Clustering Very Large Data Sets with Principal Direction Divisive Partitioning

David Littau[1] and Daniel Boley[2]

[1] University of Minnesota, Minneapolis MN 55455 littau@cs.umn.edu
[2] University of Minnesota, Minneapolis MN 55455 boley@cs.umn.edu

1 Introduction

One of the challenges in data mining is the clustering of very large data sets. We define a very large data set as a data set which will not fit into memory at once. Many clustering algorithms require that the data set be scanned many times during the clustering process. If the data cannot fit into memory, then the data must be repeatably re-scanned from disk, which can be expensive.

One approach to clustering large data sets is to adapt clustering algorithms suitable for small data sets to much larger data sets. There are two popular methods used to adapt clustering algorithms to large data sets. The first technique is to extract a sub-sample of the data, such that the sub-sample is small enough to fit into available memory and be clustered. Other techniques to accelerate the clustering process are often applied at the same time. Once a clustering is obtained, the remaining data points can be assigned to the clusters with the closest centroid. The major drawbacks to sampling are that it can be difficult to know if a given subsample is a representative sample and therefore provides an accurate clustering, and that the outliers will usually be ignored.

The second technique commonly used to adapt clustering algorithms to large data sets, as in [5, 6, 21], is to approximate a given data item by assigning it to a single representative vector. One representative vector may take the place of an arbitrary number of data items. Once a data item has been assigned to a representative, it is no longer possible to differentiate it from any other data item assigned to the same vector. Thus, the resolution of any clustering of the data is limited by the granularity of the representatives.

We propose an alternate approach to adapt the Principal Direction Divisive Partitioning (PDDP) clustering method [3] to very large data sets. We create a Low-Memory Factored Representation (LMFR) of the data, and then cluster the LMFR using PDDP. Every data point has a unique representation in the LMFR, and every data point is examined during the construction of the LMFR. The LMFR is constructed piecewise using samples of the data,

such that the samples will fit into memory. The samples are selected without replacement, and selection continues until the data set is exhausted. Once an approximation has been constructed for each sample, the approximations are assembled into an LMFR representing the entire data set.

The LMFR avoids what we claim are the major drawbacks to other techniques. All data are examined during the construction of the LMFR, which is not the case when sub-samples are clustered as representative of the entire data set. Each data item has a unique representation in the LMFR, so the granularity of the clustering can be finer than that achieved by a method which assigns many data items to a single representative vector. Every data item is examined and participates in the construction of the LMFR, so outliers will not be ignored. Furthermore, since the method is deterministic, we need not be concerned that other, perhaps better clusterings could be constructed.

The remainder of the chapter is as follows. First, we provide some background on a few of the methods available to cluster large data sets. Next, we describe the technique used to construct the LMFR. Then, we describe how the original representation of the data can be easily replaced by the LMFR in the PDDP method, a process we call Piecemeal PDDP (PMPDDP). Finally we show some experimental results which demonstrate that the clustering quality of the PMPDDP method is similar to PDDP, and that PMPDDP maintains the scalability of PDDP.

2 Background

The problem of clustering very large data sets is an active area of research. Many approaches adapt existing clustering methods such as hierarchical agglomeration [11] and k-means[9, p201] to much larger sets. There are also clustering methods which were specifically designed from the ground up to be used for large data sets. Note that the following is a sketch of some clustering methods for large data sets, and is not intended to be taken as exhaustive.

2.1 Sampling

Before we describe any specific methods, we describe sampling. Sampling is a general approach to extending a clustering method to very large data sets. A sample of the data is selected and clustered, which results in a set of cluster centroids. Then, all data points are assigned to the closest centroid. Many large data set clustering methods use sampling to overcome time and memory limitations.

2.2 Hierarchical Agglomeration and its Variants

Hierarchical agglomeration [11] produces a hierarchy of clusters, such that any given level of cluster refinement can be selected from the results. It starts

with singleton clusters, and produces the hierarchy of clusters by successively merging the two clusters which are closest. Typically, the distances between clusters are determined by computing the distance from every point in a given cluster to every other point in every other cluster. The expense of computing the distances between all points is the most serious drawback to the method.

Scatter/Gather [6] speeds up agglomeration by dividing the data into buckets and agglomerating individual buckets until the number of clusters in a given bucket is reduced by a specific amount. The clusters are replaced by their weighted centroids, and the centroids from all buckets are then placed in a smaller set of buckets and agglomerated again. The process continues until a specified number of centroids are created, after which all data are assigned to the closest centroid. While this method was specified as a speed-up for data sets which would fit into memory, combining Scatter/Gather with sampling would make it appropriate for large data sets. Alternately, the amount of data in a given bucket could be sized to fit into memory, and only one bucket of data would appear in memory and be agglomerated at a given time. The resulting centroids could be saved to disk, and another bucket of data could then be loaded and agglomerated, and so on. This would require some additional disk access, but would result in a method which could work for arbitrarily large data sets.

Cure [12] adapts hierarchical agglomeration by using a small set of well-scattered points to compute the distances between clusters, rather than considering all the points in a cluster when computing the distances between clusters. This significantly speeds up the procedure. Sub-sampling the data was also specified when the data set was too large.

There are other extensions of hierarchical agglomeration. For instance, [10] uses maximum likelihood determined by a multivariate Gaussian model to decide which two clusters should be merged. The work in [15] uses a heap to store the distances between all pairs to speed up access to distance information. Refinement of the clusters to increase quality is described in [14]. While these methods were designed to enhance the speed and quality of hierarchical agglomeration, combining them with sub-sampling the data would make them suitable for clustering very large data sets.

