Using Low-Memory Representations to Cluster Very Large Data Sets

David Littau
littau@cs.umn.edu

Daniel Boley
boley@cs.umn.edu

Abstract

Many of the algorithms designed to cluster large data sets compute representations of the data which are based on a single vector, without a unique representation of the original data items. We present an extension of Principal Direction Divisive Partitioning which creates a least-squares approximation of the data based on a small number of vectors. We show that the extension can save significant amounts of memory and cluster the data as well as the original method. We also show that in some cases using more than one vector to approximate each data item results in superior quality clusterings.

keywords: clustering, large data sets, PDDP, data mining, principal directions, matrix approximation

1 Introduction

Most clustering algorithms require that the data fit into memory at once in order to run efficiently. Clustering algorithms that require a pairwise similarly measure are restricted to clustering even fewer points, since computing and storing the pairwise similarity costs $O(m^2)$, where m is the number of points in the data set. Either available memory or computational expense restrict standard clustering algorithms to a fixed data set size.

The most direct solution to these limitations is to sub-sample the data and use the sub-sample to find the desired number of cluster centers. After the centers have been found, all the data are associated with the closest center. Sampling has the advantage of being simple and straightforward. However, it is often difficult to choose a good representative sample of the data, and sampling can miss anomalies which might be interesting.

Another solution is to compute a representation of the original data which takes up less space. The works in [6, 4, 15] are various techniques of computing a representation of the entire original data based on a much smaller set of approximation vectors, such that each data item is associated with one vector. Using one vector to represent many original data points saves in both memory and the number of pairwise similarity measures, if required. The drawback to using one vector to represent a data point is that each data point does not have a unique representation. If a cluster contains items which are all associated with the same single vector, no further differentiation among the data items is possible. This may effect clustering quality in some cases. Also, these methods do not address the issue of outliers, which might be of interest in some cases.

We propose an alternate method of approximating the original data so it can fit into memory and be clustered. Each data item is approximated by a linear combination of a small number of representative vectors chosen from a pool of vectors. The total number of vectors in the pool remains small to save memory, and the representative vectors are inexpensive to obtain. The resulting low-memory representation of the original data is stored as two matrices. One matrix contains the representative vectors, and the other is a sparse matrix containing the coefficients. The product of these two matrices is the approximation of the original data.

Since the assumption is that the data will not fit into memory at once, the low-memory representation is constructed in a piecemeal fashion. The original data are divided into disjoint sets such that each set will fit into memory, and a low-memory representation is computed for each set of data. The vectors used to represent the data are the centroids obtained from a clustering of the data, as was done in [7]. The centroids closest to the original data item are used to represent that data item in a least-squares approximation. Once the low-memory representations are computed for each of the smaller sets of data, they are assembled into a single two-matrix system which represents the entire data set.

This low-memory representation was designed to be clustered using Principal Direction Divisive Partitioning (PDDP) [2]. PDDP does not require a pairwise similarity measure to determine the clustering. The key component of PDDP is determining the principal direction of the data, which is computed using a Lanczos-based solver. The solver computes matrix-vector products such that the low-memory representation can remain in factored form.

Since PDDP is fast and scalable, we also use

PDDP to perform the intermediate clusterings used to obtain the low-memory representation. We call the method of computing the low-memory representation of the data and clustering that representation Piecemeal PDDP (PMPDDP). PMPDDP preserves the scalability of PDDP while extending PDDP to large data sets.

2 Previous Work

In the context of data mining, the construction and application of low cost and/or low memory representations of data is usually restricted to either clustering or Latent Semantic Indexing (LSI) [1]. Since the low-memory representation used in PMPDDP was designed for clustering, but has its origin in an application for LSI, we present a sample of the work that has been done in both areas.

2.1 Approximations in LSI The goal of LSI is to construct a representation of the data such that hidden relationships among the data can be discovered. The standard representation of the data for LSI is obtained using the singular value decomposition(SVD). The SVD of a data matrix $\mathbf{A}$,

$$\mathbf{A} \approx \mathbf{U\Sigma V}^T$$

has the smallest reconstruction error with respect to the original data, but it is very expensive to obtain and often takes up a significant amount of memory. Alternatives have been developed to address both of these issues.

Two techniques which save memory are the Semi-Discrete Decomposition (SDD) [11] and the method in [16]. The SDD represents the original data using two matrices $\mathbf{X}$ and $\mathbf{Y}$ which have entries taken from the set $-1, 0, 1$, and one diagonal matrix $\mathbf{D}$ constructed using real numbers. If the original data matrix is $\mathbf{A}$, then the k^{th} rank approximation to $\mathbf{A}$ is:

$$\mathbf{A}_k = \mathbf{X}_k \mathbf{D}_k \mathbf{Y}_k^T.$$

Starting with $\mathbf{A}_1$, each successive column of $\mathbf{X}_k$, $\mathbf{Y}_k$, and $\mathbf{D}_k$ is computed to reduce the residual error between the approximation and the original data, subject to the constraints mentioned. For a given level of accuracy, the SDD saves memory compared to the SVD, but is more expensive to compute than the SVD.

The method in [16] is a direct application of the SVD which saves memory with respect to the standard SVD. The method starts by computing a rank 1 SVD of the data, and then restricts the number of non-zero entries in the first column of $\mathbf{U}$ and $\mathbf{V}$. Each subsequent step involves computing a rank 1 SVD of the current residual and restricting the number of non-zero entries in the corresponding columns of $\mathbf{U}$ and $\mathbf{V}$. Again, memory is saved with respect to the SVD, but it is more expensive to compute than the SVD.

One inexpensive, though not necessarily memory saving, alternative to the SVD for LSI is the *concept decomposition* [7]. The concept decomposition demonstrated that clustering can be used to compute an inexpensive matrix approximation. In the concept decomposition, document data were clustered using spherical k-means, and the cluster centers were used as a basis for constructing a least-squares approximation to the original data. Given sufficient rank relative to the SVD, the concept decomposition had the same accuracy as the SVD on the document data set, while being less expensive to compute than the SVD.

2.2 Approximations Using One Representative Per Sample There are other approximation techniques which were specifically designed to be used for clustering. For example, BIRCH [15], Scatter/Gather [6], and the method in [4] approximate each original data item by a vector which is like a cluster centroid. More than one data item is associated with each vector. What follows is a brief overview of these techniques. Details have been omitted for the sake of brevity.

