Clustering Very Large Datasets using a Low Memory Matrix Factored Representation

David Littau *littau@cs.umn.edu*
Daniel Boley *boley@cs.umn.edu*

University of Minnesota
Department of Computer Science
200 Union St. S.E.
Minneapolis, MN 55455 USA

preliminary version of paper in *Computational Intelligence* 25(2):114-135, May 2009

Abstract

A scalable method to cluster datasets too large to fit in memory is presented. This method does not depend on random subsampling, but does scan every individual data sample in a deterministic way. The original data is represented in factored form by the product of two matrices, one or both of which is very sparse. This factored form avoids the need to multiply together these two matrices by using a variant of the PDDP algorithm which does not depend on computing the distances between the individual samples. The resulting clustering algorithm is Piecemeal PDDP (PMPDDP), in which the original data is broken up into sections which will fit into memory and clustered. The cluster centers are used to create approximations to the original data items, and each original data item is represented by a linear combination of these centers. We evaluate the performance of PMPDDP on three real data sets, and observe that the quality of the clusters of PMPDDP is comparable to PDDP for the datasets examined.

1 Introduction

There are many applications in which extremely large datasets need to be explored, but which are much too large to be processed by traditional methods. Often it is desired to compute an unsupervised clustering of the data in order to discover new patterns, anomalies, or other unanticipated features buried within the dataset. We seek a method which is capable of clustering datasets too large to fit in memory at once, but which scans through the entire dataset at least once. We do not wish to depend on random subsampling which could miss anomalies or unusual artifacts.

To satisfy this wishlist, we would like to compute a compact representation of the entire dataset which takes much less memory than the original, yet approximates the original dataset sufficiently well to allow accurate clustering. Each data item would need a unique representative within the compact representation, no sampling would be used, and the method would not be dependent on the order in which the data are presented. All of the representatives for the entire data set would have to be available in memory at once during clustering. If the clustering is to take place in a reasonable amount of time, then the clustering method used should take advantage of the form of the representations.

To achieve these goals, we present a method based on building up a sparse product representation of the entire dataset, in which each data sample is represented by a linear combination of a few sparse representative vectors. The entire dataset will be represented by the product of a matrix of a few representative vectors and a matrix of coefficients defining the linear combination for the corresponding original data vector. One or both of these matrices will be very sparse, so we leave the representation in matrix product form taking little space. This factored form is called the *Low Memory Factored Representation* (LMFR), introduced in [21]. To take full advantage of this product form, it is necessary to avoid traditional methods that compute distances between individual data vectors, such as agglomeration or k-means. Hence we use the fast, scalable clustering Principal Direction Divisive Partitioning (PDDP) algorithm [5] to build this representation and generate the final clustering, because it is uniquely able to take advantage of the matrix product representation without calculating the product explicitly. The PDDP method uses a Lanczos-based solver which requires only matrix vector products in which the matrix can be left in the given factored form, and does not require the calculation of explicit distances between individual data samples. As such, the data are never explicitly reconstructed during the clustering process.

The sparse product representation is constructed using the centers from clustering as a basis [here we use the term "basis" in the colloquial sense of "starting collection", not in the formal sense of a set of linearly independent vectors]. The original data are broken into smaller pieces, called *sections*, which will fit into memory. Each section is clustered separately using PDDP, and the centers of the clusters are extracted. The centers are then used to compute a least-squares approximation to the data comprising the section using a small fixed number of the centers closest to each data item. This can be done in parallel if desired, since the representation of each section is generated independent from all other sections. Once there is an approximate representation of each section of data, the sparse product representations for all sections are collected and used to perform one final clustering operation. This final clustering will be a approximate clustering of the entire original data set. This method is called Piecemeal PDDP (PMPDDP) [22]. The piecemeal approach is a framework used here to design an effective clustering algorithm, but the resulting data representation (LMFR) can be used for many data mining tasks [21]. When used to design a clustering algorithm, any clustering method could be applied to each section in principle, but we made the specific choice here to use PDDP since it is an example of a scalable deterministic clustering method. If an alternative scalable clustering algorithm is used to carry out the initial partitioning of the data into sections, which would then represent some sort of affinity groups among the data, then other annotations could be applied to the sections, as proposed in [28]. But in the methods proposed here, the individual sections do not represent any kind of affinity group buried within the data.

The approximations to the original data can be made as accurate as available memory allows. The number of centroids calculated for each section of data, and the number of centroids used to approximate each original data item, are completely user-determined. Generally, the more centroids calculated for each section, and the more centers used to approximate each original data point, the more accurate the final representation. However, more memory will be necessary to contain a more accurate approximation as compared to a cruder approximation.

Reducing the original data to its approximate representation allows PDDP to process the dataset in a reasonable amount of time using relatively modest amounts of memory. The method examines every data item during the process of creating the classification, and will not miss or ignore the presence of outliers. It can do the most expensive work without having to examine the entire data

set at once. In [23] we showed how it is easy to update this factored representation by replacing old data with more recent data, turning just about any algorithm into one suitable for streaming data applications. The amount of memory required by the low memory representation can be set by the user, and the only tradeoff is the accuracy of the approximation and, to a lesser extent, the accuracy of the clustering. There are no parameters which need to be selected or tuned. The ultimate objective of this work is to develop a method that will scale well to very large data sets while retaining a small memory footprint.

2 Previous Work

The problem of clustering large data sets has been addressed often in the past. Hierarchical agglomeration [16] and k-means [13, p201] are two popular starting points for developing scalable clustering methods. Hierarchical agglomeration assembles clusters by recursively joining the two closest clusters, where the distance is computed with respect to all points in all clusters. K-means starts with a fixed set of cluster centers, and goes through an iterative process of assigning the data to the closest center and then using the data in a given cluster to calculate a new center.

Random sampling is a very scalable alternative, as proposed in [19, 12, 1, 11], but the results are not deterministic, and in some cases the subsampling could miss outliers. Hence these methods might not be suitable in situations where it is required to sample every data item in a consistent and deterministic manner.

Two methods based on hierarchical agglomeration are Scatter/Gather [9] and CURE [17]. Scatter/Gather reduces the running time by only applying agglomeration to smaller sets of the data, and then by agglomerating the centers resulting from the previous operation. The method is $O(p^2)$, where p is the number of data points agglomerated at any given time, and careful choice of parameters is necessary to insure that the overall running time does not change from rectangular to quadratic. CURE reduces the running time by measuring the inter-cluster distance using a small fixed number of *well-scattered* points, and by sampling the data. This reduces the overall running time to $O(p^2 \log p)$, where p is the total number of well-scattered points chosen to represent the data.

CLARANS [25] is a k-centers algorithm which finds a set of k cluster *medoids*, which are like the centroids in k-means except they are members of the data set. Various clusterings are examined by replacing one existing medoid by a randomly chosen medoid. Only a small number of alternate clusterings are examined in order to keep the running time rectangular.

The ordinary k-means algorithm often terminates with configurations which are not a global optimum, and the resulting issues, including the need to restart the algorithm, is explored in [8]. This method is designed to find good initial centers for large data sets, such that the k-means algorithm will not have to be repeated. It does this by clustering independent samples of the data, and then clustering the resulting centroids. The final set of centroids becomes the starting centers for k-means. The applicability to large data sets can be further enhanced by the method outlined in [7]. The method applies k-means to as much data as can fit into memory and uses the cluster centers to represent the data. The cluster centroids are used to represent the data which has been clustered, new data is loaded into memory, and the process continues. Both solutions to scaling k-means to large data sets retain the same $O(kn)$ running time that k-means needs.

BIRCH [30] is a method which is used to pre-cluster large data sets. BIRCH incrementally groups the data as tightly as possible based on similarity in their attributes, and in the process

constructs as many representatives of the original data as the available memory will contain. If the amount of space taken by the data runs out during the BIRCH process, the tightness of the groupings is relaxed, the groupings are internally re-assigned, and the processing of the data continues. Since BIRCH examines every data point, it runs in $O(n)$ time. Another algorithm must be applied to obtain the actual clustering of the data.

The most common methods of clustering large data sets appear to be using one point to represent each original data item, as in BIRCH [30] and in the method in [7], or using sampling, as in CURE [17] and [8]. A given representative point is usually associated with many original data points, and once so associated all the original individual data points are no longer distinguishable from one another. In the method outlined in this work, we associate more than one representative with each data point, and the relationship between each data point and its representative(s) is uniquely determined for each data point. Hence each data point still has a unique representation in the resulting low memory factored representation. We also examine every data point when constructing our representation. We believe this method may have some advantages when compared with the clustering algorithms previously proposed for large data sets.

3 Clustering

Since clustering is performed in this work using the vector space model, the vector space model is described. Then, a more formal mathematical definition of clustering is given, followed by the description of a few measurements appropriate to clustering.