2.3 *K*-means and its Variants

K-means produces clusters using an iterative method. A random set of starting centroids is selected from the data set, and all data points are assigned to the closest centroid. Then, new centroids are computed using the data points in each cluster, and again all data points are assigned to the closest centroid. The process continues until there is no further data movement. Multiple passes with random restarts are usually performed to ensure a good clustering has been found.

One adaptation of k-means to very large data sets is provided in [5]. Samples are drawn from the data set, without replacement, and clustered. The

data points in a given cluster are replaced by a representative which is much like a weighted centroid, but provide a bit more information. This is done for all current clusters. Then more samples are drawn from the data set, and are clustered along with the weighted centroids. The process continues until the data set is exhausted or the centroids stop moving.

It is difficult to know good choices for initial centroids for k-means. Instead of repeating k-means with random restarts, [4] provides a technique to generate good candidate centroids to initialize k-means. The method works by selecting some random samples of the data and clustering each random sample separately using k-means. The centroids from each clustering are then gathered into one group and clustered to create a set of initial centroids for a k-means clustering of the entire data set.

There are other variants of k-means. The work in [19] uses a k-d tree to organize summaries of the data. It is fast, but does not perform well for dimensions higher than eight. The work in [1] used a k-d tree to cut down on the number of distance computations required, though it isn't clear if the application is limited to spatial data. We assume it is, since they are using the same kind of data structure as in [19], and their experiments were conducted on low-dimension data. Therefore these methods are more appropriate for large low-dimension data sets.

Very fast k-means (VFKM) [8] takes a different approach from other clustering methods. The stated desire is to produce a model (clustering) using a finite amount of data that cannot be distinguished from a model constructed using infinite data. The error is bounded by comparing the centroids resulting from different k-means clusterings using different sample sizes. Stated in a very simplified manner, if the centroids from the different clusterings are within a specified distance of each other, they are considered to be the correct centroids. Otherwise a new, larger sample is drawn from the data set and clustered, and the resulting centroids are compared to the centroids obtained from the previous run. The authors suggest that this method is not a reasonable approach unless the database begin clustered contains millions of items.

2.4 Summary of Cited Clustering Methods

Most of the extensions of hierarchical agglomeration were designed to speed up the process for data sets which can fit into memory. Sampling was indicated when the data sets grew too large. Sampling the data ignores outliers, which may be interesting data items in some circumstances. Also, it is difficult to know whether a truly representative sample has been drawn from the data set.

The extensions of k-means to large data sets either drew samples or assigned many data points to one representative vector. Using one vector to approximate many data items, as in [5, 6, 21], is a relatively popular technique when constructing approximations to the data. However, once the assignments have been made, there is no way to distinguish between the data

items assigned to a given representative. The resolution of any clustering of the data is limited by the resolution of the representatives.

In the clustering method we present in this chapter, no sampling of the data is necessary. All data items are exposed to the method. Each data item has a unique representation in the approximation we construct. Therefore, the resolution of the clustering is not limited by the approximation of the data. We believe these differences result in a method which provides a useful alternative to other large data set clustering methods.

3 Constructing a Low-Memory Factored Representation

The Low-Memory Factored Representation (LMFR) is comprised of two matrices. The first matrix contains representative vectors, and the second matrix contains data loadings. The representative vectors are the centroids obtained from a clustering of the data. The data loadings are a least-squares approximation to each data item using a small number of selected representative vectors.

Since the data set is assumed to be too large to fit into memory, we divide it up into smaller samples we call *sections*. Each data item from the original representation appears once and only once across all sections. We individually compute a LMFR for each section.

First, we describe the method used to obtain the LMFR for one section of data. Then, we describe how we assemble the LMFRs for each section into one LMFR which represents the entire data set.

3.1 Constructing an LMFR for One Section

Suppose we have an $n \times m$ matrix $\mathbf{A}$ of data samples, such that $\mathbf{A}$ comfortably fits into memory at once. We want to compute the factored representation

$$\mathbf{A} \approx \mathbf{C}_A \mathbf{Z}_A, \tag{1}$$

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

The first step in computing this factored form of $\mathbf{A}$ is to obtain the matrix of representative vectors $\mathbf{C}_A$. To accomplish this, we partition $\mathbf{A}$ into k_c clusters and compute the k_c centroids of the clusters. These centroids are collected into a $n \times k_c$ matrix $\mathbf{C}_A$,

$$\mathbf{C}_A = [\mathbf{c}_1 \ \mathbf{c}_2 \ \ldots \ \mathbf{c}_{k_c}]. \tag{2}$$

We use the PDDP method to compute the clustering of $\mathbf{A}$ and therefore obtain $\mathbf{C}_A$, since PDDP is fast and scalable. In principle, any clustering method could be used to obtain the components of $\mathbf{C}_A$.

The matrix of data loadings $\mathbf{Z}_A$ is computed one column at a time. In approximating each column $\mathbf{a}_i$ of $\mathbf{A}$, we use only a small number (k_z) of the representatives in $\mathbf{C}_A$. Therefore, each column $\mathbf{z}_i$ in $\mathbf{Z}_A$ has only k_z nonzero entries. For example, to approximate $\mathbf{a}_i$, we choose the k_z columns in $\mathbf{C}_A$ which are closest in Euclidean distance to $\mathbf{a}_i$ and implicitly collect them into an $n \times k_z$ matrix $\mathbf{C}_i$. Then the nonzero entries in the column $\mathbf{z}_i$ are obtained by solving for the k_z-vector $\hat{\mathbf{z}}_i$:

$$\hat{\mathbf{z}}_i = \arg\min_{\mathbf{z}} ||\mathbf{a}_i - \mathbf{C}_i\mathbf{z}||_2^2. \tag{3}$$

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. If the normal equations fail, we use the more expensive SVD to get the least-squares approximation of the data item. Even though there has been no attempt to create orthogonal representative vectors, in the majority of cases the normal equations are sufficient to solve the least-squares problem. The LMFR algorithm is shown in Figure 1.