BIRCH obtains the vectors via a process which scans the entire data set once and associates data with vectors based on proximity. When a new vector is introduced, one of three things can happen. If the new vector is close enough to an existing representative vector, it is folded in with the representative vector, which is then altered to reflect the new data item. If there are no vectors close enough, and there is memory left, then the new data item becomes a new representative vector. If there isn't enough memory left for a new representative vector, then the tolerance is changed and the set of representative vectors is re-computed to reflect the new tolerance. We omit a detailed description of how this is done, but it is possible to do this efficiently and without scanning the data again. With the data structure used, it is not necessary to compare each new data vector with every representative upon insertion.

Scatter/Gather separates the original data into buckets, and then agglomerates the data in each bucket individually, such that a given number of clusters are produced in a given bucket. Each data item in a given cluster is associated with its cluster centroid. The centroids are weighted with respect to how many data items they represent, and this weighting is used when combining them to create new cluster centroids. The centroids are gathered together, put into a smaller set of buckets, and agglomerated again. The process continues until the desired number of clusters has been computed.

The method in [4] is a variant of k-means for large data sets. A subset of the data are loaded into memory and clustered with k-means until a certain number of clusters are obtained. Then, the centroids of each cluster, weighted by the cardinality of the cluster, become the representatives of the data. The data are removed and a new set of data is loaded and clustered along with the centroids from the previous iteration. This process continues until either all the data have been scanned or until the user is satisfied that the cluster centers aren't changing significantly with the addition of new data.

2.3 Summary Most approximation methods used in clustering associate each data item with one approximation vector, with no attempt to provide a unique representation for the data items. This is practical from a pure memory-saving approach, but some clustering accuracy may be lost if the data do not form tight groups at the time the approximation is constructed.

The concept decomposition creates a computationally inexpensive approximation using cluster centroids. Each data item is approximated using the entire set of centroids. This provides a unique representation for each original data item, but will probably not save memory.

We base our low-memory representation of the data on the concept decomposition. We use cluster centroids to approximate the data, but we choose a small number of centroids from the larger collection of centroids to approximate each data item. This results in a unique representation for each data item that can still save a significant amount of memory.

3 Definitions

To insure that the explanation of the methods is clear, we provide a definition of some of the terms we will use throughout this work.

We are using the vector space model. Each data item is represented as a column vector, and the entire data sample is represented by a matrix. If we have a data set containing the attribute vectors $\mathbf{x}_i$, we can define

$$\mathbf{M} \overset{\text{def}}{=} [\mathbf{x}_1\, \mathbf{x}_2\, \ldots\, \mathbf{x}_m], \tag{3.1}$$

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

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

$$\mathbf{w}_\mathsf{C} \overset{\text{def}}{=} \frac{1}{k_\mathsf{C}} \sum_{j \in \mathsf{C}} \mathbf{x}_j \tag{3.2}$$

where k_C is the number of items in cluster $\mathbf{M}_\mathsf{C}$ and $\mathbf{x}_j$ is the j^{th} column of $\mathbf{M}_\mathsf{C}$. 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 scatter value. The scatter value is a measure of the cohesiveness of the data items in a cluster with respect to the centroid of the cluster. The scatter value SV_C of a cluster $\mathbf{M}_\mathsf{C}$ is defined as:

$$SV_\mathsf{C} \overset{\text{def}}{=} \sum_{j \in \mathsf{C}} (\mathbf{x}_j - \mathbf{w}_\mathsf{C})^2 = \|\mathbf{M}_\mathsf{C} - \mathbf{w}_\mathsf{C}\mathbf{e}^T\|_F^2, \tag{3.3}$$

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 scatter value is assumed to have the better clustering quality.

Entropy is another clustering quality measure. The entropy measures the coherence of a cluster with respect to how a cluster is labeled. The total entropy of a given set of clusters is defined by:

$$e_{total} \overset{\text{def}}{=} \frac{1}{m} \sum_j e_j \cdot k_j, \tag{3.4}$$

where

$$e_j \overset{\text{def}}{=} -\sum_i \left(\frac{c(i,j)}{\sum_i c(i,j)} \right) \cdot \log \left(\frac{c(i,j)}{\sum_i c(i,j)} \right), \tag{3.5}$$

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 total entropy e_{total}, the better the quality of the clustering.

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. Entropy cannot be used when the data have not been classified prior to clustering.

4 PDDP

The algorithm developed in this work, PMPDDP, is an extension of PDDP for large data sets. The low-memory representation of the data is constructed so that it will have a minimal impact on the execution expense of PDDP. Therefore, we now provide a detailed description of PDDP so it will be clear how it has been extended.

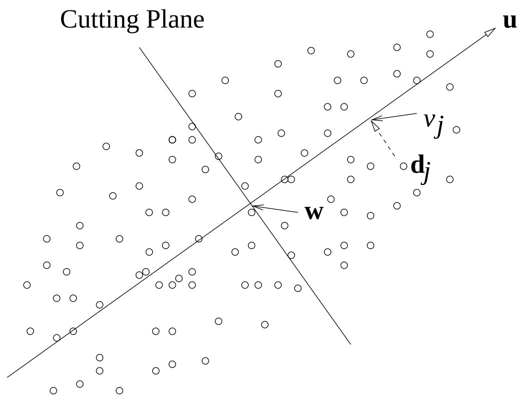

Figure 1: A PDDP split in two-dimensional space. The principal direction $\mathbf{u}$ of the data is computed, and the data points $\mathbf{d}_i$ in the cluster are projected onto $\mathbf{u}$, defining the points v_i. All points associated with v_i greater than the mean $\mathbf{w}$ are placed in one new cluster, and the rest of the points are placed in the other new cluster.

```
Algorithm PDDP. Start with a $n \times m$ matrix $\mathbf{M}$,
        where each column of $\mathbf{M}$ is a data item,
        and set the 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 scatter value,
            and L & R := left & right children of C.
4.      Compute $\mathbf{v}_{\mathtt{C}} \equiv \mathbf{u}_{\mathtt{C}}^T(\mathbf{M}_{\mathtt{C}} - \mathbf{w}_{\mathtt{C}}\mathbf{e}^T)$
5.      For $i \in$ C, if $v_i \le 0$, then assign data sample $i$
                                        to L,
                                  else  assign it to R.
6.  Result: A binary tree with $k_f$ leaf nodes forming
        a partitioning of the entire data set.
```

Figure 2: 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.