3.1 Vector Space Model

In the vector space model, an individual data item is represented by a column vector of its attributes. Each attribute must have a numerical value and is assigned a unique row in the vector. If we have a data set containing the attribute vectors $\mathbf{x}_i$, then we can define

$$\mathbf{M} \stackrel{\text{def}}{=} [\mathbf{x}_1 \, \mathbf{x}_2 \, \ldots \, \mathbf{x}_n], \tag{1}$$

where $\mathbf{M}$ is an $n \times m$ matrix, m is the number of attributes in each vector $\mathbf{x}_i$, and n is the number of items in the data set. Since the clustering and representation methods outlined in this paper use linear algebraic techniques, this representation is very convenient.

3.2 Clustering Model

Clustering is a re-arrangement of data such that the data is grouped using some measure of similarity. Let $\mathcal{M} = \{\mathbf{x}_1 \mathbf{x}_2 \ldots \mathbf{x}_n\}$, which is just the set of attribute vectors from the matrix $\mathbf{M}$ in (1). Then clustering can be interpreted as the result of the partition of the set $\mathcal{M}$ into subsets such that

$$\mathcal{M} = \bigcup_{i=1}^{K} \mathcal{M}_i, \quad \mathcal{M}_i \cap \mathcal{M}_j = \emptyset, i \neq j \tag{2}$$

where K is the total number of clusters. The partitioning takes place using only internal information and without any prior knowledge of the data. The goal of a clustering method is to maximize the similarity among the elements of each subset $\mathcal{M}_i$ and minimize the similarity of the elements in

$\mathcal{M}_i$ with respect to any other subset $\mathcal{M}_j$. The vectors in the set $\mathcal{M}_i$ can used to define the matrix $\mathbf{M}_i$ such that

$$\mathbf{M}_i \stackrel{\text{def}}{=} [\mathbf{x}_j],\ \mathbf{x}_j \in \mathcal{M}_i. \tag{3}$$

3.3 Measurements

One common way to delineate a cluster is to refer to its centroid. The centroid $\mathbf{w}_{\mathtt{C}}$ of a cluster $\mathbf{M}_{\mathtt{C}}$ is defined as:

$$\mathbf{w}_{\mathtt{C}} \stackrel{\text{def}}{=} \frac{1}{k_{\mathtt{C}}} \sum_{j \in \mathtt{C}} \mathbf{x}_j \tag{4}$$

where $k_{\mathtt{C}}$ is the number of items in cluster $\mathbf{M}_{\mathtt{C}}$ and $\mathbf{x}_j$ is the $j^{\mathtt{th}}$ column of $\mathbf{M}_{\mathtt{C}}$. The centroid represents the contents of a cluster in a compact way.

One commonly used intrinsic measure of the quality of a cluster is the *ScatterValue*. The *ScatterValue* is a measure of the cohesiveness of the data items in a cluster with respect to the centroid of the cluster The *ScatterValue* of a cluster $\mathbf{M}_{\mathtt{C}}$ is defined as:

$$\mathit{ScatterValue}_{\mathtt{C}} \stackrel{\text{def}}{=} \sum_{j \in \mathtt{C}} (\mathbf{x}_j - \mathbf{w}_{\mathtt{C}})^2 = \|\mathbf{M}_{\mathtt{C}} - \mathbf{w}_{\mathtt{C}}\mathbf{e}^T\|_F^2, \tag{5}$$

where $\mathbf{e}$ is the m-dimensional vector $[1\,1\,\ldots\,1]^T$ and $\|\ \|_F$ is the Frobenius norm. The Frobenius norm is the square-root of the sum of the squares of every entry in the matrix. When comparing two clusterings, the one with the smaller *ScatterValue* has the better clustering quality.

Another useful clustering performance measure is entropy. The entropy measures the coherence of a cluster with respect to how a cluster is labeled. An entropy calculation assumes that the labeling is perfect, which is not a good assumption in every case since any labeling performed by a human can be subjective. Since the data must be labeled, entropy cannot be used when the data have not been classified prior to clustering.

The total entropy of a given set of clusters is defined by:

$$e_{total} \stackrel{\text{def}}{=} \frac{1}{m} \sum_j e_j \cdot k_j,\ \text{where } e_j \stackrel{\text{def}}{=} -\sum_i \left(\frac{c(i,j)}{\sum_i c(i,j)} \right) \cdot \log \left(\frac{c(i,j)}{\sum_i c(i,j)} \right), \tag{6}$$

where $c(i,j)$ is the number of samples with label i in cluster j, and $k_j = \sum_i c(i,j)$ is the total number of samples in cluster j. If all of the labels of the items in a given cluster are the same, then the entropy of that cluster is zero. Otherwise, the entropy of that cluster is positive. The lower the entropy, the better the quality of the clustering.

4 The PDDP Algorithm

PDDP [5] is a clustering algorithm developed using techniques from numerical linear algebra. PDDP is a top-down method which recursively divides the data into smaller and smaller clusters, assembling all the clusters into a binary tree. Starting with the root node representing the entire dataset, PDDP computes the hyperplane which best divides the data. All the data on one side of the hyperplane is associated with one branch, and the data on the other side of the hyperplane is associated with the other branch. The process continues on each branch in turn until some stopping criteria is met, which can be based on intrinsic measures of the data in the nodes or on a final desired number of leaf nodes (clusters).

This method was originally developed as part of the WebACE Project [6] in the context of text documents where each document is represented by a scaled vector of word counts. However, this algorithm is not restricted to text domains and here it is described in general terms.

The clustering via PDDP is a recursive process that operates directly on the matrix $\mathbf{M}$. PDDP starts with a single "cluster" encompassing the entire data set and divides this cluster into subclusters recursively using a two step process. At each stage, PDDP (a) selects a cluster to split, and (b) splits that cluster into two subclusters which become children of the original cluster. The result is a binary tree hierarchy imposed on the data collection. At every stage, the leaf nodes in the tree form a partition of the entire data collection. In the process of going to the next stage, one of those leaf nodes is selected and split in two. The behavior of the algorithm is controlled by the methods used to accomplish steps (a) and (b), and these methods are independent of one another. For step (a), PDDP usually selects the cluster with the largest `ScatterValue`, though any suitable criterion can be used.

Once selected in step (a), the node is split in step (b), and this splitting process is the single most expensive step in the whole computation. The key to the computational efficiency of the entire approach is the efficient computation of the vectors needed in this step. Suppose PDDP were to split cluster $\mathtt{C}$ consisting of k data samples of attribute values. It places each data sample $\mathbf{x}$ in the left or right child of cluster $\mathtt{C}$ according to the sign of the linear discriminant function

$$g_{\mathtt{C}}(\mathbf{x}) = \mathbf{u}_{\mathtt{C}}^T(\mathbf{x} - \mathbf{w}_{\mathtt{C}}) = \sum_{i \in \mathtt{C}} u_i(x_i - w_i), \tag{7}$$

where $\mathbf{u}_{\mathtt{C}}$, $\mathbf{w}_{\mathtt{C}}$ are vectors associated with $\mathtt{C}$ to be determined. If $g_{\mathtt{C}}(\mathbf{x}) \leq 0$, the data sample $\mathbf{x}$ is placed in the new left child, otherwise $\mathbf{x}$ is placed in the new right child. Thus the behavior of the algorithm at each node in the binary tree is determined entirely by the two vectors $\mathbf{u}_{\mathtt{C}}$, $\mathbf{w}_{\mathtt{C}}$ associated with the cluster $\mathtt{C}$.

The vector $\mathbf{w}_{\mathtt{C}}$ is the *mean* or *centroid* vector, as defined in (4). The vector $\mathbf{u}_{\mathtt{C}}$ is the direction of maximal variance, also known as the leading left singular vector for the matrix $\mathbf{M}_{\mathtt{C}} - \mathbf{w}_{\mathtt{C}}\mathbf{e}^T$. This direction corresponds to the largest eigenvalue of the sample covariance matrix for the cluster. Here $\mathbf{M}_{\mathtt{C}}$ is the matrix of columns of data samples in cluster $\mathtt{C}$. The computation of $\mathbf{u}_{\mathtt{C}}$ is the most costly part of this step. It can be performed quickly using a Lanczos-based solver for the singular values of the data matrix (see [5]). This algorithm is very efficient, especially since low accuracy is all that is required, and can take full advantage of any sparsity present in the data.

The overall method is summarized in Fig. 1. As the method is "divisive" in nature, splitting each cluster into exactly two pieces at each step, the result is a binary tree whose leaf nodes are the sought-after clusters.