Algorithm `LMFR`.
0. **Start** with a $n \times m$ matrix $\mathbf{A}$, where each column of $\mathbf{A}$ is a data item, and set the values for k_c, the number of representative vectors in $\mathbf{C}$, and k_z, the number of representatives used to approximate each data item.
1. **Partition A** into k_c clusters
2. **Assemble** the k_c cluster centroids from step 1 into an $n \times k_c$ matrix $\mathbf{C}_A$ (eqn. (2) in the text).
3. **For** $i = 1, 2, \ldots, m$ **do**
4. **Find** the k_z columns in $\mathbf{C}_A$ closest to $\mathbf{a}_i$
5. **Collect** the k_z columns found in step 4 as the $n \times k_z$ matrix $\mathbf{C}_i$
6. **Compute** $\hat{\mathbf{z}}_i = \arg\min_{\mathbf{z}} ||\mathbf{a}_i - \mathbf{C}_i\mathbf{z}||_2^2$
7. **Set** the i^{th} column of $\mathbf{Z}_A = \hat{\mathbf{z}}_i$
8. **Result** $\mathbf{C}_A$ and $\mathbf{Z}_A$, which represent a factorization of $\mathbf{A}$

Fig. 1. LMFR algorithm.

When $k_z = k_c$, this factorization of $\mathbf{A}$ is essentially identical to the *concept decomposition* [7], except that we use PDDP to obtain the clustering rather than spherical k-means. We typically select a value for k_z such that $k_z \ll k_c$, which can result in significant memory savings. Since the memory savings are dependent on $\mathbf{Z}$ being sparse, we also require the condition that $k_z \ll n$. Thus a low-dimension matrix is not a good candidate for this factorization technique from a memory-savings standpoint.

To obtain memory savings, it is also necessary to control the size of $\mathbf{C}_A$, which is done by making k_c as small as possible. There is a trade-off between

the two parameters k_c and k_z, since for a given amount of memory available to contain the LMFR $\mathbf{C}_A\mathbf{Z}_A$, increasing one of the parameters requires decreasing the other.

3.2 Constructing a LMFR of a Large Data Set

Once a LMFR has been computed for each section, they are assembled into a single factored representation of the entire original data set. A graphical depiction of this technique is shown in Figure 2. What follows is a formal definition of the entire process of constructing the LMFR of a large data set.

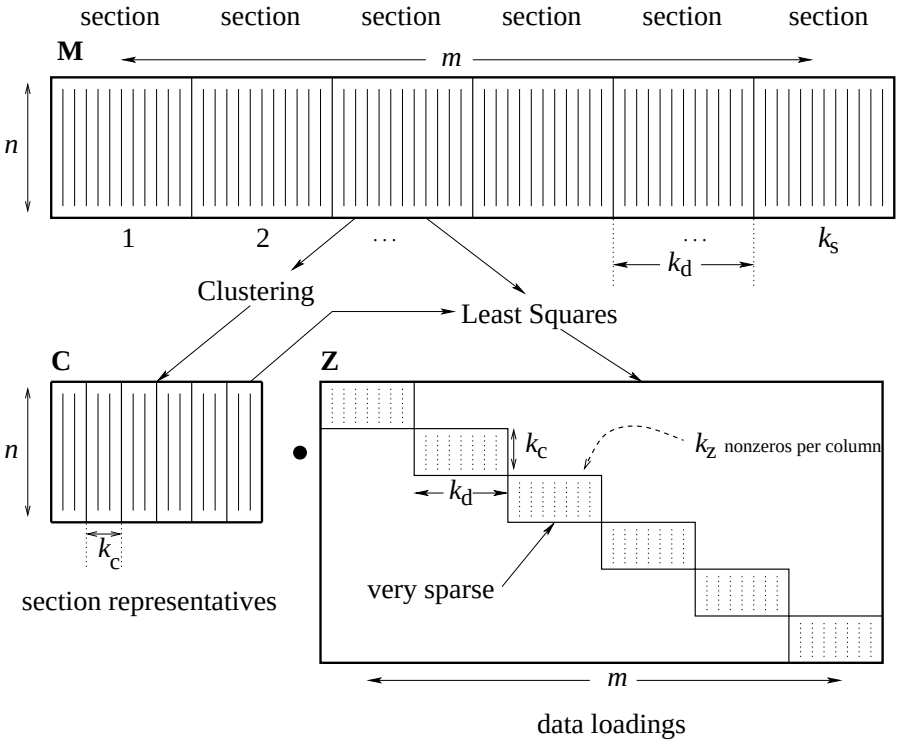

Fig. 2. Construction details of the low-memory representation. **M** is divided into k_s sections, and the low-memory representation of each section is computed without referring to any other section. Each section is associated with a subdivision of **C** and **Z**. The columns of a subdivision of **C** are the cluster centroids resulting from a clustering of the associated section. A column of a subdivision of **Z** is computed with a least-squares approximation to the corresponding column of **M**, using the k_z closest centroids from the associated subdivision of **C**.

