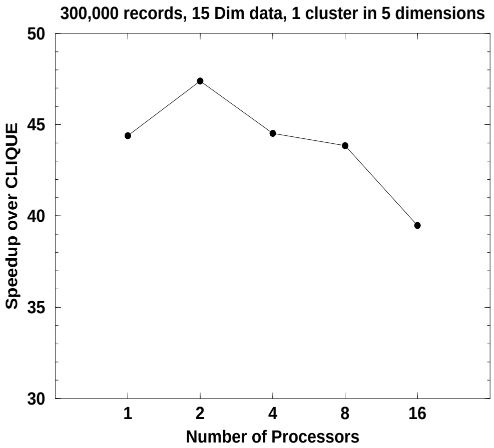
We present performance results of our adaptive grid technique in MAFIA and compare it with our implementation of CLIQUE. The results presented here are on our synthetic data sets as efforts to procure the data sets used in CLIQUE were unfruitful. Further, we show results for scalability of MAFIA with 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 MAFIA observing both scale up and speedup.

6.2.1 Effect of using Adaptive Grids

Figure 4(a) shows the speedup that we have obtained over CLIQUE on different number of processors. The results shown are for a database with 300,000 records with data in 15 dimensions. There is one cluster embedded in a 5 dimensional subspace. The algorithm sets θ to 1% in the cluster dimension and 24% for dimensions with equi-distributed data in MAFIA. 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. Note that we do not use pruning as defined in CLIQUE to maintain the correctness of the clustering. We set the number of bins in each dimension to be 10 as used in CLIQUE. We see good parallelization for both CLIQUE and MAFIA from the table in Figure 4(b). It can be seen from Figure 4(a) that MAFIA performs 40 to 50 times better than CLIQUE for this data set. We expect the speedups obtained for data sets with much larger dimensionality to be even more impressive. This is due to the very nature of our algorithm which decides upon a minimum set of bins for each dimension based on its *interestingness* as observed from the histogram of the data in every dimension. This results in a set of candidate dense units whose cardinality is much lower than the one obtained by setting equal number of bins in all dimensions. The latter results in an explosion of candidate dense units in a very high dimension space. We were unable to run CLIQUE in a much higher dimension space for a large number of records in a reasonable amount of time (8 hours). Hence, we are unable to present comparisons for large data sets in a high dimensional space. We present results only for MAFIA for these data sets. The table in Figure 4(b) shows the comparative execution times of MAFIA and CLIQUE for the data set used in Figure 4(a).

6.2.2 Effect of Subspace Overlap in Clusters

Another important point to note here is the effect of subspace overlap on the computation time. Consider an example data set with 1 million records in 20 dimensions which has 3 clusters each of 5 dimensions, embedded in a different subspace. The computation time for this data set heavily depends on the number of distinct cluster dimensions in the 3 subspaces put together as the number of candidate dense units explode when we have to consider more combinations of dimensions. This effect would be even more pronounced in CLIQUE as it treats all dimensions equally without exploiting the data distribution in each dimension. For this reason it is not possible to make a



(a)

	Number of Processors				
	1	2	4	8	16
MAFIA	55.61	27.95	14.93	7.71	4.68
CLIQUE	2469.12	1324.51	664.65	338.19	184.36

(b)

Figure 4: (a) Speedup of MAFIA over CLIQUE (b) Comparative execution times (in seconds) for 300,000 records in a 15 dimensional space with 1 cluster in 5 dimensions

direct comparison to the results reported in [5] as it does not provide enough information about the subspace overlap of its clusters. Table 1 shows the effect of subspace overlap in the embedded clusters on the execution times of MAFIA, which calculates adaptive finite intervals in every dimension. The threshold values were chosen to be the same as those reported for the data set in Figure 4(a).