4.1 PDDP Algorithm PDDP is an unsupervised divisive clustering algorithm. The values of the data attributes are used to determine the clustering, and there is no prior knowledge of class membership of the data items. PDDP constructs a hierarchical binary tree by recursively splitting clusters. The clusters are split using the direction of maximal variance of the data attributes, also known as the principal direction of the data.

The PDDP algorithm begins with the root cluster, which contains all the data samples. The root cluster is split by finding the principal direction $\mathbf{u}$ of the cluster and projecting all the data in the cluster onto $\mathbf{u}$. All data on one side of the mean $\mathbf{w}$ of the cluster are placed in one child cluster, and the rest are placed in the other child cluster. An example of a two-dimensional split is shown in Figure 1.

After the root cluster is split, there are two leaf clusters. One of the leaf clusters is selected and split. Usually, the leaf cluster with the largest scatter value (3.3) is chosen, but it is possible to use other selection criteria, such as choosing the leaf cluster with the largest cardinality. The process of selecting and splitting leaf clusters continues until some stopping criterion is met. The leaf clusters of the PDDP tree define the clustering of the data set. The PDDP algorithm is shown in Figure 2.

The principal direction of the cluster is determined by computing the left leading singular vector of the data in the cluster. If $\mathbf{M}_{\mathtt{C}}$ is the matrix of columns of data samples in cluster C, we compute the left leading singular vector $\mathbf{u}_{\mathtt{C}}$ of $\mathbf{M}_{\mathtt{C}} - \mathbf{w}_{\mathtt{C}}\mathbf{e}^T$. This direction corresponds to the largest eigenvalue of the sample covariance matrix of the cluster. The computation of $\mathbf{u}_{\mathtt{C}}$ can be accomplished quickly using an iterative Lanczos-based solver for the singular values of $\mathbf{M}_{\mathtt{C}}$. This algorithm is very efficient, especially since low accuracy is all that is required, and it can take full advantage of any sparsity present in the data.

4.2 PDDP Complexity Analysis We present a complexity analysis of PDDP for completeness. The cost of PDDP is dominated by the cost of computing the splitting vector for each non-leaf node. The primary expense of the Lanczos-based solver is the computation of a matrix-vector product of the form $\mathbf{M}_{\mathtt{C}}\mathbf{v}$, where $\mathbf{v}$ is some vector. Therefore, the cost to split a root cluster which has m data samples with n attributes per sample is:

$$c_1 m \gamma n, \tag{4.6}$$

where c_1 represents the number of Lanczos iterations to convergence and γ is the fill fraction of the matrix. It is assumed that the product computation for sparse matrices ($\gamma << 1$) will take advantage of the sparseness present. At this point, the PDDP tree contains two leaf clusters, the left and right children of the root. If we assume for the sake of this analysis that these two leaf clusters will be split next, then the cost of computing the next two splits will be

$$c_1 m_{\mathtt{left}} \gamma n + c_1 m_{\mathtt{right}} \gamma n, \tag{4.7}$$

where $m_{\mathtt{left}}$ and $m_{\mathtt{right}}$ are the number of data items in the left and right children of the root, respectively. This is the same as the cost of splitting the root cluster. As long as the tree is constructed as a completely balanced binary tree, the cost of computing one new level of leaves in the tree is the same as the cost of computing the root split.

If we compute a balanced PDDP tree with k_f clusters, such that k_f is a power of 2, then the cost of producing the entire PDDP is

$$c_1 m \gamma n \log_2(k_f), \tag{4.8}$$

This analysis shows that PDDP is scalable in the sense that the cost of clustering is linear in the number of samples and the number of attributes, and logarithmic in the number of clusters. The scalability of PDDP has been demonstrated experimentally as well in [3]. Figure 3 is a graph for PDDP on document data sets of various sizes, and shows that PDDP is linear in the number of data samples and non-zero attributes.

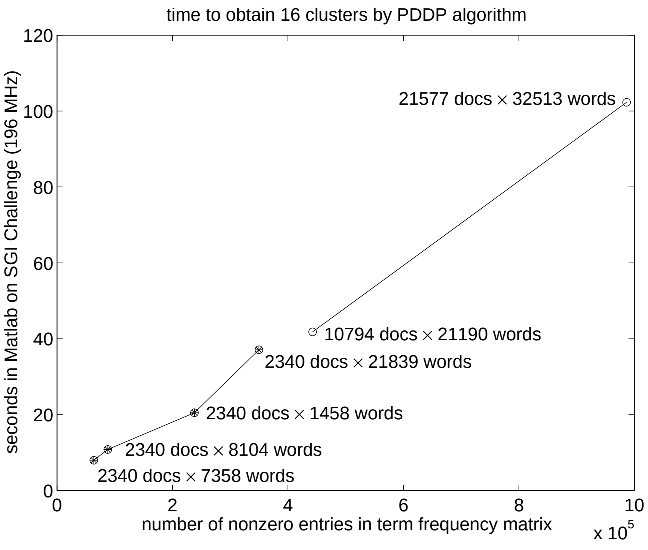

Figure 3: The time for PDDP clustering of various document data sets.

5 PMPDDP

PDDP requires the data set to be small enough to fit into memory for efficient computation, since the data are scanned many times during the iterative process used to obtain the rank 1 SVD. There are many large data sets which will not fit into the memory of most workstations, and therefore cannot be clustered quickly using PDDP as well as many other clustering algorithms. PMPDDP resolves this problem by creating a low-memory representation of the original data set that will fit into memory, and then clusters the low-memory representation using PDDP.

First, we describe the technique we use to obtain a low-memory approximation of a data set. Then, we show how we apply the technique to a data set which will not fit into memory at once. Last, we provide a complexity analysis of PMPDDP.

5.1 Matrix Approximation Using Cluster Centroids

Our approximation technique is based on the concept decomposition [7] The concept decomposition uses cluster centroids to form a basis for a least-squares approximation to the original data. Our method is similar, except that we limit the number of centroids which participate in the least-squares approximation to a given data item.