We consider the original representation of the data set as an $n \times m$ matrix **M**, such that **M** will not fit into memory at once. We seek the single factored

representation $\mathbf{CZ}$ such that

$$\mathbf{M} \approx \mathbf{CZ}, \tag{4}$$

where $\mathbf{C}$ and $\mathbf{Z}$ will fit into memory and can be used to cluster the data in $\mathbf{M}$. Since $\mathbf{M}$ cannot fit into memory at once, $\mathbf{M}$ is divided into k_s disjoint sections

$$\mathbf{M} = [\mathbf{M}_1\ \mathbf{M}_2\ \ldots\ \mathbf{M}_{k_s}], \tag{5}$$

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 a LMFR

$$\mathbf{M}_j \approx \mathbf{C}_j\mathbf{Z}_j \tag{6}$$

for each section $\mathbf{M}_j$ of $\mathbf{M}$ using the technique from §3.1. After computing the approximation (6) for each section of data, they can be assembled into the two-matrix system

$$\begin{aligned} \mathbf{C} &= [\,\mathbf{C}_1 \quad \mathbf{C}_2 \quad \ldots \quad \mathbf{C}_{k_s}\,] \\ \mathbf{Z} &= \begin{bmatrix} \mathbf{Z}_1 & & & \\ & \mathbf{Z}_2 & & \mathbf{0} \\ & & \ddots & \\ \mathbf{0} & & & \mathbf{Z}_{k_s} \end{bmatrix}, \end{aligned} \tag{7}$$

where $\mathbf{C}$ has dimension $n \times k_s k_c$ and $\mathbf{Z}$ has dimension $k_s k_c \times n$. We call this system a general LMFR. The parameters used to construct the general LMFR are summarized in Table 1.

Table 1. Definition of the parameters used in constructing a general LMFR.

parameter	description
m	total number of data items
n	number of attributes per data item
γ_1	fill fraction of the attributes in $\mathbf{M}$
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

Note that the idea of processing the data in separate pieces and assembling the results has been done previously in the context of principal component analysis [13]. However, in that case the application was intended for situations in which different sets of attributes for a given data point were distributed across separate databases. The LMFR construction method is designed to process data points that have all attributes present.

3.3 Applications of the LMFR

The only application of the LMFR which we will be covering in this work is using the LMFR to extend PDDP to large data sets. However, this is not the only successful application of the LMFR in data mining. In [18] we showed an adaptation of the LMFR to general stream mining applications. The LMFR allows for more of the stream to be exposed to a given stream mining method at once.

Another application we have investigated is using the LMFR for document retrieval [16]. We demonstrated that we could construct an LMFR of a given data set that had better precision vs. recall than an SVD of a specific rank, while taking less time to construct and occupying less memory than the SVD. Specifically, the LMFR with $k_z = 5$ and $k_c = 600$ for a 7601 item data set took 187 seconds to construct and occupied 11.32 MB of memory, while a rank 100 SVD took 438 seconds to construct and occupied 40.12 MB of memory. Given the advantage in construction time and memory used, the LMFR appears to be a viable alternative to the SVD for document retrieval.

4 Complexity of the LMFR

We now provide a complexity analysis of the cost of computing the LMFR. To make the explanation simpler, we will make the following assumptions: the data set represented by $\mathbf{M}$ 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 $m >> n$. These are not necessary conditions to construct an LMFR, but they do make the explanation clearer.

4.1 Cost of Obtaining the Section Representatives

The first step in computing the LMFR $\mathbf{C}_j\mathbf{Z}_j$ for a given section of data $\mathbf{M}_j$ is to obtain the section representatives which comprise $\mathbf{C}_j$. These are found via a clustering of $\mathbf{M}_j$. We assume that PDDP will be used to obtain the clustering. To simplify the analysis, we assume that we will create a perfectly balanced binary tree. This means that all of the leaf clusters in a given "level" will have the same cardinality, and that all of the clusters on a given level will be split before any clusters on the next level.

The majority of the cost of computing the PDDP clustering is the cost of computing the principal direction of the data in the current cluster being split. The principal direction is determined by the rank 1 SVD of the cluster. The rank 1 SVD is computed using the iterative procedure developed by Lanczos. The majority of the cost in finding the rank 1 SVD is computing a matrix-vector product of the form $\mathbf{M}_j\mathbf{v}$ twice each iteration

PDDP starts with the root cluster, which is all of the data being clustered. In this case, the root cluster is the $n \times k_d$ matrix $\mathbf{M}_j$, where n is the number

of attributes and k_d is the number of data items in the section. Computing the product of one row in $\mathbf{M}_j$ with a right vector $\mathbf{v}$ takes k_d multiplications and additions. There are n rows in $\mathbf{M}_j$. Therefore, the cost of computing a single matrix-vector product is

$$\gamma_1 n k_d, \tag{8}$$

where γ_1 is the fill fraction of $\mathbf{M}_j$. If the data in $\mathbf{M}$ are dense, $\gamma_1 = 1$. The overall cost of determining the principal direction of the root cluster $\mathbf{M}_j$ is

$$c_1 \gamma_1 n k_d, \tag{9}$$

where c_1 is a constant encapsulating the number of matrix-vector products computed before convergence.

After splitting the root cluster, we have two leaf clusters. Due to our assumption that we are creating a perfectly balanced binary tree, each of the two current leaf clusters contains the same number of data items, and the next two clusters chosen to be split will be the two current leaf clusters. Therefore, the cost of splitting the next two clusters is

$$2c_1 \gamma_1 n \left(\frac{k_d}{2} \right) = c_1 \gamma_1 n k_d, \tag{10}$$

which is the same as the cost of computing the splitting of the root cluster. The PDDP tree now contains four leaf clusters. The cost of splitting these four leaf clusters is

$$4c_1 \gamma_1 n \left(\frac{k_d}{4} \right) = c_1 \gamma_1 n k_d. \tag{11}$$

This result and the previous result indicate that the cost of computing a given level in the binary tree is the same for all levels. Every new level created in the perfectly balanced binary tree increases the number of leaf clusters by a power of 2. This progression is shown in Figure 3.