Table 1: Comparative execution times (in seconds) on 16 processors of MAFIA for a data set with 1 Million records in a 30 dimensional space with 3 clusters in 5 dimensions each

	Cluster Dimensions	Total No. of Distinct Dimensions	Total Number of Dense Units Processed	Time
MAFIA (overlapped subspaces)	{1,3,7,8,10} {1,3,6,8, 9} {1,4,5,7, 9}	9	8,645	242.25
MAFIA (non-overlapping subspace)	{3,4,6,10,11} {1,2,5,7,8} {13,15,17,18,19}	15	345,823	10999.42

6.2.3 Scalability with Database Size

Figure 5 shows the results for scalability with the size of the database. In this experiment we used 20 dimensional data and varied the number of records from 1.3 million to 10.4 million. There were 5 clusters embedded in 5 different subspaces each of 5 dimensions. The experiment reported here

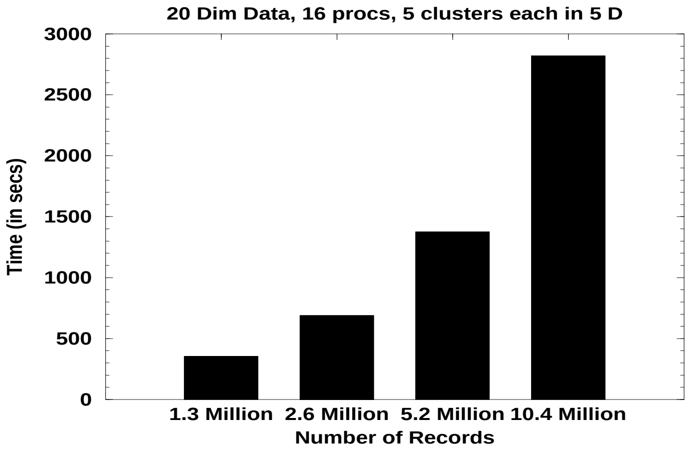


Figure 5: Scalability with increasing database size

is run on 16 processors. Similar behavior is observed on other number of processors. The threshold percentage, θ, was set to 1% in the cluster dimensions by the algorithm. In the dimensions which we found to have data equi-distributed, we divide the dimension into a default number of bins and correspondingly the algorithm sets the value of θ. In this experiment we set the default number of bins to be 5 and correspondingly threshold percentage for that dimension, θ, was automatically set to 24%. The graph clearly shows the relationship between time and the database size to be almost linear. This linear behavior is exactly what is expected as the number of passes over the database depends only on the dimensionality of the cluster. An increase in the size of database just means that more data has to be scanned on every pass over the database while finding the dense units and this results in a linear increase in time.

6.2.4 Scalability with Dimensionality of Data

In Figure 6 we see that MAFIA scales very well with the increase in the data dimension. The results presented are for a data set with 250,000 records with 3 clusters each in a five dimensional subspace and with a total of 9 distinct dimensions. The results are obtained by using 16 processors. Similar behavior is observed on other number of processors. The threshold values were set to 1% in the cluster dimensions and 24% in the dimensions with equi-distributed data by the algorithm. The near linear behavior is due to the fact that our algorithm makes use of data distribution in every dimension and only depends on the maximum cluster dimensionality and the subspace overlap of the clusters. CLIQUE depends on subspace overlap and also on the data dimensionality. Hence it exhibits a quadratic behavior with respect to the data dimensionality as reported in [5].

6.2.5 Scalability with Increasing Cluster Dimensionality

Figure 7 shows the scalability observed with increasing cluster dimensionality in MAFIA. The results reported are for a data set in 50 dimensions with 650,000 records containing 1 cluster.