5 Matrix Representation Using Cluster Centers

An approximation or representation to a matrix $\mathbf{A}$ can be thought of as a system which captures the most "important" information in $\mathbf{A}$. Obtaining the representation usually involves a tradeoff among accuracy, the memory space occupied by the representation, and the time required to obtain the representation. These three aspects of the representation are usually in conflict, and must be tailored to fit the desired application. Unlike the recent low-rank representation methods in [2, 3, 20, 31], this representation is not used to extract directly the essential concepts that pervade an entire dataset, in which each new "concept vector" is forced to be independent (orthogonal in

Algorithm PDDP.
0. **Start** with $n \times m$ matrix $\mathbf{M}$ of vectors, one for each data sample, and a desired number of clusters k_f.
1. **Initialize** Binary Tree with a single Root Node.
2. **For** $c = 2, 3, \ldots, k_f$ **do**
3. **Select** leaf node C with largest *ScatterValue* (5), and L & R := left & right children of C [step (a) in the text].
4. **Compute** $\mathbf{v}_\mathtt{C} = g_\mathtt{C}(\mathbf{M}_\mathtt{C}) \equiv \mathbf{u}_\mathtt{C}^T(\mathbf{M}_\mathtt{C} - \mathbf{w}_\mathtt{C}\mathbf{e}^T)$
5. **For** $i \in$ C, if $v_i \leq 0$, then assign data sample i to L, else assign it to R [step (b) in the text].
6. **Result:** A binary tree with k_f leaf nodes forming a partitioning of the entire data set.

Figure 1: PDDP. $\mathbf{M}_\mathtt{C}$ is the matrix of data vectors for the data samples in cluster C, and $\mathbf{w}_\mathtt{C}, \mathbf{u}_\mathtt{C}$ are the centroid and principal direction vectors, respectively for C.

some cases) of the preceding "concept vectors." Rather, we are interested in using it only as a *low-memory* representation which can be used in place of the original data in the matrix vector products needed by the PDDP algorithm. Hence we can afford to use a faster method to generate the approximate representation.

One basis for approximation which has proven to be useful is the collection of centroids resulting from a clustering operation. The following description was taken from [10] and has been slightly modified. Say that you have a clustering of a data set with k clusters. The centroids of the clusters can be gathered into a $n \times k$ matrix $\mathbf{C}$ such that:

$$\mathbf{C} = [\mathbf{c}_1\, \mathbf{c}_2\, \ldots\, \mathbf{c}_k]. \tag{8}$$

Given the original matrix $\mathbf{M}$ representing the data, it is possible to construct an approximation to $\mathbf{M}$:

$$\mathbf{M} \approx \mathbf{CZ}, \tag{9}$$

where $\mathbf{Z}$ is a $k \times m$ matrix such that, for a given column $\mathbf{z}_i$ of $\mathbf{Z}$,

$$\mathbf{z}_i = \arg\min_{\mathbf{z}} \|\mathbf{x}_i - \mathbf{Cz}\|_2, \tag{10}$$

where $\mathbf{x}_i$ is the $i^{\mathtt{th}}$ column of $\mathbf{M}$. The direct way to solve for each $\mathbf{z}_i$ is to use a least-squares approximation. In the context of [10] $\mathbf{Z}$ was dense, and this was called a *concept decomposition*. The concept decomposition was designed to be a low-cost alternative to the singular value decomposition as used in e.g. citeBoleyPDDP98. One finds similar approaches proposed in the literature. One used centroids and least squares approximation formulated as a rank reduction problem [26], which was also shown to be useful in text classification. Another approach has been to use the left singular vectors in place of the centroids to represent the data space [4, 29].

The amount of memory occupied by the representation is variable. Assuming a reasonably small number of centroids in $\mathbf{C}$ and a dense $\mathbf{M}$, the bulk of the memory will be taken up by $\mathbf{Z}$. If $\mathbf{Z}$ is allowed to be dense, then the memory savings will be minimal. The solution is to enforce a sparsity condition on $\mathbf{Z}$, so that each column of the original matrix $\mathbf{M}$ is represented by a linear

combination of small fixed number k_z of columns of $\mathbf{C}$ instead of all k_c of them. We will propose one way to select which k_z columns to choose for each original column of $\mathbf{M}$.

This representation is designed to take up as little memory space as possible while reproducing the columns of $\mathbf{M}$ with enough accuracy to allow good clustering. It will probably not satisfy any requirements which might normally be imposed on a matrix approximation used in a traditional linear algebraic application, such as the linear independence of the basis vectors.

6 Clustering Large Data Sets with Approximate Representations

Recall that our problem is to cluster data sets that are too large to fit into memory. One solution to this problem is to divide the original data set into smaller pieces which will fit into memory, cluster them, and obtain an approximation to each piece of data. Once this is done, the approximations can be gathered into one system and then clustered. A more formal description of the method follows.

A matrix $\mathbf{M}$ can be divided into k_s disjoint *sections* such that:

$$\mathbf{M} = [\mathbf{M}_1 \, \mathbf{M}_2 \, \ldots \, \mathbf{M}_{k_s}], \tag{11}$$

where each section $\mathbf{M}_j$ is $n \times k_d$. The partitioning of $\mathbf{M}$ is assumed to be virtual or arbitrary (e.g. only the data for one section is in memory at a given time), and the ordering of the columns of $\mathbf{M}$ is assumed to be unimportant, as illustrated by the experiments in section 9.3 below. Once a section $\mathbf{M}_j$ is available, an approximation to $\mathbf{M}_j$ can be constructed:

$$\mathbf{M}_j \approx \mathbf{C}_j \mathbf{Z}_j, \tag{12}$$

where $\mathbf{C}_j$ is an $n \times k_c$ matrix of the form (8) and $\mathbf{Z}_j$ is has the form (9, 10) with k_c rows. Each column of $\mathbf{Z}_j$ has at most k_z nonzeroes. The centroids in each $\mathbf{C}_j$ are obtained through some kind of clustering algorithm.

Once an approximate representation is available for each section of data, they can be assembled into the approximate *low memory factored representation* of the entire data set $\mathbf{M}$:

$$\mathbf{M} \approx \mathbf{C}_M \mathbf{Z}_M, \tag{13}$$

where

$$\mathbf{C}_M = [\mathbf{C}_1 \, \mathbf{C}_2 \, \ldots \, \mathbf{C}_{k_s}] \quad (\text{an } n \times k_s k_c \text{ matrix}) \tag{14}$$

and

$$\mathbf{Z}_M = \begin{bmatrix} \mathbf{Z}_1 & & & \\ & \mathbf{Z}_2 & & \\ & & \ddots & \\ & & & \mathbf{Z}_{k_s} \end{bmatrix}, \tag{15}$$

(a $k_s k_c \times m$ matrix with k_z nonzeroes per column). Now that an approximate representation of $\mathbf{M}$ is available, and will fit into memory (because it was designed to fit into memory through judicious choice of the total number of centroids in each $\mathbf{C}_j$ and the number of nonzero elements in each $\mathbf{Z}_j$), it can be used to cluster the entire original data set. The algorithm for the method described in (11-15) is shown in Fig. 2.

Algorithm PMPDDP.

0. **Start** with a $n \times m$ matrix $\mathbf{M}$ of vectors, one vector for each data sample, and a desired number of final clusters k_f. Set the values for k_s (the desired number of sections), k_c (the number of clusters produced for each section), and k_z (the number of centroids used to approximate each data point).
1. **Partition M** into k_s disjoint sections, $|\mathbf{M}_1\, \mathbf{M}_2\, , \ldots ,\, \mathbf{M}_{k_s}|$.
2. **For** $j = 1, 2, \ldots, k_s$ **do**
3. **Compute** the PDDP tree for the section $\mathbf{M}_j$ with k_c clusters.
4. **Assemble** the k_c centroids from the leaf clusters into an $n \times k_c$ matrix $\mathbf{C}_j$.
5. **Compute** the $k_c \times m$ matrix $\mathbf{Z}_j$ minimizing the quantity $\|\mathbf{M}_j - \mathbf{C}_j\mathbf{Z}_j\|_F$ subject to the constraint on the number of nonzero elements k_z in each column of $\mathbf{Z}_j$.
6. **Assemble** the matrices $\mathbf{C}_M$ and $\mathbf{Z}_M$ as in (14, 15) in the text, using all the matrices $\mathbf{C}_j$ and $\mathbf{Z}_j$ from all passes through steps 2-5.
7. **Compute** the PDDP tree for the system $\mathbf{C}_M\mathbf{Z}_M$ with k_f clusters.
8. **Result:** A binary tree with k_f leaf nodes forming a partitioning of the entire data set.

Figure 2: PMPDDP algorithm.

PDDP is an ideal choice of algorithm to cluster the data using this representation. Central to PDDP is the calculation of the principal direction of the data to determine the splitting hyperplane. The principal direction is calculated using the iterative procedure developed by Lanczos, and this iterative procedure is based on the formation of the matrix-vector product $\mathbf{Mv}$. For any given split, the matrix $\mathbf{M}$ can be replaced by the quantity $\mathbf{C}_M\mathbf{Z}_M$, resulting in the formation of the product $\mathbf{C}_M(\mathbf{Z}_M\mathbf{v})$. Note that by calculating the matrix-matrix-vector product in this order, the product of $\mathbf{C}_M\mathbf{Z}_M$ never needs to be formed explicitly. The increase in the computational cost shouldn't be severe, and there won't be any increase in the memory requirements with respect to PDDP when performed on the original matrix.