The cost of obtaining the k_c section representatives for the section $\mathbf{M}_j$ is

$$c_1 \gamma_1 n k_d \log_2 (k_c), \tag{12}$$

assuming that the number of section representatives k_c is an integer power of 2. If we have a total of k_s sections with k_d data points per section, and we obtain the same number of section representatives k_c for each section, the total cost of obtaining all section representatives for the entire data set will be

$$cost_{\mathsf{C}} = c_1 \gamma_1 n k_s k_d \log_2 (k_c) = c_1 \gamma_1 n m \log_2 (k_c). \tag{13}$$

For clarity, we reproduce all of the assumptions involved in the formulation, as well as the final result for the cost of computing the section representatives, in Figure 4.

Number of clusters	Cost
2	$c_1\gamma_1 n k_d$
4	$c_1\gamma_1 n k_d + 2c_1\gamma_1 n\left(\frac{k_d}{2}\right)$
	$= 2c_1\gamma_1 n k_d$
8	$c_1\gamma_1 n k_d + 2c_1\gamma_1 n\left(\frac{k_d}{2}\right) + 4c_1\gamma_1 n\left(\frac{k_d}{4}\right)$
	$= 3c_1\gamma_1 n k_d$
16	$c_1\gamma_1 n k_d + 2c_1\gamma_1 n\left(\frac{k_d}{2}\right) + 4c_1\gamma_1 n\left(\frac{k_d}{4}\right) + 8c_1\gamma_1 n\left(\frac{k_d}{8}\right)$
	$= 4c_1\gamma_1 n k_d$
k_c	$c_1\gamma_1 n k_d \log_2(k_c)$

Fig. 3. Complexity for the PDDP tree computation, shown for the number of clusters computed for a given section of data. The value of k_c is assumed to be an integer power of 2, and the tree is assumed to be perfectly balanced.

Assumptions:
1. PDDP is used to obtain the section representatives
2. a perfectly balanced binary tree is created
3. each section has the same value of k_d and k_c
4. $k_d k_s = m$
5. k_c is an integer power of two

Result: Cost of obtaining $\mathbf{C}$ is

$$cost_{\mathbf{C}} = c_1\gamma_1 nm \log_2(k_c)$$

Fig. 4. Summary of the cost of obtaining $\mathbf{C}$.

4.2 Computing the Data Loadings

Computing the data loadings in $\mathbf{Z}_j$ is a multi-step process. To find the least-squares approximation to a given data item $\mathbf{x}_i$ in $\mathbf{M}_j$, it is necessary to find the distance from $\mathbf{x}_i$ to every section representative $\mathbf{c}_l$ in $\mathbf{C}_j$, select the k_z section representatives $\mathbf{c}_l$ that are closest to $\mathbf{x}_i$, and compute the least-squares approximation using the normal equations.

Computing the distance from $\mathbf{x}_i$ to a single representative in $\mathbf{C}_j$ requires $\gamma_1 n$ multiplications and subtractions. Since there are k_c representatives in $\mathbf{C}_j$, the total cost of computing the distances for one data item $\mathbf{x}_i$ is $\gamma_1 n k_c$. We assume that the number of representatives k_z used to approximate x_i is very small, so that it will be less expensive to directly select the k_z closest representatives, rather than sorting the distances first. Therefore, it takes $k_c k_z$ searches through the representatives to find the k_z closest representatives, which are used to form the $n \times k_z$ matrix $\mathbf{C}_i$ as in §3.1.

The final step is to compute the least-squares approximation for each data item using the k_z centroids obtained in the previous step. The normal equations are used to obtain the least-squares approximation. The cost of computing a least-squares approximation for the $n \times k_z$ system is:

$$\gamma_1 n k_z^2 + \frac{1}{3} k_z^3,$$

if we ignore lower order terms. The total combined cost for obtaining the loadings for one data item $\mathbf{x}_i$ is

$$\gamma_1 n k_c + k_c k_z + \gamma_1 n k_z^2 + \frac{1}{3} k_z^3, \tag{14}$$

and the cost of obtaining all data loadings for all sections is

$$cost_{\mathbf{Z}} = m \left(\gamma_1 n k_c + k_c k_z + \gamma_1 n k_z^2 + \frac{1}{3} k_z^3 \right). \tag{15}$$

The assumptions and final cost for computing the data loadings which comprise the $\mathbf{Z}$ matrix are shown in Figure 5.

Assumptions:
1. $k_z \ll k_c$, so direct search for k_z closest representatives in $\mathbf{C}$ is less expensive than sorting
2. same value for k_z is used for all sections
3. normal equations are used to obtain least-squares
4. additional lower-order terms from least squares are ignored

Result: Cost of obtaining $\mathbf{Z}$ is

$$cost_{\mathbf{Z}} = m \left(\gamma_1 n k_c + k_c k_z + \gamma_1 n k_z^2 + \tfrac{1}{3} k_z^3 \right)$$

Fig. 5. Summary of the cost of obtaining $\mathbf{Z}$.

5 Clustering Large Data Sets Using the LMFR

Now that we have an LMFR of the entire data set, we can replace the original representation of the data with the LMFR to obtain a clustering using PDDP. We call the extension of PDDP to large data sets Piecemeal PDDP (PMPDDP). The piecemeal part of the name is from the fact that the LMFR is constructed in a piecemeal fashion, one section at a time, and from the fact that PDDP is used to compute the intermediate clusterings used in the construction of the LMFR.

The PMPDDP clustering method is straightforward. The process is to first construct the LMFR of the data, and then cluster the LMFR using PDDP.

