A Scalable Parallel Subspace Clustering Algorithm for Massive Data Sets*

Harsha S Nagesh
ECE Department
Northwestern University
harsha@ece.nwu.edu

Sanjay Goil
Performance Technology Group
Sun Microsystems Inc.
sgoil@eng.sun.com

Alok Choudhary
ECE Department
Northwestern University
choudhar@ece.nwu.edu

Abstract

Clustering is a data mining problem which finds dense regions in a sparse multi-dimensional data set. The attribute values and ranges of these regions characterize the clusters. Clustering algorithms need to scale with the data base size and also with the large dimensionality of the data set. Further, these algorithms need to explore the embedded clusters in a subspace of a high dimensional space. However, the time complexity of the algorithm to explore clusters in subspaces is exponential in the dimensionality of the data and is thus extremely compute intensive. Thus, parallelization is the choice for discovering clusters for large data sets. In this paper we present a scalable parallel subspace clustering algorithm which has both data and task parallelism embedded in it. We also formulate the technique of adaptive grids and present a truly un-supervised clustering algorithm requiring no user inputs. Our implementation shows near linear speedups with negligible communication overheads. The use of adaptive grids results in two orders of magnitude improvement in the computation time of our serial algorithm over current methods with much better quality of clustering. Performance results on both real and synthetic data sets with very large number of dimensions on a 16 node IBM SP2 demonstrate our algorithm to be a practical and scalable clustering technique.

1. Introduction

Clustering has been extensively studied in statistics, machine learning [12], pattern recognition and image processing [8]. Intuitively, clustering techniques find *interesting* and previously unknown patterns in large scale data, embedded in a large multi-dimensional space and are applied to a wide variety of problems like customer segmentation based on similarity of buying interests [17], detection of clusters in geographic information systems [13], etc. Clustering algorithms need to efficiently scale up with the dimensionality of data sets and also with the data base size. Noise present with data makes cluster detection harder. Further, clusters embedded in a subspace of the total data space, result in an explosion in the search space which is exponential in the data dimension. Parallelization of the clustering process adds the much needed computational power for these algorithms to be applied to massive financial data sets and large scale scientific data. Effective representation of the detected clusters is as important as cluster detection and improves its usability. Most of the earlier works in statistics and data mining [7, 19] operate and find clusters in the whole data space. Most clustering algorithms [13, 1, 19] require user input of several parameters like the number of clusters, average dimensionality of the cluster, etc. which are not only difficult to determine but are also not practical for real-world data sets. The output of these clustering algorithms are very sensitive to the input parameters.

*This work was supported by the Department of Energy's Accelerated Strategic Computing Initiative (ASCI) program under a subcontract No W-7405-ENG-48 from Lawarence Livermore national Laboratories and by NSF CDA-9703228.

We use a grid and density based approach for cluster detection in subspaces. Density based approaches regard clusters as higher density regions than their surroundings. The quality of results and the computation requirements heavily depend on the number of bins in each dimension. Larger bin sizes result in poor cluster quality, while finer bins result in an enormous amount of computation. Hence, determination of bin sizes automatically based on the data distribution greatly helps in finding correct clusters of high quality and reduces the computation substantially. However, subspace clustering algorithms need to explore all possible subspaces for embedded clusters in a bottom-up algorithm resulting in a time complexity which is exponential in the data dimension. Thus parallelization is the choice for grid based subspace clustering algorithms for massive data sets.

In this paper we present pMAFIA (for Merging of Adaptive Finite Intervals), a scalable parallel subspace clustering algorithm using adaptive computation of the finite intervals (bins) in each dimension, which are merged to explore clusters in higher dimensions. The parallelization strategy involves both task and data parallelism with negligible communication overheads. Adaptive grid sizes improve the clustering quality by concentrating on the portions of the data space which have more points and thus are more likely to be part of a cluster region enabling minimal length DNF (disjunctive normal form) expressions, important for interpreting results by the end-user. Further, pMAFIA requires no user input, making it a completely un-supervised data mining algorithm. We describe recent work on clustering techniques in databases in Section 2. Density and grid based clustering is presented in Section 3 and we also describe our approach of using adaptive grids. Section 4 describes the parallel algorithms for detecting clusters and also explains the task and data parallel algorithms followed by their analysis. Section 5 presents the performance evaluation on a wide variety of synthetic and real data sets with large number of dimensions, highlighting both scalability of the algorithms and the quality of the clustering. Section 6 concludes the paper.