Suppose we have an $n \times m$ matrix $\mathbf{A}$ of data samples. We partition $\mathbf{A}$ into k_c clusters and compute the $\mathbf{c}_{k_c}$ centroids of the clusters. The centroids are used to define the $n \times k_c$ matrix $\mathbf{C}_A$,

$$\mathbf{C}_A = [\mathbf{c}_1 \; \mathbf{c}_2 \; \ldots \; \mathbf{c}_{k_c}], \tag{5.9}$$

which is the matrix of representative vectors used in the approximation. The expectation is that PDDP will be used to compute the clustering of $\mathbf{A}$ and therefore obtain $\mathbf{C}_A$, but this is not essential to the method.

We use $\mathbf{C}_A$ to compute the approximation

$$\mathbf{A} \approx \mathbf{C}_A \mathbf{Z}_A, \tag{5.10}$$

where $\mathbf{Z}_A$ is a $k_c \times m$ matrix. Each column $\mathbf{z}_i$ of $\mathbf{Z}_A$ approximates the corresponding column in $\mathbf{A}$ using a linear combination of the vectors in $\mathbf{C}_A$.

We only use a small number of the $\mathbf{c}_{k_c}$ vectors in $\mathbf{C}_A$ to form the approximation to each column of $\mathbf{A}$. Say we are approximating $\mathbf{a}_i$, where $\mathbf{a}_i$ is the i^{th} column of $\mathbf{A}$. We choose the k_z columns in $\mathbf{C}_A$ which are closest in Euclidean distance to $\mathbf{a}_i$ and use them to form the $n \times k_z$ matrix $\mathbf{C}_i$. Then we can compute the column $\mathbf{z}_i$ of $\mathbf{Z}_A$ such that:

$$\mathbf{z}_i = \arg\min_{\mathbf{z}} ||\mathbf{x}_i - \mathbf{C}_i \mathbf{z}||_2. \tag{5.11}$$

Using the k_z closest centroids to approximate each $\mathbf{a}_i$ instead of computing the approximation using all the centroids in $\mathbf{C}_A$ creates the opportunity to save memory with respect to a concept decomposition. If we set $k_z = k_c$, then our technique of creating an approximation is essentially identical to the concept decomposition.

To save memory with respect to the concept decomposition, a necessary, though not sufficient, condition is that $k_z < \min\{n, k_c\}$. Given this condition, the resulting least-squares problem is under-determined. If the k_z vectors in $\mathbf{C}_i$ are linearly independent, we use the normal equations with the Cholesky decomposition to solve the least-squares problem. Otherwise, we use the SVD to get the least-squares approximation of the data item. Even though there has been no attempt to create orthogonal basis vectors, in the majority of cases the normal equations give a satisfactory approximation.

PDDP has a unique advantage when clustering this low-memory representation. Recall that the Lanczos-based solver used to compute the principal direction is an iterative procedure which computes matrix-vector products of the form $\mathbf{Av}$, where $\mathbf{v}$ is some vector. If we replace $\mathbf{A}$ with $\mathbf{C}_A\mathbf{Z}_A$, and group the product computation as $\mathbf{C}_A(\mathbf{Z}_A\mathbf{v})$, then the principal direction can be computed without explicitly regenerating the sample vectors represented by $\mathbf{C}_A\mathbf{Z}_A$. Any clustering algorithm which requires a pairwise distance or similarity measure would not enjoy this advantage.

5.2 Constructing a Low-Memory Representation of a Matrix Now that we have a technique to construct a low memory representation of a data set, we need to apply it to larger data sets. The basic idea of what follows is to break the original data set into smaller pieces, construct a low-memory representation of each piece, and then assemble the individual representations into one system which represents the entire original data set.

We start with an $n \times m$ matrix $\mathbf{M}$ of data, such that $\mathbf{M}$ will not fit into memory at once. We want to construct a system $\mathbf{CZ}$ such that

$$\mathbf{M} \approx \mathbf{CZ}, \tag{5.12}$$

where $\mathbf{C}$ and $\mathbf{Z}$ will fit into memory and can be used to cluster the data in $\mathbf{M}$.

$\mathbf{C}$ and $\mathbf{Z}$ are constructed in a piecemeal fashion. $\mathbf{M}$ is divided into k_s disjoint *sections*

$$\mathbf{M} = [\mathbf{M}_1\ \mathbf{M}_2\ \ldots\ \mathbf{M}_{k_s}], \tag{5.13}$$

such that each section $\mathbf{M}_j$ of $\mathbf{M}$ will fit into memory. This partitioning of $\mathbf{M}$ is virtual since we assume only one section $\mathbf{M}_j$ will be in memory at any given instance. We also assume that the ordering of the columns of $\mathbf{M}$ is unimportant. We can now construct an approximation

$$\mathbf{M}_j \approx \mathbf{C}_j\mathbf{Z}_j \tag{5.14}$$

for each section $\mathbf{M}_j$ of $\mathbf{M}$ using the technique from §5.1.

After computing an approximation for each section of data, they can be assembled into the two-matrix system

$$\mathbf{C} = [\mathbf{C}_1\ \mathbf{C}_2\ \ldots\ \mathbf{C}_{k_s}] \tag{5.15}$$

$$\mathbf{Z} = \begin{bmatrix} \mathbf{Z}_1 & & & \\ & \mathbf{Z}_2 & & \\ & & \ddots & \\ & & & \mathbf{Z}_{k_s} \end{bmatrix}, \tag{5.16}$$

where $\mathbf{C}$ has dimension $n \times k_s k_c$ and $\mathbf{Z}$ has dimension $k_s k_c \times n$. The process of creating the low-memory representation of $\mathbf{M}$ is illustrated in Figure 4, and the parameters used to construct the low-memory representation are summarized in Table 1. The PMPDDP algorithm is shown in Figure 5