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 (see Table 1) and k_f, the number of final clusters computed
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 LMFR (cf. Fig. 1) for the section $\mathbf{M}_j$ using PDDP to compute $\mathbf{C}_j$ (cf. §3.2)
4.. **Assemble** the matrices $\mathbf{C}$ and $\mathbf{Z}$ as in (7) in the text, using all the matrices $\mathbf{C}_j$ and $\mathbf{Z}_j$ from all passes through steps 2-3
5.. **Compute** the PDDP tree with k_f clusters for the entire system $\mathbf{CZ}$.
6. **Result:** A binary tree with k_f leaf nodes forming a partitioning of the entire data set.

Fig. 6. PMPDDP algorithm.

The PMPDDP algorithm is shown in Figure 6. An earlier version of PMPDDP appeared in [17].

PDDP is useful for producing the section representatives in $\mathbf{C}$ because it is fast and scalable. Since we are only interested in finding suitable representatives, we do not require the optimal clustering of the data in a section, just an inexpensive one. More expensive clustering algorithms will probably not significantly alter the results, though of course the values in the factorization would change. However, we could replace PDDP with any other suitable clustering algorithm without difficulty, since when we compute the section representatives we are dealing with a piece of the original data that will fit into memory. K-means, especially bisecting k-means [20], for example, would be candidate methods to replace PDDP at this stage.

However, when clustering the factored form, the situation is different. Any clustering algorithm which uses a similarity measure, such as the aforementioned k-means method, would require that the data be reconstructed each time a similarity measure was needed. Reconstructing the entire data set at once requires at least as much memory as the original data set, defeating the purpose of the LMFR. Reconstructed sparse data will take up more space than the original data, since the section representatives will be denser than the original data items. The LMFR only saves memory as long as it remains in factored form. Naturally, small blocks of the original data could be reconstructed on the fly every time a similarity measure is required, but that could add considerable additional expense.

PDDP does not use a similarity measure when determining the clustering. Instead, PDDP uses the principal direction of the data in a cluster to determine how to split that cluster. The principal direction is computed using an iterative procedure developed by Lanczos which computes products of the form:

$$\left(\mathbf{M} - \mathbf{w}\mathbf{e}^{\mathsf{T}}\right)\mathbf{v} = \mathbf{M}\mathbf{v} - \mathbf{w}\left(\mathbf{e}^{\mathsf{T}}\mathbf{v}\right), \text{ where } \mathbf{w} = \frac{1}{m}\left(\mathbf{M}\mathbf{e}^{\mathsf{T}}\right), \tag{16}$$

where $\mathbf{v}$ is some vector. We can replace $\mathbf{M}$ by the factored form $\mathbf{CZ}$, group the products accordingly, and compute:

$$\mathbf{C}\left(\mathbf{Z}\mathbf{v}\right) - \hat{\mathbf{w}}\left(\mathbf{e}^{\mathrm{T}}\mathbf{v}\right), \text{ where } \hat{\mathbf{w}} = \frac{1}{m}\mathbf{C}\left(\mathbf{Z}\mathbf{e}^{\mathrm{T}}\right), \tag{17}$$

and in doing so we never explicitly reconstruct the data. Therefore, the LMFR is well-suited to being clustered using the PDDP method.

5.1 Scatter Computation

There is one other aspect of PMPDDP to consider. PDDP normally chooses the next cluster to split based on the scatter values of the leaf clusters. Computing the scatter when clustering the LMFR $\mathbf{CZ}$ would require that the data be reconstructed. For a scatter computation, this could be done in a block-wise fashion without too much difficulty. However, we wish to have a method which does not require reconstruction of the data.

Instead of reconstructing the data to directly compute the scatter, we estimate the scatter. If we could compute all of the singular values σ_i, we could compute the exact scatter s as

$$s = \sigma_1^2 + \sigma_2^2 + \ldots + \sigma_n^2. \tag{18}$$

This formula can be re-written as

$$s = \sigma_1^2\left(1 + \frac{\sigma_2^2}{\sigma_1^2} + \frac{\sigma_3^2}{\sigma_1^2} + \ldots + \frac{\sigma_n^2}{\sigma_1^2}\right). \tag{19}$$

Now, we can use the two leading singular values to estimate the scatter as

$$s \approx \sigma_1^2\left(1 + \frac{\sigma_2^2}{\sigma_1^2} + \left(\frac{\sigma_2^2}{\sigma_1^2}\right)^2 + \ldots + \left(\frac{\sigma_2^2}{\sigma_1^2}\right)^{n-1}\right) = \sigma_1^2\left(\frac{1 - \left(\frac{\sigma_2^2}{\sigma_1^2}\right)^n}{1 - \frac{\sigma_2^2}{\sigma_1^2}}\right), \tag{20}$$

where σ_1 is the leading singular value of the cluster and σ_2 is the next singular value. The estimate assumes that the singular values are decreasing geometrically, which from empirical observation seems to be a reasonable assumption. Note that if we computed all of the singular values, we could compute the exact scatter. However, computing all or even a significant number of the singular values would be prohibitively expensive. A high degree of accuracy is not necessary, since this scatter computation is only used to determine which cluster to split next. The estimate needs only to be consistent with the data being clustered. Presumably, if the estimate is either too low or too high, the same type of estimation error will exist for all clusters.

The leading singular value is associated with the principal direction, and an estimate of the second singular value is available without much additional cost. Estimating the scatter requires that the principal direction of all leaf clusters needs to be computed, whether they are split or not. We could choose another splitting criteria, but from previous results with PDDP on various data sets, scatter seems to be a very good way to select the clusters to split.