2. Related Work

Clustering algorithms have long been studied in statistics and databases. k-means, k-mediods, CLARANS [14], BIRCH [19], CURE [9] are some of the earlier works. Wavecluster [16], a density and grid based approach using wavelet transform on the multi-dimensional space, is computationally efficient but applicable to only low dimensional data. However, none of the above algorithms detect clusters in subspaces. PROCLUS [1], 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. Density and grid based approaches regard clusters as regions of data space in which objects are dense and are separated by regions of low object density (noise) [10]. The grid size determines the computations and the quality of the clustering. CLIQUE, a density and grid based approach for high dimensional data sets [2], detects clusters in the highest dimensional subspaces. It takes the size of the grid and a global density threshold for

2000 International Conference on Parallel Processing (ICPP'00)
0-7695-0768-9/00 $10.00 @ 2000 IEEE

clusters as input parameters. The computation complexity and the quality of clustering is heavily dependent on these parameters. ENCLUS [4], an entropy based subspace clustering algorithm requires a prohibitive amount of time to just discover interesting subspaces in which clusters are embedded. It also requires input of entropy thresholds which is not intuitive for the user. A survey of parallel algorithms for hierarchical clustering using distance based metrics is given in [15]. These are more theoretical PRAM algorithms. Recently, k-means algorithm has been parallelized [5], 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.

3. Density and Grid based Clustering

Density based approaches regard clusters as higher density regions than their surroundings. A common way of finding high-density regions in the data space is based on the grid cell densities [10]. 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 [2] 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 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 [2] 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.

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. A bottom-up approach of finding dense units and merging them to find dense clusters in higher dimensional subspaces has been proposed in CLIQUE [2]. 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 [2] 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. However, this method of combining dense units does not explore all possible candidate dense cells. 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 [2]. 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. 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. In [2] 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 [2] 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.

3.1. Adaptive Grids

We propose an adaptive interval size in which bins are determined based on the data distribution in a particular dimension. The size of the bin and hence number of bins in each dimension in turn determine the computation and quality of clustering. Finer grids leads to an explosion in the number of candidate dense units, while coarser grids leads to fewer bins, and regions with noise data might also get propagated as dense cells. Also, a user defined uniform grid size may fail to detect many clusters or may yield very poor quality results. 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.

Algorithm 1 *Adaptive Grid Computation*

D_i - Domain of A_i
N - Total number of data points in the data set
a - Size of a generic bin
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 β
    /* Single bin implies 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

3.2. Adaptive Grids and Quality of Clustering

Figure 1.1(a) illustrates the uniform grid used in CLIQUE and, as seen, generates many more candidate dense units than an adaptive grid illustrated in Figure 1.1(b). CLIQUE also uses a greedy algorithm as a post-processing phase to generate the minimal description length of the clusters to make the cluster definitions more amenable to the end-user. It 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. On the other hand, since pMAFIA uses adaptive grid boundaries its cluster definitions are minimal DNF expressions and report the boundaries of clusters far more accurately. Figure 1.2 shows a cluster definition in two dimensions. The cluster reported by CLIQUE, *pqrs*, shown in Figure 1.2(a), loses the boundaries

2000 International Conference on Parallel Processing (ICPP'00)
0-7695-0768-9/00 $10.00 @ 2000 IEEE

of the cluster. pMAFIA develops grid boundaries very close to the boundaries of the cluster and reports $abcdefghijkl$ shown in Figure 1.2(b) as the cluster with the DNF expression $(l, y) \wedge (m, z) \wedge (n, y) \wedge (m, x) \wedge (m, y)$.

4. pMAFIA Implementation

We introduce a scalable parallel formulation of the subspace clustering algorithm incorporating both data and task parallelism on a distributed memory architecture. 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 leading to data 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. To achieve good parallelization, the overhead of communication has to be reduced by allocating tasks to processors such that relatively few small messages are exchanged. 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.

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

---

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 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. 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.

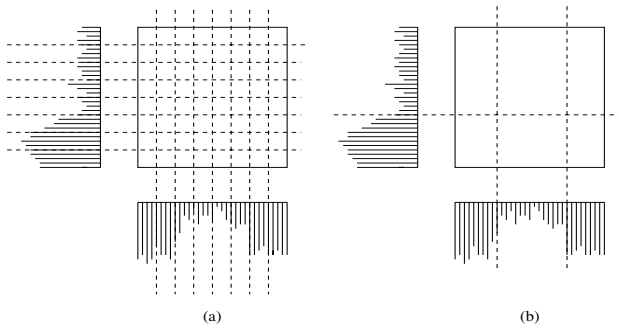
4.2. Task Parallelism

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.