Many other clustering algorithms would not be able to take full advantage of the memory savings given by $\mathbf{C}_M\mathbf{Z}_M$. Any method which requires a similarity measure between two data points in order to make a clustering decision would require either that $\mathbf{C}_M\mathbf{Z}_M$ be formed explicitly (most likely negating much of the memory savings), or that individual columns of $\mathbf{C}_M\mathbf{Z}_M$ be calculated every time they are needed, which would result in a large increase in computational cost. Hierarchical agglomeration and k-means are two algorithms which would not be suitable with this representation, and any derivative algorithms of the two would most likely not be appropriate either.

7 Complexity Analysis of PMPDDP

There are many costs associated with computing a PMPDDP clustering. There is the cost of clustering each section to obtain the centers used as basis vectors, the cost of getting the least-squares approximation, and the cost of clustering the approximation.

We will use the same notation as in the explanation of the piecemeal PDDP algorithm with the following assumptions: the data set is evenly distributed among the sections so that k_c and k_d are the same for each section, and k_z is the same for each section k_s, $k_d k_s = m$, and $m >> n$. Note that these are not necessary conditions to perform a PMPDDP clustering, they are merely used to

make the analysis clearer.

7.1 Obtaining the Basis

The basis consists of the cluster centers computed for each section of data. We assume that PDDP will be used to compute the clustering. We further assume that the majority of the cost of computation is the cost of computing the principal direction, and this cost in turn is dominated by the cost of computing vector-matrix products. This cost is $c_1 k_d n$, where c_1 is a constant factor representing the number of inner iterations (matrix-vector products) to convergence. If we further assume that we are computing a balanced tree, and that each PDDP split perfectly divides the data, then we can say that the splitting costs for one section of data are:

Number of clusters	Cost
2	$c_1 k_d n$
4	$c_1 k_d n + 2c_1 \left(\frac{k_d}{2}\right) n = 2c_1 k_d n$
8	$c_1 k_d n + 2c_1 \left(\frac{k_d}{2}\right) n + 4c_1 \left(\frac{k_d}{4}\right) n = 3c_1 k_d n$
16	$c_1 k_d n + 2c_1 \left(\frac{k_d}{2}\right) n + 4c_1 \left(\frac{k_d}{4}\right) n + 8c_1 \left(\frac{k_d}{8}\right) n = 4c_1 k_d n$
k_c	$c_1 k_d n \log_2(k_c)$

Therefore, the cost of computing the basis vectors for each section is $c_1 k_d n \log_2(k_c)$. The total cost of computing all basis vectors is:

$$c_1 k_s k_d n \log_2(k_c) = c_1 m n \log_2(k_c).$$

7.2 Computing the Approximate Representation

Computing the representation is a multi-step process. First, it is necessary to compute the distance from every data point k_d in the section to each center k_c in the section. It takes n multiplications to compute the distance, and we can ignore the computation of the square root since we are only interested in relative distances. Therefore it costs $k_d k_c n$ to compute all the distances for a given section, or $k_s k_d k_c n = m n k_c$ to compute the distances for the entire data set.

The next step is to find the k_z closest centers to each data item k_d in a section. The simplest way to do this for a small k_z is to search over all the distances to find the closest center, eliminate the center from consideration, and repeat the process until the k_z closest centers have been found. For one data point, this will result in $k_z k_c$ searches, which means the cost is $k_d k_z k_c$ searches for each section. The overall cost for all sections is $m k_z k_c$. There are other ways to find the k_z closest centers without scanning the entire data set multiple times, but implementing them in MATLAB would be slower since they would not be able to use as many intrinsic functions.

The final step is to compute the least-squares approximation for each data item using the k_z centers obtained in the previous step. The normal equations are the most inexpensive way to obtain the approximations. They are not as accurate as other methods, but a high degree of accuracy is not required in this application. The cost of computing a least-squares approximation for the $n \times k_z$ system using the normal equations is:

$$k_z^2 n + \frac{1}{3} k_z^3,$$

if we ignore the $O(k_z^2)$ term. The total cost of obtaining the least-squares approximation for every data item is:

$$m(k_z^2 n + \frac{1}{3}k_z^3).$$

The overall cost for computing the approximate representations for all sections of data is:

$$mnk_c + mk_z k_c + m(k_z^2 n + \frac{1}{3}k_z^3) = m(nk_c + k_z k_c + k_z^2 n + \frac{1}{3}k_z^3).$$

The memory occupied by the final approximate representation of the entire data set is

Factor	Dimensions	Number of memory cells needed		
$\mathbf{C}_M$	$n \times k_s k_c$	$k_s k_c n$	(typically dense)	(16)
$\mathbf{Z}_M$	$k_s k_c \times m$	$k_z m$	(only k_z nonzeroes per column)	
Total		$k_s k_c n + k_z m$		

7.3 Clustering with the Approximate Representation

Replacing the original matrix $\mathbf{M}$ with the representation $\mathbf{CZ}$ in the PDDP method changes the calculation in the splitting process from a vector-matrix product to a vector-matrix-matrix product. This product can be written as $(\mathbf{v}^\mathrm{T}\mathbf{C})\mathbf{Z}$, where $\mathbf{v}$ is a "generic" $n \times 1$ vector, $\mathbf{C}$ is a $n \times k_s k_c$ matrix and $\mathbf{Z}$ is a $k_s k_c \times m$ matrix. We will also assume that $\mathbf{Z}$ is a sparse matrix with k_z non-zeroes per column, and that the only computation cost with respect to $\mathbf{Z}$ is incurred when computing the product of the non-zero elements in $\mathbf{Z}$ with the elements in $\mathbf{v}^\mathrm{T}\mathbf{C}$. The cost of computing the principal direction is:

$$c_2(k_s k_c n + k_z m)$$

The above cost is for the first split. As the tree is built (again assuming a binary, perfectly balanced tree), the only thing that changes is the number of columns in $\mathbf{Z}$ which must be considered. The cost of computing $\mathbf{v}^\mathrm{T}\mathbf{C}$ is fixed, since $\mathbf{C}$ is fixed. We write the computations associated with the number of clusters computed:

Number of clusters	Cost
2	$c_2 k_s k_c n + c_2 k_z m$
4	$c_2 k_s k_c n + c_2 k_z m + 2c_2 k_s k_c n + 2c_2 k_z \left(\frac{m}{2}\right) = 3c_2 k_s k_c n + 2c_2 k_z m$
8	$c_2 k_s k_c n + c_2 k_z m + 2c_2 k_s k_c n + 2c_2 k_z \left(\frac{m}{2}\right) + 4c_2 k_s k_c n + 4c_2 k_z \left(\frac{m}{4}\right)$
	$= \; 7c_2 k_s k_c n + 3c_2 k_z m$
16	$c_2 k_s k_c n + k_z m + 2c_2 k_s k_c n + 2c_2 k_z \left(\frac{m}{2}\right) + 4c_2 k_s k_c n + 4c_2 k_z \left(\frac{m}{4}\right)$
	$+ \; 8c_2 k_s k_c n + 8c_2 k_z \left(\frac{m}{8}\right)$
	$= \; 15c_2 k_s k_c n + 4c_2 k_z m$
k_f	$c_2(k_f - 1)k_s k_c n + c_2 \log_2(k_f) k_z m$

7.4 Collected Costs

For clarity, we reproduce all of the costs of performing a piecemeal PDDP clustering in one place:

Operation	Cost
Obtaining Centers	$cmn \log_2(k_c)$
Computing the Approximation	$m(k_c k_z + k_c n + k_z^2 n + \frac{1}{3}k_z^3)$
Clustering Approximation	$c(k_f - 1)k_s k_c n + c \log_2(k_f) k_z m$

The cost for computing a standard PDDP clustering is $cmn \log_2(k_f)$. Depending on the dimensionality and the variable choices, clustering the approximate representation can be less expensive than clustering the original data set. However, the bulk of the expense of piecemeal PDDP is associated with obtaining the representation in the first place.

8 Data

There are five real data sets which will be used to measure the performance of PMPDDP. Three of them are small enough to fit into memory at once and be clustered comfortably on a computer with 512 MB of memory. One of them requires at least 1GB to be clustered at once, and the remaining data set is too large to fit into the memory of most workstations.

The astronomical data was derived from the Minnesota Automated Plate Scan (APS) [27]. This is a project to digitize the contents of photographic plates of the heavens from the Palomar Observatory Sky Survey (POSS I) originally produced in the 1950s, before the advent of artificial satellites. The sample used consists of 212089 galactic objects, each with 26 attribute values. An attribute might be coordinate information, color, brightness, etc. The attributes were generated by analyzing the blobs in the images, and the values were normalized so they lie on the interval $[0, 1]$.