6 Complexity of a PMPDDP clustering

In the following we develop some formulas for the cost of a general PMPDDP clustering. We will use the same assumptions as we did for the analysis used to get the section representatives comprising $\mathbf{C}$ (cf. §4.1). We assume we will produce a completely balanced binary tree with a number of leaves being an integer power of 2, and that the cost of clustering is basically the cost of obtaining the principal directions which determine how each cluster is split.

Replacing the original matrix $\mathbf{M}$ with the approximation $\mathbf{CZ}$ in the PDDP method changes the calculation in the splitting process from a matrix-vector product to a matrix-matrix-vector product. This product can be written as $\mathbf{C}(\mathbf{Zv})$, where $\mathbf{v}$ is a "generic" $m \times 1$ vector, $\mathbf{C}$ is a $n \times k_s k_c$ matrix and $\mathbf{Z}$ is a $k_s k_c \times m$ matrix. Note that we group the product such that the matrix-vector product is computed before multiplying by the other matrix. We must avoid explicitly forming the product $\mathbf{CZ}$, since the result of that product will not fit into memory. $\mathbf{Z}$ is a sparse matrix with k_z non-zeroes per column, and therefore 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 $\mathbf{v}$. We show all the parameters involved in a PMPDDP clustering in Table 2.

Table 2. Definition of the parameters used in PMPDDP.

parameter	description
m	total number of data items
n	number of attributes per data item
γ_1	fill fraction of the attributes in $\mathbf{M}$
γ_2	fill fraction of the attributes in $\mathbf{C}$
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
k_f	number of final clusters

Again, we start the analysis with the root cluster. The cost of computing the principal direction of the root cluster is

$$c_2(\gamma_2 n k_s k_c + m k_z), \tag{21}$$

where γ_2 is the fill fraction of $\mathbf{C}$ and c_2 is a constant encapsulating the number of matrix-matrix-vector products required to convergence. The mk_z portion of the formula is the contribution from forming the product $\mathbf{Zv}$, where $\mathbf{v}$ is some vector, and nk_sk_c is the cost of multiplying $\mathbf{C}$ by the resultant of the product $\mathbf{Zv}$.

At this point, we have two leaf clusters. One might be tempted to assume that the expense would follow the same relationship as in regular PDDP, and

that the cost of splitting these two clusters is the same as the cost of splitting the root cluster, but that is incorrect. The reason for the difference is that while the cost of forming the product **Zv** decreases with decreasing cluster size, the cost of multiplying **C** by the resultant of the product **Zv** does not decrease with decreasing cluster size. As a result, the cost of splitting these two clusters is

$$2c_2\gamma_2 nk_s k_c + 2c_2\left(\frac{m}{2}\right)k_z = 2c_2\gamma_2 nk_s k_c + c_2 mk_z. \tag{22}$$

It might be possible to reduce the computational expense associated with **C** by only considering the columns of **C** which actually participate in the product when splitting the leaf clusters. However, there does not appear to be an inexpensive way to determine which columns in **C** would be required at each step, so we leave the method as stated and accept the expense.

Following the pattern to its conclusion, as shown in Figure 7, we have the result for the cost of clustering **CZ**,

$$c_2\gamma_2 nk_s k_c(k_f - 1) + c_2 mk_z \log_2(k_f). \tag{23}$$

Number of clusters	Cost
2	$c_2\gamma_2 nk_s k_c + c_2 mk_z$
4	$c_2\gamma_2 nk_s k_c + c_2 mk_z + 2c_2\gamma_2 nk_s k_c + 2c_2\left(\frac{m}{2}\right)k_z$
	$= \; 3c_2\gamma_2 nk_s k_c + 2c_2 k_z m$
8	$c_2\gamma_2 nk_s k_c + c_2 mk_z + 2c_2\gamma_2 nk_s k_c + 2c_2\left(\frac{m}{2}\right)k_z$
	$+ \; 4c_2\gamma_2 nk_s k_c + 4c_2\left(\frac{m}{4}\right)k_z$
	$= \; 7c_2\gamma_2 nk_s k_c + 3c_2 mk_z$
16	$c_2\gamma_2 nk_s k_c + c_2 mk_z + 2c_2\gamma_2 nk_s k_c + 2c_2\left(\frac{m}{2}\right)k_z$
	$+ \; 4c_2\gamma_2 nk_s k_c + 4c_2\left(\frac{m}{4}\right)k_z$
	$+ \; 8c_2\gamma_2 nk_s k_c + 8c_2\left(\frac{m}{8}\right)k_z$
	$= \; 15c_2\gamma_2 nk_s k_c + 4c_2 mk_z$
k_f	$c_2\gamma_2 nk_s k_c(k_f - 1) + c_2 mk_z \log_2(k_f)$

Fig. 7. Complexity for the PMPDDP tree computation, shown for the number of clusters computed. The additional expense of computing the estimated scatter is not considered.

We have not yet considered the fact that PMPDDP uses the estimated scatter to determine which cluster is split next. To obtain the estimated scat-

ter, it is necessary to compute the principal direction of all the leaf clusters before we split them. We effectively incur the expense of computing an additional level in the PDDP tree, which doubles the number of splits computed. Therefore, the actual cost of computing a PDDP clustering of **CZ** when using the estimated scatter is

$$cost_{\texttt{clusterCZ}} = c_2\gamma_2 n k_s k_c(2k_f - 1) + c_2 m k_z \log_2(2k_f). \tag{24}$$

For clarity, we reproduce all of the costs of computing a PMPDDP clustering in Table 3, and all of the assumptions used to write the formulas in Figure 8. Note that the costs are higher than computing a PDDP clustering. This is expected since we already incur more cost than a PDDP clustering just by obtaining the section representatives which comprise **C**, assuming $k_c > k_f$.