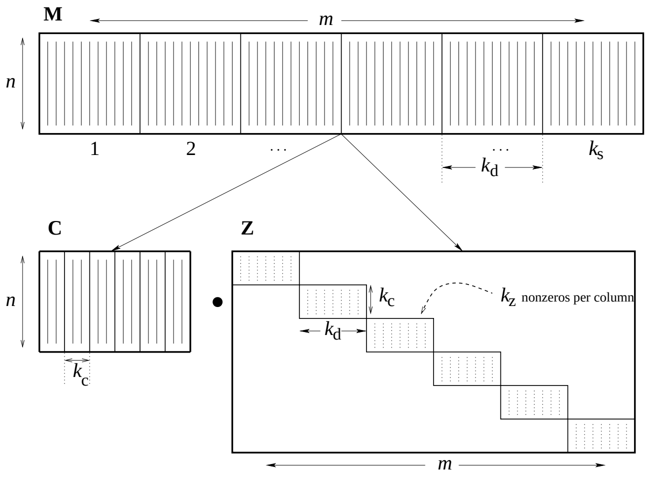

Figure 4: Construction details of the low-memory representation. $\mathbf{M}$ is divided into k_s sections, and the low-memory representation of each section is computed without referring to any other section. Each section is associated with a subdivision of $\mathbf{C}$ and $\mathbf{Z}$. The columns of a subdivision of $\mathbf{C}$ are the cluster centroids resulting from a clustering of the associated section. A column of a subdivision of $\mathbf{Z}$ is computed with a least-squares approximation to the corresponding column of $\mathbf{M}$, using the k_z closest centroids from the associated subdivision of $\mathbf{C}$.

5.3 PMPDDP Complexity Analysis We break down the cost of computing a PMPDDP clustering as being the cost of computing the low-memory representation of the data and the cost of clustering the low-memory representation. Constructing the low-memory representation for each section of data requires: (1) clustering the data in the section, (2) extracting the cluster centroids to use as a basis, (3) finding the centroids closest to each data item, and (4) computing a least-squares approximation for each data item. We present the cost for computing the low-memory representation of a given section, after which we will show the cost of clustering the low-memory representation of the entire data set.

Consider a $n \times k_d$ matrix $\mathbf{M}_j$ which defines a section. The cost of computing a PDDP clustering of this section with k_c clusters is:

$$c_1 k_d \gamma n \log_2(k_c), \tag{5.17}$$

parameter	description
m	total number of data items
n	number of attributes per data item
γ	fill fraction for the attributes
k_s	number of sections
k_d	number of data items per section
k_c	number of centroids per section
k_z	number of centroids approximating each data item
$\tilde{k}_z$	$\min\{k_z, \log_2(k_c)\}$
k_f	number of final clusters

Table 1: Definition of the parameters used in PMPDDP (see Figure 4).

which is the cost of a PDDP clustering of the section (4.8) with the appropriate parameters inserted. The $\mathbf{c}_{k_c}$ centroids of the clusters are extracted at negligible expense, and are used to form the matrix $\mathbf{C}_j$.

The $k_c \times k_d$ matrix $\mathbf{Z}_j$ is formed one column at a time. For a given data item $\mathbf{x}_i$ in $\mathbf{M}_j$, it is necessary to compute the distance from $\mathbf{x}_i$ to the k_c centroids in $\mathbf{C}_j$, choose the k_z centroids closest to $\mathbf{x}_i$, and compute the least-squares approximation to $\mathbf{x}_i$ using those centroids. The cost of computing the Euclidean distance from $\mathbf{x}_i$ to all the centroids in $\mathbf{C}_j$ is

$$\gamma n k_c, \tag{5.18}$$

and the cost of finding the k_z centroids closest to $\mathbf{x}_i$ is

$$\tilde{k}_z k_c, \tag{5.19}$$

where

$$\tilde{k}_z = \min\{k_z, \log_2(k_c)\}. \tag{5.20}$$

If $k_z < \log_2(k_c)$, the closest centroids are found by scanning all the distances. Otherwise, it is more efficient to sort the distances before selecting the centroids.

Now that the k_z closest centroids have been selected, the column $\mathbf{z}_i$ of $\mathbf{Z}_j$ which approximates $\mathbf{x}_i$ can be computed. The cost of finding the least squares approximation to $\mathbf{x}_i$ using the normal equations is

$$k_z^2 n + \frac{1}{3}k_z^3, \tag{5.21}$$

assuming that the centroid vectors are dense. The overall cost of computing $\mathbf{Z}_j$ for the k_d vectors in $\mathbf{M}_j$ is

$$k_d\left(k_c(\gamma n + k_z) + k_z^2 n + \frac{1}{3}k_z^3\right). \tag{5.22}$$

The cost of computing $\mathbf{C}_j$ (5.17) and $\mathbf{Z}_j$ (5.22) is the same for each section $\mathbf{M}_j$ of data, since the assumption

```
Algorithm PMPDDP.
0. Start with a $n \times m$ matrix $\mathbf{M}$, where each
       column of $\mathbf{M}$ is a data item, and set the
       values for $k_s$, $k_c$, $k_z$, and $k_f$ (see Table 1).
1. Partition $\mathbf{M}$ 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 $\mathbf{C}$ and $\mathbf{Z}$ as in
       (5.15, 5.16) 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 $\mathbf{CZ}$
       with $k_f$ clusters.
8. Result: A binary tree with $k_f$ leaf nodes
       forming a partitioning of the entire data set.
```

Figure 5: PMPDDP algorithm.

is that k_d, k_c, and k_z are the same for each section. The approximations to each section are assembled into $\mathbf{CZ}$ at negligible cost.

At this point, we have a low-memory representation of the original data set. We use PDDP to cluster $\mathbf{CZ}$ to get a clustering of $\mathbf{M}$. Replacing the original data matrix $\mathbf{M}$ with the factored form $\mathbf{CZ}$ means that the matrix-vector product $\mathbf{Mv}$ will be replaced by the product $\mathbf{C}(\mathbf{Zv})$ when computing the principal direction of the root cluster of the PDDP tree.

We start the cost analysis of clustering $\mathbf{CZ}$ with the cost of splitting of the root cluster. The cost of computing the matrix-vector product associated with $\mathbf{C}$ is

$$c_2 k_s k_c n, \tag{5.23}$$

and the cost of computing the matrix-vector product associated with $\mathbf{Z}$ is

$$c_2 k_z m, \tag{5.24}$$

where c_2 is the number of Lanczos iterations to convergence. Since $\mathbf{Z}$ is sparse, only the non-zero entries in $\mathbf{Z}$ will participate in the product computation.