The ISOLET (Isolated Letter Speech Recognition) data was generated by having 150 subjects speak the name of each letter of the alphabet twice. The attributes were extracted from recordings of the speakers, and include contour features, sonorant features, pre-sonorant features, and post-sonorant features. A total of 617 attributes were extracted from the pronunciation of each letter, and were scaled so they all lie on the interval $[-1.0, 1.0]$. There are a total of 7797 items available when the training and test sets are combined. The data first appeared in [14], and is in the UCI repository [24].

The k1 data set [6] consists of text documents selected from 20 news categories from the YAHOO web site. This data set has been included to demonstrate the effectiveness of the algorithms on document collections that might typically be retrieved from the World-wide Web. The data set consists of 2340 documents spanning 21839 words. The words were stemmed using Porter's suffix stripping algorithm [15], and the stop words were removed. The document vectors were scaled to until length, but no other scaling was performed.

The forest cover data set [18] consists of both continuous (e.g. elevation) and binary (e.g. soil type) attributes associated with the types of forest cover in a 30x30 meter square area. There are 54 attributes per data item, and a total of 581012 data items in the set. All of the data were labeled with respect to the kind of tree growing on the square. Each attribute scaled to have a mean of

zero and a variance of one, again with the scaling only being done with respect to the data present in memory. The scaling also had the side effect of turning a sparse data set into a dense one.

The kddcup99 data set [18] consists of a set of variables associated with network connections. The original task in the competition was to build an intrusion detector which could distinguish between "good" or normal connections and "bad" or intrusive connections. The data consists of both numerical(e.g. number of failed logins) and qualitative (e.g. connection protocol type, such as http or tcp) attributes. For the purpose of the experiments in this work, all of the qualitative attributes were converted into binary form. This resulted in each item in the data set having 122 attributes. There are a total of 4,898,431 items in the data set. All of them were labeled as being either a normal connection or one of a set of bad connections. Each attribute was scaled to have a mean of zero and a variance of one, with the scaling being done only with respect to the data in memory. Again, the scaling had the effect of turning a sparse data set into a dense data set.

dataset		astro	isolet	k1	forest cover	kddcup99
number of samples	m	212089	7997	2340	581012	4898431
number of attributes per sample	n	26	617	21839	54	122
sparsity (percentage nonzeroes)		dense	dense	0.68%	dense	dense
number of columns in assembled $\mathbf{C}_M$ (14)	$k_s k_c$	10000	750	250	11620	97968
memory footprint: $\mathbf{C}_M\mathbf{Z}_M$ vs $\mathbf{M}$:	(16)	24%	10%	56%	11%	6%
number of centers in final clustering	k_f	2000	150	50	2000	2000

Figure 3: Datasets and parameter values used for experiments. The approximate memory footprint is listed for $k_z = 3$. In the k1 dataset, the $\mathbf{C}$ factor was sparse yielding a smaller memory footprint than that predicted by (16), but the original data was also sparse limiting the total memory savings.

9 Experimental Results

The method was evaluated through a series of experiments. The first group of experiments was designed to demonstrate that PMPDDP will give a clustering which is as good as standard PDDP while allowing more data to be clustered due to the reduced memory requirements. This was done on the smaller data sets. The second group of experiments was designed to demonstrate that it is possible to predict the amount of time it will take to cluster a very large data set using PMPDDP by computing a clustering on a small subset of the data.

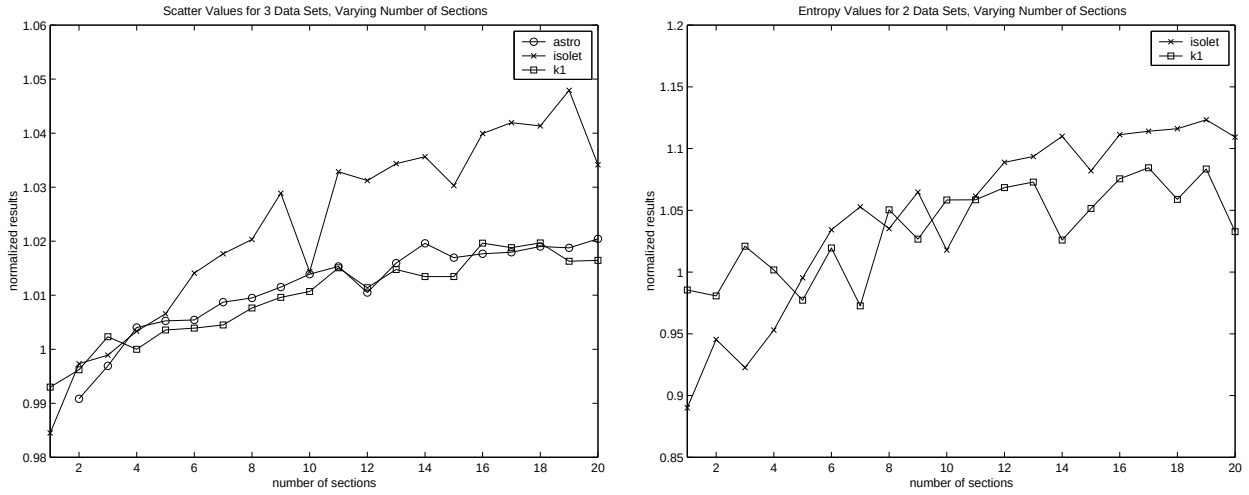
In almost every case, the least-squares problem (10) was solved using the normal equations applied to the k_z closest centers. On the rare occasion that the k_z closest centers were nearly linearly dependent, the singular value decomposition was used to construct the least-squares solution.

9.1 Fixed Memory

For these experiments, the amount of memory used by the approximate representation was fixed. The number of sections k_s (see Fig. 2) was varied over a range of 1 to 20. The product $k_s k_c$ was fixed, and the value of k_z was 5. Given the different sizes for the datasets, we used different values for the various parameters. These are summarized in Fig. 3. The results shown in the figures were normalized with respect to standard PDDP (Fig. 1) using the same value for k_f as was used in

PMPDDP. Recall that each data item is being approximated by k_z centers chosen out of a pool of k_c centers from its own section. As the number of sections grows, fewer centers are available to approximate each data item in step 5 of Fig. 2. As a result, the quality of the approximations would be expected to decrease as the number of sections increases. There are no results for the astronomical data for $k_s = 1$, $k_c = 10000$, since there was not enough memory available to produce 10000 clusters.

The results for the normalized scatter values are shown in Fig. 4(a). In some cases, applying PMPDDP to the entire data set together ($k_s = 1$) unexpectedly improved the quality of the clustering. This might be the result of the data being somewhat "smoothed" by the averaging operation which results from using the centroids to approximate the data. The results are worse as the number of sections increases, which is most likely the result of having fewer centers to choose from when selecting the five closest centers while obtaining the least-squares approximation (step 5 of Fig. 2).



(a) (b)

Figure 4: Results for the scatter (a) and entropy (b) values for a varying number of sections k_s, $k_z = 5$ using the parameters in Fig. 3 normalized with respect to standard PDDP (Fig. 1) using the same value for k_f. The astronomical data is unlabeled, so no entropy can be calculated.

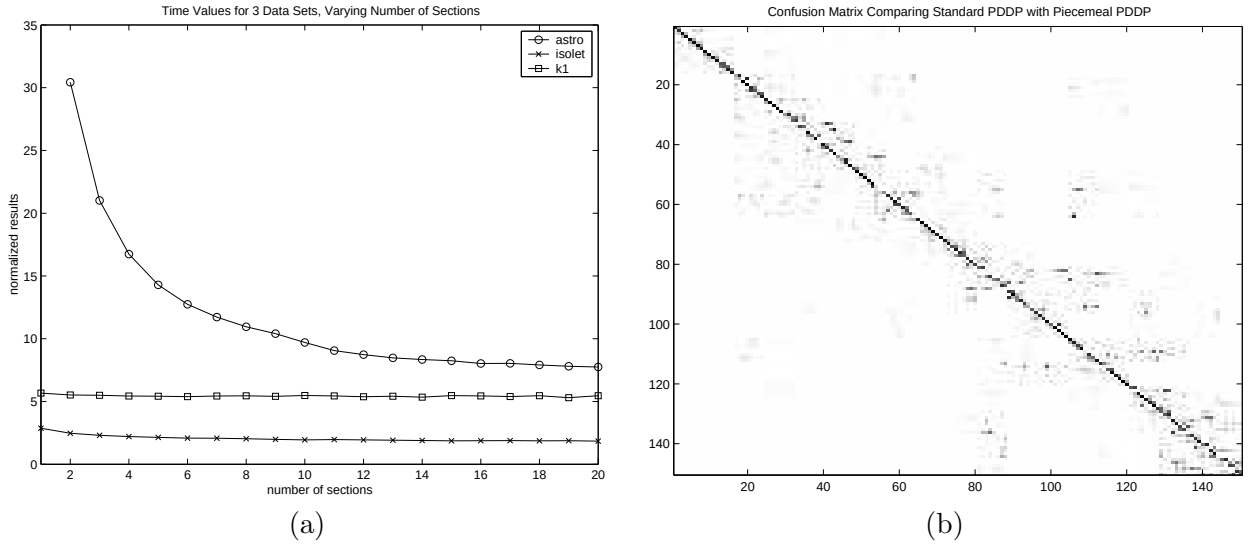
The results for the normalized entropy values are shown in Fig. 4(b). Note that since the astronomical data is not labeled, no entropy values can be calculated. The entropy values again indicate a reduction in quality as the number of sections increases, but for the Isolet data the entropy is better than standard PDDP until the number of sections is 5 or more. The results for the k1 document data are not as favorable, but are still comparable with the standard PDDP results.