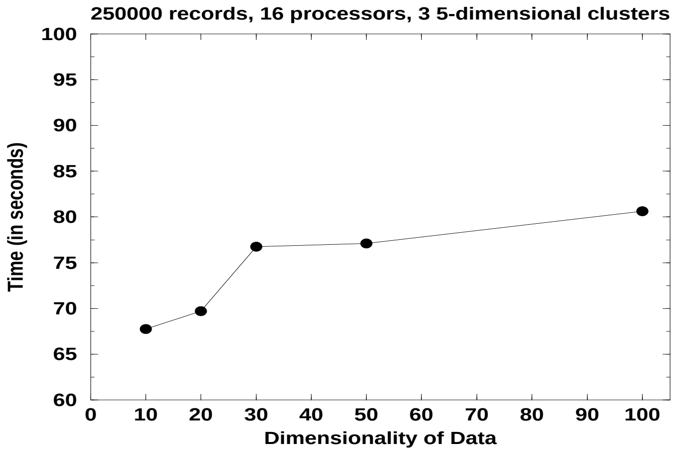


Figure 6: Scalability with increasing dimensionality of data

The threshold values were set by the algorithm to 1% in the cluster dimensions and 24% in the dimensions with equi-distributed data. The results reported are on 16 processors. Results show that the time increase with cluster dimensionality reflects the time complexity of the algorithm, $O(\frac{N}{p}k + \frac{N}{pB}k\gamma + \alpha Spk + c^k)$, 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.

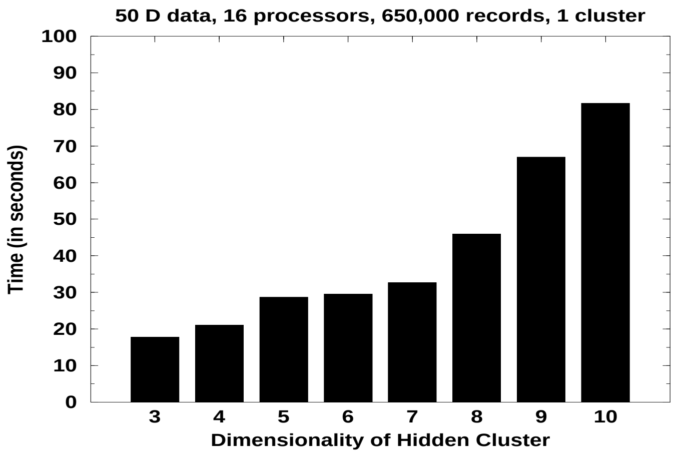


Figure 7: Scalability with increasing cluster dimensionality

6.2.6 Quality of Results

We compare the quality of the results obtained by MAFIA with those of CLIQUE shown in Table 2. In this experiment we report the results for a relatively small data set with 400,000 records with data in 10 dimensions. The data set consisted of 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 give an arbitrary number of bins in each of the 10 dimensions (with a minimum of 5 bins to a maximum of 20 bins per dimension) and set the threshold in each dimension to 1%. In the first case CLIQUE reported the correct dimensions of the 2 clusters, but, however it was only able to partially detect the 2 clusters and large parts of the clusters were thrown away as outliers. In the second run, with 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. It is also prohibitive to experiment with different number of bins, for each run on 16 processors took about 3 hours for this particular data set. Also for a real life data set validation of the results obtained would be a very hard task and bin selection would be a non trivial problem.

When we ran MAFIA with the same data set on 16 processors, both the clusters and the cluster boundaries in each dimension were accurately reported. The reported cluster boundaries differed from the user specified cluster description by a maximum of 1 unit (of size 1). Greater resolution (say, units of size 0.1) can also be used to calculate the adaptive bin boundaries since the run-time of the adaptive grid algorithm is negligible. This will allow finer cluster descriptions.