There are now two leaf clusters in the tree, which are the left and right children of the root. We will assume that these two clusters are split next, which creates the next level in the PDDP tree. The total cost of computing these next two splits is

$$2c_2 k_s k_c n + c_2 k_z m_{\mathtt{left}} + c_2 k_z m_{\mathtt{right}}, \tag{5.25}$$

where $m_{\mathtt{left}}$ and $m_{\mathtt{right}}$ are the number of data items in the left and right children of the root, respectively. The cost of computing the matrix-vector product associated with $\mathbf{C}$ (5.23) is the same for every cluster, no matter the number of items in the cluster. The reduction in cost for splitting smaller clusters seen in the analysis of PDDP is only reflected in the terms associated with computing the matrix-vector product associated with $\mathbf{Z}$ (5.24). So unlike the results for PDDP on the original data matrix, the cost of clustering $\mathbf{CZ}$ increases with each new level in the tree.

Again, we will assume for the purpose of cost analysis that we split the clusters such that we create a full level in the PDDP tree before we create any leaves on the next level. We will again choose a number of final clusters k_f such that k_f is a power of 2. Therefore, the total cost of computing a PDDP tree using the product representation $\mathbf{CZ}$ in place of $\mathbf{M}$ is

$$c_2(k_f - 1)k_s k_c n + c_2 \log_2(k_f) k_z m. \tag{5.26}$$

The $(k_f - 1)$ term reflects the fact that to produce k_f clusters, $(k_f - 1)$ splits must be computed. This means that the matrix-vector product associate with $\mathbf{C}$ must be computed $(k_f - 1)$ times.

The result in (5.26) is an upper bound on the cost of PMPDDP. Once the clusters become small enough, it is less expensive to regenerate the original data and compute the splits using ordinary PDDP. The condition required for this approach to be less expensive is

$$n k_z m_c + c_2 n m_c < c_2 k_s k_c n + c_z k_z m_c, \tag{5.27}$$

where m_c is the number of data items in the cluster, $n k_z m_c$ is the cost of regenerating the data items, and $c_2 n m_c$ is the cost of computing the split using the regenerated data. The equation on the right hand side of the inequality represents the cost of computing the split using the factored form, as in (5.23,5.24). Algebraic manipulation can be used to expose the condition

$$m_c < \frac{c_2 k_s k_c}{k_z + c_2\left(1 - \frac{k_z}{n}\right)}. \tag{5.28}$$

In practice, it is possible to keep track of the number of iterations being performed for each split, and use that number and (5.28) to estimate when it would be advisable to reconstruct the data before determining the split. Otherwise, the data might be reconstructed once the memory required is below a certain threshold. It is possible that for a given data set, the relation in (5.28) may never be satisfied.

Table 2 collects the cost for all the operations performed by PMPDDP. Depending on the number of attributes n in the data set and the choice of parameters, clustering the approximation can be less expensive than clustering using the original data set. The bulk of the expense of PMPDDP is in obtaining the low-memory representation of the original data.

Operation	Cost
Obtaining $\mathbf{C}$	$c_2 m \gamma n \log_2(k_c)$
Computing $\mathbf{Z}$	$m\left(k_c k_z \gamma n + k_c \tilde{k}_z\right) + k_z^2 n + \frac{1}{3}k_z^3\Big)$
Clustering $\mathbf{CZ}$	$c_2(k_f - 1)k_s k_c n + c_2 \log_2(k_f) k_z m$

Table 2: Collected costs of producing a PMPDDP clustering. See Table 1 for a definition of the parameters.

6 Experiments

We selected four real data sets to measure the performance of PMPDDP. They were chosen to demonstrate that PMPDDP will cluster the data using the low-memory approximation as accurately as PDDP can using the original data set. All of the data can be clustered with PDDP on a machine with 1 GB of memory.

The experiments focused on comparing the performance of PMPDDP when using multiple centroids to approximate each data item, as compared to constructing a low-memory representation using the centroid closest to the data item to approximate the data item. We also include some experiments comparing PMPDDP to k-means, with k-means being performed on both the original data set as well as the low-memory approximation.

6.1 Data Sets

The ISOLET (Isolated Letter Speech Recognition) data set was taken from the UCI Machine Learning Repository [13]. The 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 [8].

The k1 data set [3] 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 stop words removed, and then the words were stemmed using Porter's suffix stripping algorithm [9]. The document vectors were scaled to until length, but no other scaling was performed.

The Reuters-21578 data set [12] is a collection of news articles. The original data set has 21578 items with 19968 total words in the dictionary. We chose to use only the 9494 documents which were associated with a single topic per document. There were a total of 66 categories. The documents were processed in the same manner as the k1 document set.

The forest cover data set [10] consists of both continuous and binary attributes associated with the types of forest cover in a 30x30 meter square area. There are 10 continuous attributes associated with measurements such as position and elevation, 40 binary attributes associated with soil type, and 4 binary attributes associated with wilderness area. There were 7 forest cover types in the classification. Each data item had 54 attributes, 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 was scaled to have a mean of zero and a variance of one.

6.2 Effect of Number of Approximating Centroids For this set of experiments, we fixed all the parameters except k_z to the values shown in Table 3. The parameter values used are representative of a typical application of PMPDDP. They are not the optimal values. As a basis of comparison, we constructed a representation $\mathbf{CZ}_{\mathsf{cc}}$ that used the same construction techniques as $\mathbf{CZ}$, but instead of computing a least-squares approximation to each data item, $\mathbf{CZ}_{\mathsf{cc}}$ approximates each original data item with the closest centroid.