The results for the normalized time values are shown in Fig. 5(a). The time cost is the one category in which PMPDDP suffers compared with PDDP. The expectation from [5] is that PDDP should be approximately linear in the number of samples being clustered, depending on the convergence of the singular value solver. Since the piecemeal algorithm must compute a large collection of intermediate centers, we would expect a corresponding increase in the time cost. The code for PDDP has been highly optimized, while the code for PMPDDP is not as mature. There is the

additional expense in obtaining more singular values than in standard PDDP, finding the k_z centroids which are closest to each data item, obtaining the least-squares approximations, and in the additional matrix multiplication required for every Lanczos iteration when the final clustering is obtained. These comments apply when the data set is small enough to be processed by the standard algorithm. The piecemeal method can be applied to data sets that are too large to be processed by the standard algorithm, and in these cases the piecemeal method would become competitive with any "out-of-core" variant of the standard method.

In the case of the astronomical data, PMPDDP varies from eight times to thirty times slower than PDDP. The relative time required for PMPDDP on the other two data sets is not as high, with the k1 data set taking a relatively steady six times longer, and the Isolet data taking from two and a half to at most four and a half times longer.

A confusion matrix for the Isolet data is shown in Fig. 5(b). PDDP was used as the ground truth, and the parameters for PMPDDP were $k_z = 5$, $k_s = 5$, $k_c = 150$. The number of final clusters k_f was 150 for both methods. The rows of the matrix were permuted so the most similar clusterings are along the diagonal. The figure shows that the clustering computed by PMPDDP is similar to the clustering computed by standard PDDP. This was true for the astronomical and k1 data as well.



(a) (b)

Figure 5: Part (a) shows the results for the time values for a varying number of sections k_s, $k_z = 5$, and using the parameters in Fig. 3, normalized with respect to standard PDDP (Fig. 1) using the same value for k_f. Part (b) is the confusion matrix for the isolet data with $k_z = 5$, $k_s = 5$, $k_c = 150$, comparing PMPDDP with standard PDDP with $k_f = 150$. The rows were permuted so the most similar clusters are along the diagonal.

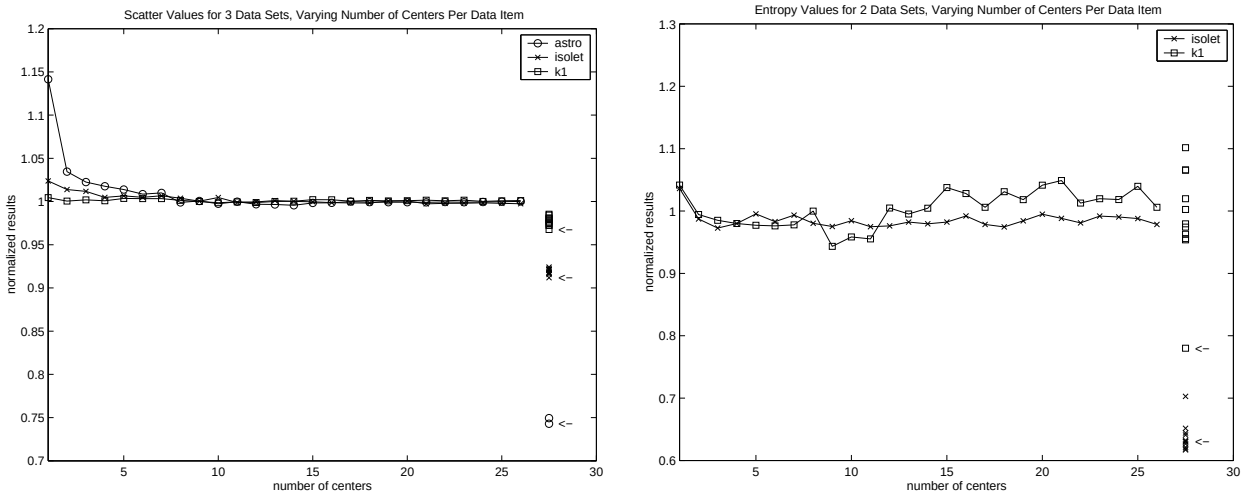
Since the product $k_s k_c$ was fixed, the size of both C_M and Z_M remained constant throughout these experiments. The memory used by the approximation for the astronomical data was about 32 percent of the memory used by the original data. The size of the approximation of the Isolet data was roughly 11 percent of the size of the original data, and the approximation to the k1 data was about 58 percent of the size of the original data. These results seem to indicate that dense data will have the most memory savings when using this technique, and that the higher the attribute

dimension of a dense data set, the more significant the memory savings.

9.2 Varying Number of Representatives

For the experiments in figs. 6–7(a), the number of centroids k_z used to approximate each data point was varied from 1 to 26, and the values of the other variables were fixed at $k_c = 1000$, $k_f = 2000$, $k_s = 10$ for the astronomical data, $k_c = 150$, $k_f = 150$, $k_s = 5$ for the Isolet data, and $k_c = 50$, $k_f = 50$, $k_s = 5$ for the k1 data. The values reported in the figures were normalized with respect to standard PDDP using the same value for k_f as was used in PMPDDP.

The results for the scatter values are shown in Fig. 6(a). Using more centers to approximate each data item improved the result for the astronomical and Isolet data sets, while the k1 data did not seem to be sensitive to the number of centers used. The most dramatic improvement in scatter value was for moving from using 1 center to using 2 centers to approximate each data item in the astronomical data set. The results for random-start k-means are vertically stacked at the right hand side of the graph, 10 runs each for the isolet and k1 data, and 1 run for the astronomical data. For these data sets, k-means was able to produce a better clustering than either PDDP or PMPDDP, but a k-means clustering takes substantially longer to compute. For each k-means iteration, it is necessary to find the distance from every point to every center, find the closest center, and compute a new center. This will cost $mnk_f + mk_f + mn$ per iteration, or $c_3(mnk_f + mk_f + mn)$. The arrows (<-) in the graph(s) refer to k-means applied to clusters computed by PDDP. PDDP seems to provide good starting centers for k-means.



(a) (b)

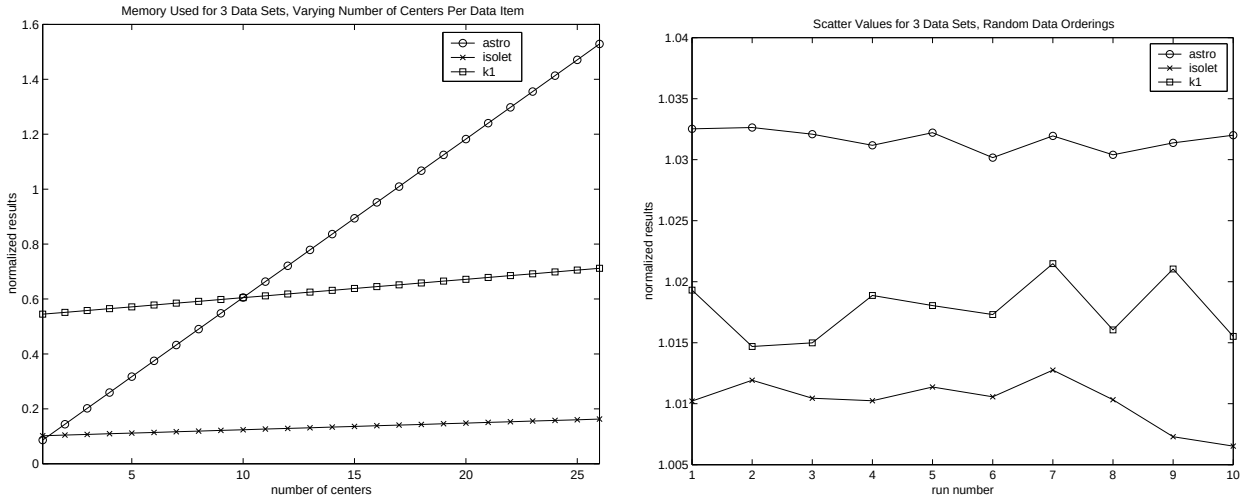
Figure 6: Results for the scatter (a) and entropy (b) values for k_z varying from 1 to 26, with (i) $k_s = 10$, $k_c = 1000$, $k_f = 2000$ (astronomical), (ii) $k_s = 5$, $k_c = 150$, $k_f = 150$ (Isolet), and (iii) $k_s = 5$, $k_c = 50$, $k_f = 50$ (k1), normalized with respect to PDDP using the same k_f. The astronomical data is unlabeled, so no entropy can be calculated. The vertically stacked points to the right of both graphs are the normalized results of random start k-means, 10 runs each for the isolet and k1 data, and 1 run for the astronomical data. The <- markers refer to k-means runs using the clusters from PDDP as a starting point. For comparison, the cost of k-means is $c_3(mnk_f + mk_f + mn)$.