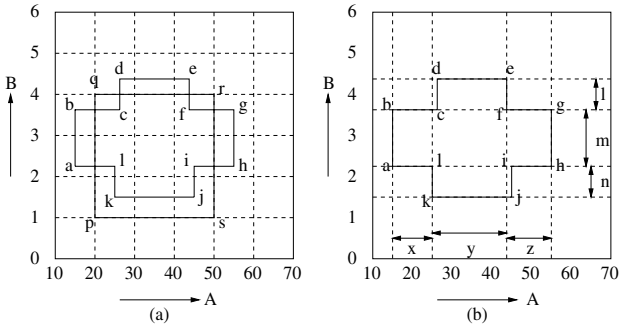
4.3. Building Candidate Dense Units

Figure 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 are 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 huge amount of 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

2000 International Conference on Parallel Processing (ICPP'00)
0-7695-0768-9/00 $10.00 @ 2000 IEEE



(1.1a) Uniform (1.1b) Adaptive



(1.2a) CLIQUE (1.2b) pMAFIA

Figure 1. (1) Grid Size (2) Cluster Discovered

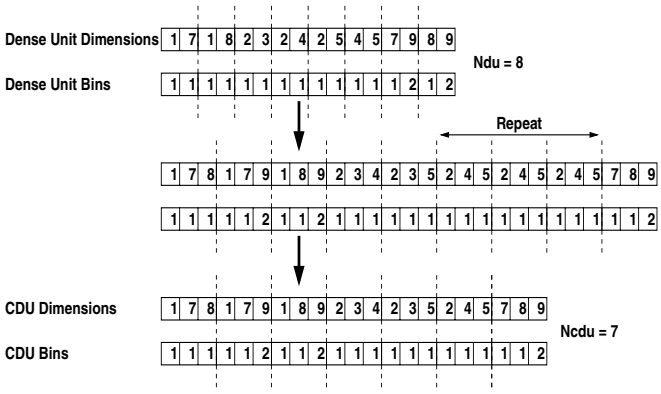


Figure 2. Building CDUs. (Dimension, k = 3)

$0 < n_1 < n_2 < \ldots < n_{p-1} < Ndu$ 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, we have

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

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.

It can be seen from Figure 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.

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; τ - 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

4.4. Identification of Dense Units

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

2000 International Conference on Parallel Processing (ICPP'00)
0-7695-0768-9/00 $10.00 @ 2000 IEEE

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.
}

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 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 un-supervised 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 with greater computation time and better cluster quality. 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.

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 the 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

4.5. 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

2000 International Conference on Parallel Processing (ICPP'00)
0-7695-0768-9/00 $10.00 @ 2000 IEEE

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.

5. Performance Evaluation

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). We use Message Passing Interface (MPI) for communication between processors.

5.1. Data Sets

We created a data generator to produce the data sets used in our experimental results. The data generator takes from the user the extents of the cluster 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 and clusters can have arbitrary 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. All dimensions are scaled to lie in the range $[0, \ldots, 100]$. Data points are generated such that each unit cube, 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 as used in [2], ensures that we have a cluster exactly as defined by the user and helps to validate the results. For the remaining attributes we select a value at random from a uniform distribution over the entire range of the attribute. We use a better random number generator called the Inversive Congruential Generator [6] as long sequences of Unix random number generators (LCGs) exhibit regular behavior by falling into specific planes. An additional 10% noise records is added 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. Also, user specified cluster definition is permuted to ensure no dependency on the order of input records.

5.2. Experimental Results

We present performance results of pMAFIA in terms of the speedups on different number of processors, scalability of the algorithm with the database size and scalability of the algorithm with dimensionality of the data and cluster dimensionality. Further, we compare CLIQUE with pMAFIA, followed by results of the improvement in quality of the clustering obtained by our algorithm. 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.

5.3. Parallel Performance

Figure 3 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.

Table 1. Execution times (in secs)

	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

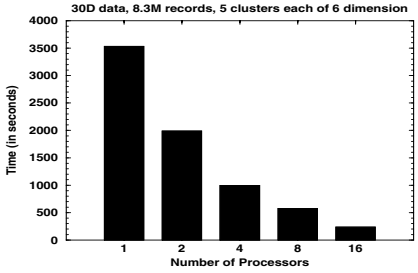


Figure 3. Parallel run times of pMAFIA

5.4. Effect of using Adaptive Grids

Figure 4 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 1. Figure 4 shows that pMAFIA performs 40 to 80 times better than CLIQUE for this data set. The results presented for CLIQUE are using the CDU generation algorithm in [2]. This speedup is due to the minimal set of bins in each 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. Table 1 shows the comparative execution times of pMAFIA and CLIQUE for the data set used in Figure 4.