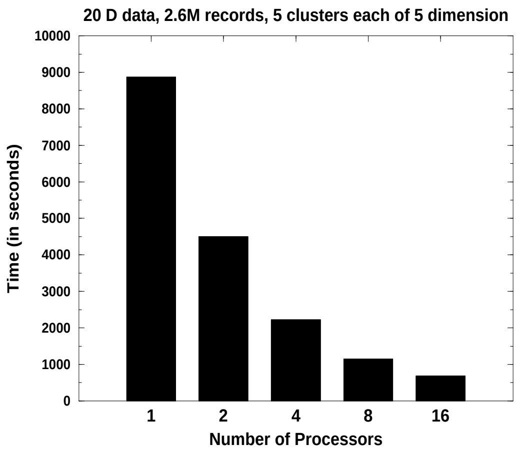
Table 2: Comparison of the quality of results obtained by MAFIA and CLIQUE

	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}
MAFIA	{1,7,8,9} {2,3,4,5}	{1,7,8,9} {2,3,4,5}

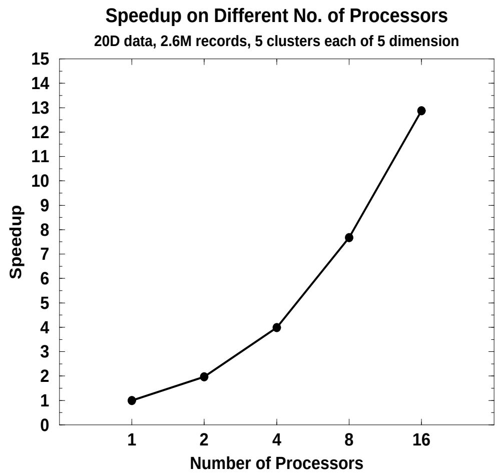
In the other experiments that we conducted with MAFIA, in very few cases, few dimensions into which random data had been pumped in was also reported as part of some cluster. However, upon close examination of the data in those dimensions, the histograms revealed to us that the random dimension had been divided into just two largely unequal sized bins due to a *not so random* distribution of numbers in that dimension, an artifact of the data generation procedure. However, this would not be a concern in real life data sets wherein adaptive finite intervals would be highly efficient.

6.2.7 Parallel Performance

Figure 8 presents the times obtained by the parallelization of MAFIA. The results presented are for a 20 dimensional data set with 2.6 million records containing 5 clusters each in a different 5 dimensional subspace. The threshold percentages were automatically set by the algorithm in various 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 fact that the algorithm is heavily data parallel and time for computation goes down linearly with the increase in the number of processors. Also the additional task parallelism that we have introduced in finding the dense units and building up the candidate dense units for a higher dimension contributes to the speedup, but this contribution is negligible as compared to the gains from data parallelism. Table 3 summarizes the various components of total time obtained by running the data set on different number of processors. The bulk of the time is taken in populating the candidate dense units which is completely data parallel. It can be seen that the communication overheads introduced due to the parallel algorithm is negligible as compared to the total time. Also the time taken to build the initial histogram is a very small percentage of the total time.



(a)



(b)

Figure 8: (a) Parallel run times of MAFIA (b) Speedup observed

7 Conclusions

In this paper we presented MAFIA, an efficient algorithm for subspace clustering using a density and grid based approach, which uses adaptive finite intervals. This performs an order of magnitude better than existing formulations of the method and also improves the quality of clustering. A scalable parallel algorithm is developed which can handle massive data sets and makes it a viable algorithm for discovering clusters in high dimensional data for large scale data mining applica-

Table 3: Timing Results (in seconds) of parallel MAFIA on 2.6 million records, 20 dimensional data with 5 clusters, each of 5 dimensions

Components	Number of Processors				
	1	2	4	8	16
Total time for populating the candidate dense units	8486.33	4302.03	2134.49	1035.27	503.18
I/O Time	287.01	130.24	20.67	3.82	1.86
Total communication time	0.000	1.536	2.0395	3.044	3.514

tions. Experimental evaluation on a variety of synthetic data sets, with varying dimensionality and database sizes show the gains in performance and quality of clusters. The use of adaptive grids in MAFIA leads to a 40 times improvement over CLIQUE, for a 15 dimensional data set with 300,000 records. Gains on higher dimensional data and larger data sets has been observed to be even more dramatic.

References

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

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

[3] 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.

[4] 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.

[5] 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.

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

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

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

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

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

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

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

[20] 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.