The results for the entropy values are shown in Fig. 6(b). The entropy values vary quite a bit

over the range of the number of centers shown, but are always comparable to those from PDDP. In some cases, they are even better. The most dramatic improvement seems to occur when going from using 1 center to approximate each data item to using 2 centers. The entropies for the k-means clustering of the k1 data set are similar to those for the PMPDDP clustering, while the entropies for the isolet data set for the k-means clustering are substantially better than PMPDDP. Again, using PDDP to provide starting centers for k-means appears to be superior to either method on its own.

The time taken by PMPDDP was dependent on the number of centers k_z used to approximate each data item. As expected, increasing k_z increases cost of the PMPDDP clustering. K-means took roughly the same amount of time as standard PDDP for the k1 data set and 2-5 times longer for the isolet data set, depending on the rate of convergence. The astronomical data took many hours to cluster using k-means (exact times were unavailable due to load conditions), but only about 90 seconds to cluster using standard PDDP.

The memory used by the approximations $\mathbf{C}_M\mathbf{Z}_M$ normalized with respect to M is shown in Fig. 7(a). The values for the k1 data are the actual number of nonzeroes returned by MATLAB, while the values for the astronomical and Isolet data were calculated experimentally. In all cases, the amount of memory used by the approximation increases linearly with the number of centers used to approximate each data item. This is expected. The results for the astronomical data show that using more than 17 centers to approximate each data item is a break-even proposition as far as memory is concerned. This is not surprising since the astronomical data has only 26 attributes. Recall that the sparse array containing the matrix $\mathbf{Z}_M$ has more overhead per entry than a dense matrix. The Isolet data saves a significant amount of memory when it is approximated. The k1 document data saves some memory, but the effect is not as striking because the original dataset is already very sparse. However, it is interesting to note that even with the sparsity of the original data, it is still possible to obtain entropies as good as the original with less memory.



(a) (b)

Figure 7: Part (a) shows the results for the memory used for k_z varying from 1 to 26, with the same parameters as in Fig. 6, normalized with respect to standard PDDP using the same k_f. Part (b) shows the results for the scatter for 10 random orderings of the data sets with $k_z = 5$ and the remaining parameters the same as in Fig. 6, normalized with respect to standard PDDP using the same k_f.

9.3 Sensitivity to Ordering

To test the sensitivity of the method to the ordering of the data, the experiment with $k_s = 10$, $k_c = 1000$, $k_f = 2000$, $k_z = 5$ for the astronomical data, $k_s = 5$, $k_c = 150$, $k_f = 150$, $k_z = 5$ for the Isolet data, and $k_s = 5$, $k_c = 50$, $k_f = 50$, $k_z = 5$ for the k1 data, was repeated 10 times with a different random ordering of the data set. The results for the scatter values, normalized with respect to standard PDDP using the same k_f, are shown in Figure 7(b). The graph seems to indicate that while there is some sensitivity to data ordering, it is not severe.

9.4 Predicting Clustering Expense

The previous results have indicated that it is more expensive to obtain a clustering using PMPDDP than it is to cluster the data at once, as expected. However, PMPDDP was designed to be used when the data set is too large to fit into memory and be clustered. We will now examine the performance of the method on the two largest data sets, the forest cover type and the kddcup99 data.

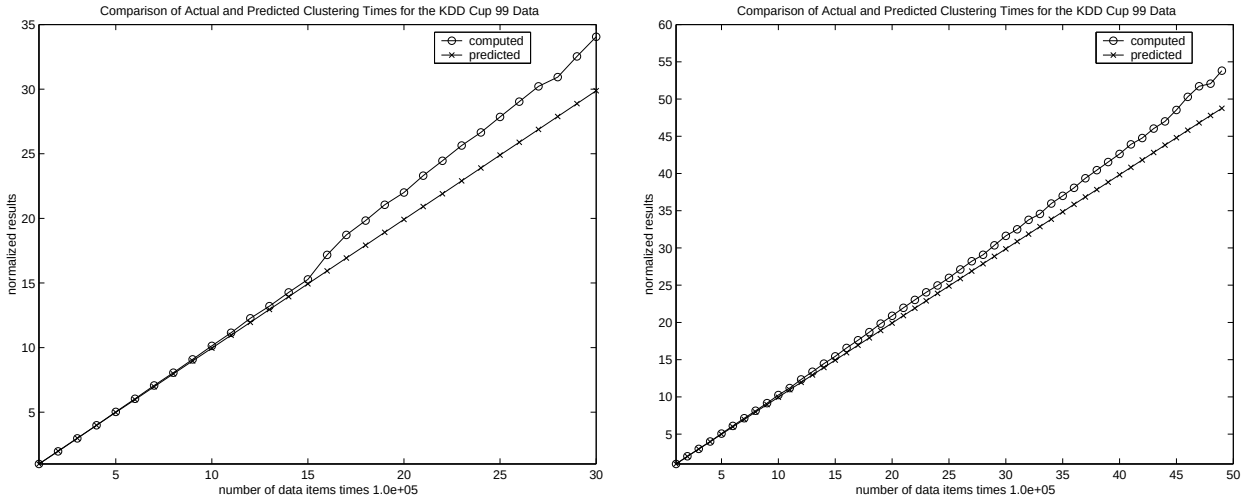
Since this data will not fit into memory at once on most workstations, the order of the samples in each data set was first randomized and then divided into smaller files. The smaller files consisted of 100,000 data items apiece, with the last data file in a given data set containing the remainder of the data items. The forest cover data was therefore divided into five files containing 100,000 items and one additional file containing 81012 items, and the kddcup99 data was divided into 48 data sets containing 100,000 items and one additional file containing 98,431 items. These numbers were chosen so that it would be possible to cluster each data file separately while still allowing enough memory for the overhead associated with PMPDDP on a computer with 512MB of memory. The total number of centers used to approximate each section of data was adjusted for the last file in both cases so that each section had a proportionate number of centers. For example, each of the first 48 sections of the kddcup data had 2000 centers as a basis for approximation, so the last section had 1968 centers as a basis.

Unlike the previous experiments in which all computations were done at once in one pass, in this case the approximation to each section of data was computed separately and saved in a temporary file. Then, when the clustering of the data was computed, the approximation to each section was read in as necessary. The timing results reported are the sum of the costs incurred in computing the approximation and the final clustering, but do not include any I/O costs. However, reading the approximation in from a file and assembling the approximation to the data only takes a few seconds at most, so the bulk of the unreported I/O costs are associated with the initial reading of the data. These I/O costs are assumed to be independent of the clustering method.

The clusterings of the approximations were computed in the following manner. The approximation associated with the first section of data was read in and then clustered. Then the approximation to the first two sections were read in and then clustered. This process continued until the last step, which was the computation of the clustering of the entire data set using all the approximations. The purpose of this approach was to demonstrate how the times increased as the size of the dataset increased.

The costs for the kddcup data set are shown in Fig. 8. The results in part (a) are for a clustering with $k_f = 2000$ final clusters, and the results in part (b) are for a clustering with $k_f = 500$. In both cases, the approximations to each section of data were constructed using $k_c = 2000$ (except for the last section, which had $k_c = 1968$) and $k_z = 3$. The results are normalized with respect to

the cost of computing the clustering of the first section of data. The theoretical costs of computing the approximation and the clustering are also shown in each graph, again normalized with respect to the theoretical cost of clustering the first section. As can be seen, the theoretical predication is a linear increase in cost with a linear increase in the amount of data clustered, and that is supported by the actual results. The numbers stop at 30 sections in part (a) due to a lack of memory. The computer(s) (identical dual-processor PIIIs with 2 GB total RAM) used in the experiments were shared, and the loads were sometimes unpredictable. However, each experiment was run on a dedicated processor. We believe that the deviation from the predicted costs as the number of sections rises are mostly due to cpu sharing and memory sharing/fragmentation issues. In any case, it appears that even under less than ideal conditions, PMPDDP scales well for this data set.