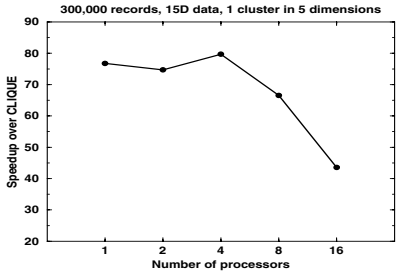


Figure 4. Speedup of pMAFIA over CLIQUE

5.5. Adaptive Grids and Computation 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 [2] where candidate dense units in dimension k are formed

2000 International Conference on Parallel Processing (ICPP'00)
0-7695-0768-9/00 $10.00 @ 2000 IEEE

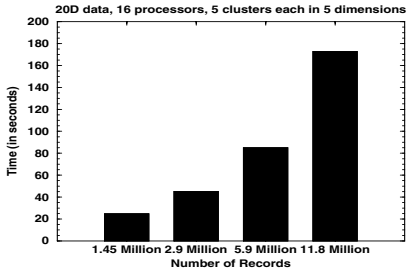


Figure 5. Scalability with database size

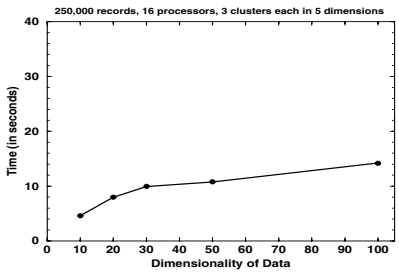


Figure 6. Scalability with Data Dimension

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 2 shows the number of CDUs (Ncdu) and the number of dense units (Ndu) generated in this experiment. Results presented for [2] 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 at least 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 [2] 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 2. 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 2. CDUs generated: pMAFIA, 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

5.6. Scalability with Database Size

Figure 5 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.

5.7. Scalability with Data and Cluster Dimensions

In Figure 6 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 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 [2].

Figure 7 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, which is exponential in the number of distinct cluster dimensions. Similar behavior is observed on other number of processors.

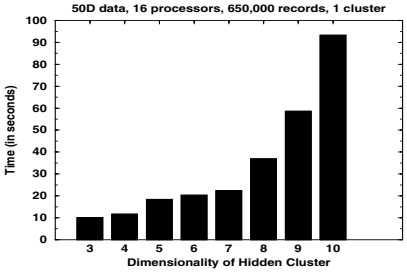


Figure 7. Scalability with Cluster Dimension

5.8. Quality of Results

We compare the quality of the results obtained by pMAFIA with those of CLIQUE, shown in Table 3. 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 3. Quality of Clustering

	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}

5.9. Real Data Sets

We applied pMAFIA on three 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 un-supervised 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.

2000 International Conference on Parallel Processing (ICPP'00)
0-7695-0768-9/00 $10.00 @ 2000 IEEE

Table 4. Clusters Discovered in DAX Data Set

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

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 [18]. 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 4. The results reported are with 8 processors taking 8.16 seconds.

2. **Ionosphere Data**

   We applied pMAFIA to the radar data that was collected by a system in Goose Bay, Labrador [3]. 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 [1] 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. Further, [1] also requires the user to specify k, the number of clusters in the data set which cannot be known apriori for real data sets.

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 [11] over a period of 18 months. During this period 72916 users entered a total of $2,811,983$ (≈ 2.8 Million) 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 clusters all of dimension 2 in just about 28 seconds on a 400 MHz Pentium II processor. Clusters discovered by pMAFIA revealed interesting information about which set of movies were rated most by which set of users. The experiment reveals the scalability with respect to the data size of pMAFIA on real data as seen from the run times in Table 5.

Table 5. Parallel Performance.

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

6. Conclusions

In this paper we presented pMAFIA, an efficient parallel algorithm for subspace clustering using a density and grid based approach with adaptive finite intervals. This performs two orders of magnitude better than CLIQUE and also improves the quality of clustering greatly as compared to both CLIQUE and PROCLUS. pMAFIA requires no user input, which makes it a totally un-supervised data mining algorithm. Experimental evaluations on a variety of synthetic and real data sets, with varying dimensionality and database sizes, show the gains in performance and quality of clusters. The use of adaptive grids in pMAFIA leads to large improvements over CLIQUE, as large as 80 times better for some data sets. Near linear speedups have also been obtained with very small communication overheads in the parallel formulation.

References

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

[2] 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.

[3] C. Blake and C. Merz. UCI repository of machine learning databases. 1998. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[4] C. Cheng, A. 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.

[5] I. Dhillon and D. 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.

[6] J. Eichenauer-Herrmann and H. 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.

[7] 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.

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

[9] 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.

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

[11] P. McJones. Digital Equipment Corporation, Systems Research Center. 1997. http://www.research.digital.com/SRC/.

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

[13] R. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. *Proc. 20th Int. Conf. on Very large Data Bases, Santiago, Chile*, pages 144–155, 1994.

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

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

[16] 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.

[17] L. Ungar and D. Foster. Clustering methods for collaborative filtering. *AAAI Workshop on Recommendation Systems*, 1998.

[18] 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.

[19] 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.