The results for the entropies of the four data sets with varying k_z are shown in Figure 6. They are normalized with respect to a PDDP clustering of the approximation $\mathbf{CZ}_{\mathsf{cc}}$ using the same k_f. Therefore, any time a value drops below 1, using a least-squares approximation with k_z centroids is an improvement over using the closest centroid. For the data examined, in the context of PDDP clustering, not computing a least-squares approximation when using one centroid gives better results. For the isolet and reuters data, using more than one centroid to approximate each data item gives better clustering results than using just one centroid, but using 3 to 4 centroids appears to be as good as using more. The k1 results are erratic, with an advantage to using 9 to 11 centroids to approximate the data, and no advantage otherwise. The forest cover data is best served by using the closest centroid to approximate the data. Recall that the forest cover data set has only 7 categories, and many of the attributes were converted from discrete values. Many of the original data items are very likely to be quite close together. This means that using the closest centroid probably results in a very accurate representation of the original data items.

dataset	isolet	k1	reuters	forest
m	7997	2340	9494	581012
n	617	21839	19968	54
categories	26	20	66	7
γ	dense	0.68%	0.20%	dense
k_s	5	5	5	5
k_c	150	50	100	500
k_z	5	5	5	1
k_f	150	50	100	500

Table 3: Datasets and parameter values used for PMPDDP experiments. The isolet data is from Murphy and Aha [13], the k1 document data is from [3], the reuters data is the standard reuters data set [12] selecting the items which had only one topic assigned to them, and the forest data is from [10].

We also compared a PMPDDP clustering to a standard PDDP clustering with the same value for k_f. The results are shown in Table 4. In every case, the clustering quality with respect to entropy is improved somewhat when using PMPDDP. The memory savings are less striking with the document data sets. Dense data sets seem better able to take advantage of the memory savings. However, a larger document data set will probably see a higher percentage of memory savings, since the number of possible attributes in document data sets is limited by the number of words in the dictionary, which usually levels off with increasing numbers of documents. As expected, PMPDDP is more expensive than PDDP. However, PMPDDP has the ability to cluster data sets which PDDP cannot cluster without the added expense of paging. Note how the approximation error

$$\frac{||\mathbf{CZ} - \mathbf{M}||_F}{||\mathbf{M}||_F} \tag{6.29}$$

can be very high, yet the approximation is sufficiently accurate to provide a good clustering of the data set.

6.3 Comparison of PMPDDP with K-means The second set of experiments compares PMPDDP with a well-known clustering method, k-means. K-means has the advantage of being simple to implement, and the drawback that it is usually necessary to perform many

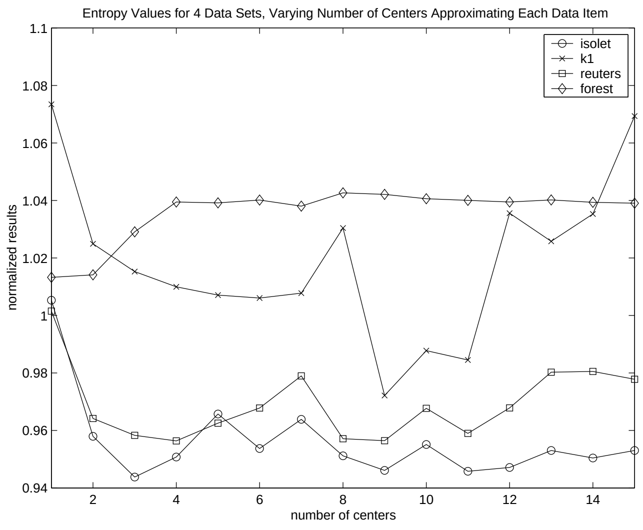

Figure 6: Entropy values for four data sets with an increasing number of centroids k_z approximating each data item. The results are normalized with respect to a PDDP clustering of $\mathbf{CZ}_{\mathsf{cc}}$, where Z_{cc} was constructed using the closest centroid to approximate each data item with no least-squares computation.

dataset	isolet	k1	reuters	forest
PDDP entropy	.814	.982	.799	.689
PMPDDP entropy	.810	.960	.787	.672
PDDP time	34.41	14.19	17.63	196.01
PMPDDP time	66.20	70.62	137.99	476.83
CZ memory % of M	10.9	57.2	68.4	4.13
CZ error %	37.9	74.3	72.0	15.9

Table 4: Comparison of PDDP and PMPDDP clusterings for the four data sets, using the parameters shown in Table 3. The approximation error was computed as $||\mathbf{CZ} - \mathbf{M}||_F / ||\mathbf{M}||_F$.

random starts of k-means in order to insure that a good clustering has been found.

Previous results [14] have demonstrated that using a PDDP clustering to initialize k-means can give a good k-means clustering without requiring restarts. We used the PMPDDP clusterings with the parameters in Table 3 as a starting point, and ran 1 k-means trial per data set to 40 iterations, using the clustering found by PMPDDP as a starting point. The k-means clustering was performed using the low-memory representation of the data.

As a basis for comparison, we ran 30 random-start k-means trials using the $\mathbf{CZ}$ approximation to the data computed with the parameters shown in Table 3, and averaged the results. Each trial was taken to 40 iterations, whether or not it converged. We believe this is a realistic application of k-means to large data sets, since most of the data point movement occurs early, and true convergence can take an extremely long time. The smaller data sets often converged before 40 iterations, but other tests on the forest cover data didn't show convergence in even 100 iterations.

The entropies for the k-means trials are shown in Figure 7, and are normalized with respect to the averaged random-start trials mentioned in the previous paragraph. K-means gives a better clustering than PMPDDP. However, using PMPDDP clusters as an initializer to k-means can give better results than either method will individually. The only data set which didn't show even slightly better results when using the PMPDDP clusters as a starting point was the forest cover data set. The isolet data apparently clusters well with k-means, so little improvement was noted. The two document data sets, k1 and reuters, had a better than average clustering after only 1 iteration of k-means using the PMPDDP clusters to start. The results indicate that using PMPDDP clusters as a starting point for k-means clustering will at least find an average k-means clustering in very few iterations as compared to random-start k-means, and does not require repetition to insure good results.

We also ran experiments for k-means run on the original data sets. We carried the k-means to 40 iterations, and averaged the results over 30 trials. A comparison with random-start k-means using the $\mathbf{CZ}$ representation of the data is shown in Table 5. For three of the data sets examined, k-means has better entropy values when using the original data, as compared to using the approximation. Only the k1 data had better results when clustering the approximation.