(a) (b)

Figure 8: Costs for the kddcup99 data set. Part (a) shows the total clustering cost for $k_f = 2000$. Part (b) shows the total clustering cost for $k_f = 500$. In both cases, the costs are normalized with respect to the cost of clustering the first section of data. Each graph also contains the theoretical costs of clustering the data, again normalized with respect to the cost of clustering the first section.

It is not possible to compare the entropy of a PMPDDP clustering of the kddcup99 data with a PDDP clustering of the data, since a PDDP clustering of the data cannot be computed. However, a PDDP clustering of each section of data was computed individually and the results were compared with a clustering of the approximation to each section with the same number of final clusters ($k_f = 500$). The entropies in both cases were almost identical. In some cases, clustering the approximation to the section of data produced a slightly better entropy, and in other cases a PDDP clustering of the section of data had better entropy. These results would seem to indicate that the approximations to each section are capturing the aspects of the data essential in obtaining a good clustering.

The clustering costs for the forest cover data were similar to the results for the kddcup99 data set. The cost of PMPDDP increased linearly with the number of data samples clustered. As in the previous case, each section of data was clustered using PDDP and the entropies were compared with a clustering of approximation to each section. The entropies compared favorably, with no clear advantage to either method. Also, since the data set is large but not huge, it was possible

to cluster the entire set at once when a large amount (over 1 GB) of memory was available. In this case, the entropies for PDDP and PMPDDP were virtually identical when using $k_c = 2000$ for the first five sections and $k_c = 1620$ for the last section. They only differed after the 4th decimal place. This would seem to indicate that PMPDDP is able to cluster this data set well, and that the approximations are of sufficient quality for clustering.

10 Conclusions

The experiments demonstrate that PMPDDP works for the data sets examined. The performance indicated by the scatter and entropy values does not suffer significantly with respect to PDDP when the approximations to the data are used to cluster the original data. The amount of memory taken by the approximation can be varied to suit the application. In general, it appears that as much memory as possible should be used to contain the approximation, since the accuracy of the clustering generally increases with the number of centroids used to approximate each data item.

PMPDDP appears to be able to compute clusterings of a quality almost as good as that from PDDP, and in some cases better, while using significantly less memory. PMPDDP does suffer from an increased time cost due to the many intermediate clusters computed and all of the additional costs associated with computing the approximations. These costs can, however, be predicted by computing a PMPDDP clustering of a portion of the data set and using the analysis of the algorithm to scale the results to the entire data set.

Most of the methods for large data sets involve some kind of sampling of the data. PMPDDP examines every data point and creates an approximation to every data point. Other clustering methods such as Birch [30] and the method in [7] examine every data point as well, but still use only one vector to represent each data point. PMPDDP is more flexible in that it can use as many centroids to represent each data point as memory allows, which seems to increase the accuracy of the clustering. The result is a method which will cluster large data sets to good accuracy in a reasonable amount of time.

Acknowledgements

This research was supported by U.S. NSF grants IIS-0208621 and IIS-0534286.

Dedication

The authors dedicate this article to Marco Somalvico, who was always able to inspire us with his unbounded spirit and enthusiasm. The second author was particularly inspired by long and deep conversations on Artificial Intelligence and academic research, beginning in 1983 during an extended visit to the Politecnico di Milano, and continuing during many later occasions when the second author was invited to the Politecnico to give seminars or short courses.

References

[1] D. Achlioptas and F. McSherry. Fast computation of low rank matrix approximations. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 611–618, 2001.

[2] Rie Kubota Ando. Latent semantic space: Iterative scaling improves inter-document similarity measurement. In *Proceedings of 23rd SIGIR*, pages 216–223, 2000.

[3] M. W. Berry, S. T. Dumais, and G. W. O.'Brien. Using linear algebra for intelligent information retieval. *SIAM Rev.*, 37:573–595, 1995.

[4] D. Boley. A scalable hierarchical algorithm for unsupervised clustering. In R. Grossman, C. Kamath, P. Kegelmeyer, V. Kumar, and R. Namburu, editors, *Data Mining for Scientific and Engineering Applications*, pages 383–400, 2001. Kluwer.

[5] D.L. Boley. Principal Direction Divisive Partitioning. *Data Mining and Knowledge Discovery*, 2:325–344, 1998.

[6] D.L. Boley, M. Gini, R. Gross, E-H Han, K. Hastings, G. Karypis, V. Kumar, B. Mobasher, and J. Moore. Document categorization and query generation on the World Wide Web using WebACE. *AI Review*, 13(5-6):365–391, 1999.

[7] P. S. Bradley, Usama M. Fayyad, and Cory Reina. Scaling clustering algorithms to large databases. In *Knowledge Discovery and Data Mining*, pages 9–15, 1998.

[8] Paul S. Bradley and Usama M. Fayyad. Refining initial points for K-Means clustering. In *Proc. 15th International Conf. on Machine Learning*, pages 91–99. Morgan Kaufmann, San Francisco, CA, 1998.

[9] D.R. Cutting, D.R. Karger, J.O. Pedersen, and J.W. Tukey. Scatter/gather: a cluster-based approach to browsing large document collections. In *15th Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'92)*, pages 318–329, 1992.

[10] I. Dhillon and D.S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1):143–175, 2001.

[11] P. Drineas, R. Kannan, A. Frieze, S. Vempala, and V. Vinay. Clustering of large graphs via the singular value decomposition. *Machine Learning*, 56:9–33, 2004.

[12] P. Drineas, R. Kannan, and M. Mahoney. Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix. Technical Report, TR-1270, Computer Science Dept.,Yale Univ., February 2004.

[13] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, 1973.

[14] M. Fanty and R. Cole. Spoken letter recognition. In *Advances in Neural Information Processing Systems 3*, pages 220–226, 1991.

[15] W. B. Frakes. Stemming algorithms. In W. B. Frakes and R. Baeza-Yates, editors, *Information Retrieval Data Structures and Algorithms*, pages 131–160. Prentice Hall, 1992.

[16] Earl Gose, Richard Johnsonbaugh, and Steve Jost. *Pattern Recognition and Image Analysis*. Prentice Hall, 1996.

[17] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: an efficient clustering algorithm for large databases. In *Proc. of 1998 ACM-SIGMOD Int. Conf. on Management of Data*, pages 73–84, 1998.

[18] S. Hettich and S. D. Bay. The UCI KDD archive, 1999. `kdd.ics.uci.edu/`.

[19] R. Kannan, S. Vempala, and A. Vetta. On clusterings: Good, bad and spectral. *Journal of the ACM*, 51(3):497–515, May 2004.

[20] Tamara G. Kolda and Dianne P. O.'Leary. A semidiscrete matrix decomposition for latent semantic indexing in information retrieval. *ACM Trans. Information Systems*, 16:322–346, 1998.

[21] D. Littau and D. Boley. Using low-memory representations to cluster very large data sets. In *SDM'03 Third SIAM Conference on Data Mining*, pages 341–345, May 2003.

[22] D. Littau and D. Boley. Clustering very large data sets with principal direction divisive partitioning. In J. Kogan, C. Nicholas, and M. Teboulle, editors, *Grouping Multidimensional Data: Recent Advances in Clustering*, pages 107–134, 2005. Springer.

[23] David Littau and Daniel Boley. Streaming data reduction using low-memory factored representations. *Journal of Information Sciences*, 176(14):2016–2041, 2006.

[24] P. M. Murphy and D. W. Aha. UCI repository of machine learning databases, 1994. `www.ics.uci.edu/~mlearn/MLRepository.html`.

[25] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In Jorgeesh Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *20th International Conference on Very Large Data Bases, September 12–15, 1994, Santiago, Chile proceedings*, pages 144–155, Los Altos, CA 94022, USA, 1994. Morgan Kaufmann Publishers.

[26] H. Park, M. Jeon, and J.B. Rosen. Lower dimensional representationof text data based on centroids and least squares. *BIT*, 43(2):1–22, 2003.

[27] R.L. Pennington, R.M. Humphreys, S.C. Odewahn, W. Zumach, and P.M. Thurmes. The automated plate scanner catalog of the palomar sky survey – scanning parameters and procedures. *P. A. S. P.*, 105:521ff, 1993.

[28] D. Zeimpekis and E. Gallopoulos. PDDP(l): Towards a flexing principal direction divisive partitioning clustering algorithms. In D. Boley, I. Dhillon, J. Ghosh, and J. Kogan, editors, *Proc. IEEE ICDM '03 Workshop on Clustering Large Data Sets*, pages 26–35, 2003. Melbourne, Florida.

[29] D. Zeimpekis and E. Gallopoulos. Clsi: A flexible approximationscheme from clustered term-document matrices. In H. Kargupta et al., editor, *Proc. Fifth SIAM Int'l Conf. Data Mining*, pages 631–635, April 2005.

[30] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2):141–182, 1997.

[31] Z. Zhang, H. Zha, and H. Simon. Low-rank approximations with sparse factors I: Basic algorithms and error analysis. *SIAM J. Matrix Anal.*, 2001. to appear.