If the results in Table 5 are compared with those in Table 4, it can be seen that k-means on the original data produced a better clustering with respect to en-

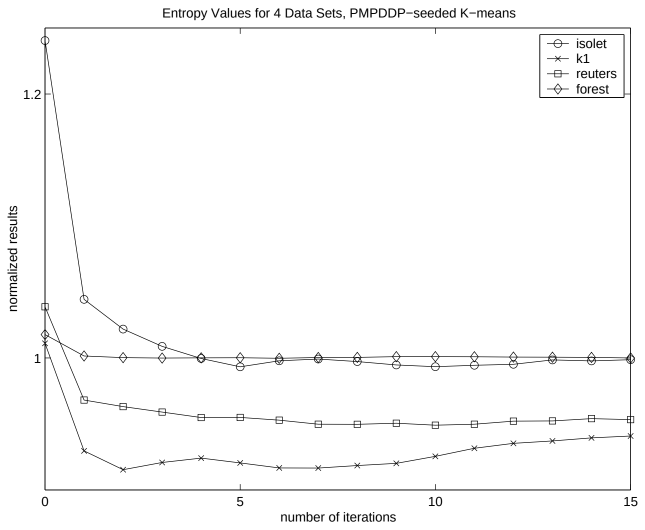

Figure 7: Entropy values for the four data sets using PMPDDP clusters as the initial centroids for k-means, with the parameters from Table 3. K-means was performed using the approximation $\mathbf{CZ}$ in place of the original data. The results are normalized with respect to the entropy after 40 iterations of random-start k-means using the same approximation $\mathbf{CZ}$, averaged over 30 trials.

tropy than PMPDDP for every data set except the k1 data. The isolet, reuters, and forest data cluster well on average using k-means. However, k-means typically requires many iterations to compute a good clustering of the data, and many random starts to find a good clustering, although [5] provides a method to get initial centroids which does not require random restarts. PMPDDP is faster and does not require any restarts to insure that a good clustering has been found.

The iteration times were quite high when clustering the approximations of the document data sets, k1 and reuters. The k-means code is virtually identical for both cases, so we expect that there is a speed issue associated with a sparse-matrix multiplication of the approximation. The distances between the cluster centroids and the data points for both cases were computed using blocking, with 50 columns per block. Therefore, when clustering the low-memory representation, only 50 columns of data were reconstructed in memory at any given time.

6.4 Scalability of PMPDDP We ran an experiment to determine the scalability of PMPDDP. The forest data set was used since it is the largest data set we considered in this work. We used the same parameters as in Table 3, but computed the PMPDDP tree starting

dataset	isolet	k1	reuters	forest
k-means on M entropy	.523	1.01	.668	.656
k-means on CZ entropy	.653	.949	.758	.659
k-means on M time	91.31	34.65	92.17	5156.22
k-means on CZ time	89.61	356.73	1171.26	4436.21

Table 5: Comparison of k-means clusterings on the original data $\mathbf{M}$ and on the approximation $\mathbf{CZ}$, where $\mathbf{CZ}$ was constructed using the parameters shown in Table 3. K-means was run for 40 iterations, and the results were averaged over 30 trials.

with 50,000 random samples, and increased the sample size by 50,000 at every step until we clustered the entire data set. The results for the amount of time taken to cluster the data are shown in Figure 8. The results show that PMPDDP is linear in the number of data items for this data set.

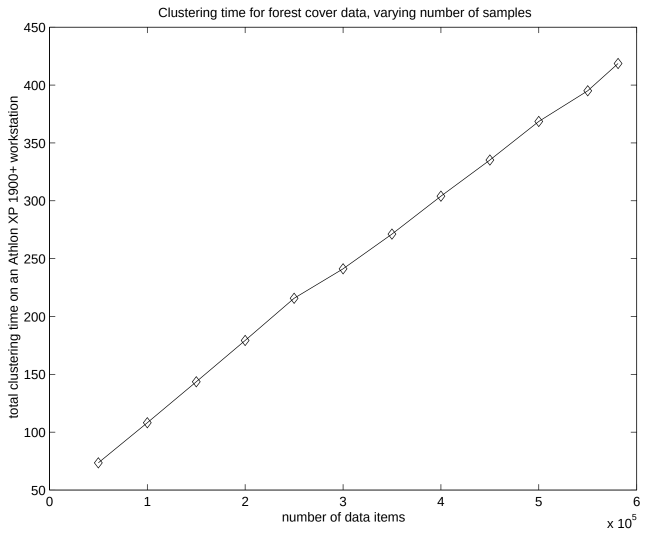

Figure 8: PMPDDP clustering time using various subsample sizes of the forest cover data set. The remaining parameters are the same as in Table (3).

7 Conclusion

Approximations to the data are often constructed as an aid when clustering large data sets. The usual solution is to create vectors which resemble cluster centroids, and then assign each original data item to the closest vector. These approximations are either creating during a clustering process, or used to cluster the data once they are constructed.

We presented PMPDDP, which is clustering algorithm which uses more than one vector to approximate each data item. The data were divided into small pieces, and an approximation to each piece was constructed using the centroids from a clustering of the piece of data. Each data item is examined during the process which computes the low-memory representation, and no data items are removed from consideration. The approximations were then gathered and clustered at once to produce a clustering of the entire original data set.

PMPDDP was able to cluster the data to greater accuracy than PDDP for the data sets examined. The low-memory representation of the original data saved a significant amount of memory while still providing a unique representation for each data item.

We also demonstrated that the low-memory representation can be used to find a k-means clustering of the data. Given sufficient time and random restarts, k-means found a better clustering than PMPDDP. However, the combination using PMPDDP clusters as the starting point for k-means can produce clusterings either equal to or superior than k-means on its own in less time and without requiring random restarts.

8 Acknowledgements

This research was partially supported by NSF grant IIS-0208621.

References

[1] M. W. Berry, S. T. Dumais, and Gavin W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37:573–595, 1995.

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

[3] 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.

[4] 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.

[5] 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.

[6] Douglass R. Cutting, Jan O. Pedersen, David Karger, and John W. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 318–329, 1992.

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

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

[9] 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.

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

[11] 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.

[12] D. Lewis. Reuters-21578. `http://www.research.att.com/~lewis`, 1997.

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

[14] S. M. Savaresi and D. L. Boley. Bisecting k-means and PDDP: a comparative analysis. In *First SIAM International Conference on Data Mining*, April 2001. to appear.

[15] 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.

[16] Z. Zhang, H. Zha, and H. Simon. Low-rank approximations with sparse factors I: Basic algorithms and error analysis. *SIAM J. Matrix Anal.*, 23:706–727, 2002.