Table 3. Collected costs of PMPDDP, including the costs of obtaining the LMFR. See Table 2 for a definition of the parameters, and Figure 8 for the assumptions made when writing the formulas.

operation	amortized cost
clustering sections to obtain **C**	$c_1\gamma_1 nm \log_2(k_c)$
find distance from data points to centroids	$\gamma_2 nmk_c$
find k_z closest centroids	mk_ck_z
compute best least-squares approx	$m(\gamma_2 nk_z^2 + \frac{1}{3}k_z^3)$
cluster the representation **CZ** using PDDP	$c_2\gamma_2 nk_sk_c(2k_f - 1) + c_2 mk_z log_2(2k_f)$
Compare cost of PDDP	$c_1\gamma_1 nm \log_2(k_f)$

6.1 Complexity for One Varying Parameter

In this section, we produce the PMPDDP complexity formulas for the case in which we vary then number of representatives k_z used to approximate each data item and the number of representatives k_c produced for each section of data, while leaving all other parameters fixed. We also produce formulas for the cost of PMPDDP with respect to the number of data items m and the number of attributes n.

Before we proceed further, we collect the results from the formulas in (13, 15, 24) and write the total cost of PMPDDP as:

$$\begin{aligned} m\ (&c_1\gamma_1 n \log_2(k_c) + \gamma_2 nk_c + k_ck_z + \gamma_2 nk_z^2 \\ &+ \tfrac{1}{3}k_z^3 + c_2\gamma_2 \tfrac{n}{m} k_s k_c(2k_f - 1) + c_2 k_z \log_2(2k_f)). \end{aligned} \tag{25}$$

We will use this result when computing the formulas for each instance.

Obtaining C:
1. PDDP is used to obtain the section representatives
2. a perfectly balanced binary tree is created
3. each section has the same value of k_d and k_c
4. $k_d k_s = m$
5. k_c is an integer power of two

Obtaining Z:
6. $k_z \ll k_c$, so direct search for k_z closest representatives in $\mathbf{C}$ is less expensive than sorting
7. same value for k_z is used for all sections
8. normal equations are used to obtain least-squares
9. additional $O(k_z^2)$ term from least squares is ignored

Clustering CZ:
11. PDDP is used to obtain the clustering
12. a perfectly balanced binary tree is created
13. scatter is estimated by pre-computing the splits for all leaves
14. k_f is an integer power of 2

Fig. 8. Summary of the assumptions used to obtain the formulas in Table 3.

Varying k_z

In this section, we show the cost of PMPDDP when all parameters except k_z are fixed. This will demonstrate the effect on the overall cost of PMPDDP when changing the number of representatives used to approximate each data item. Since k_z is independent from all other parameters, it is possible to fix the remaining parameters to constant values.

We start by examining (25) and extracting only those components which depend on k_z. Note that while the other components may contribute a very high cost, that cost will be fixed. The resulting formula is

$$m\left(\left(k_c + c_2 \log_2\left(2k_f\right)\right) k_z + \gamma_2 n k_z^2 + \frac{1}{3}k_z^3\right). \tag{26}$$

This formula indicates that there may be a strong linear component in k_z when the quantity $\gamma_2 n k_z^2$ is relatively small, as would probably be the case for relatively low-dimension dense data sets. In the general case, since we expect k_z to be small, the square term will dominate the costs. With k_z sufficiently large, the cost will grow cubicly.

Increasing k_z is expensive from a memory standpoint, since each column of $\mathbf{Z}$ has k_z non-zero entries. Keeping k_z small controls both the computation cost and memory footprint of the LMFR.

Varying k_c

The other PMPDDP parameter we will consider is the number of representatives per section k_c. We demonstrated in [16] that k_c is important to clustering

accuracy, so it is useful to know the trade-off in cost. As before, if we only consider the elements of the formula (25) that involve k_c, we have the result

$$m\left(c_1\gamma_1 n\log_2(k_c)+\gamma_2 nk_c+k_ck_z+c_2\gamma_2\frac{n}{m}k_sk_c(2k_f-1)\right).$$

If we factor out the k_c term, we have the result

$$m\left(c_1\gamma_1 n\log_2(k_c)+k_c\left(\gamma_2 n+k_z+c_2\gamma_2\frac{n}{m}k_s(2k_f-1)\right)\right). \tag{27}$$

We expect that the cost of PMPDDP will increase slightly more than linearly with k_c due to the logarithmic term.

Varying n

We now consider the contribution to the cost from the number of attributes n. Taking all of the terms from (25) which involve n, we have

$$m\left(c_1\gamma_1 n\log_2(k_c)+\gamma_2 nk_c+\gamma_2 nk_z^2+c_2\gamma_2\frac{n}{m}k_sk_c(2k_f-1)\right).$$

We can factor n from this formula with the resulting cost being

$$nm\left(c_1\gamma_1\log_2(k_c)+\gamma_2 k_c+\gamma_2 k_z^2+c_2\gamma_2\frac{1}{m}k_sk_c(2k_f-1)\right). \tag{28}$$

From this result, we expect the cost of PMPDDP to be linear in the number of attributes.

Varying m

The final result we will consider is the cost in terms of the number of data items m. Note that in (25), all terms inside the outermost parenthesis are dependent on m except

$$c_2\gamma_2\frac{n}{m}k_sk_c(2k_f-1),$$

since this term will be multiplied by m. With this consideration, and with all values fixed except m, and rather than re-writing the entire formula, we can recast (25) as

$$c_3m+c_4, \tag{29}$$

where c_3 and c_4 encompass the appropriate parameters in (25). From this result we can expect that PMPDDP is linear in the number of data items m.

7 Experimental Results

In this section we show some experimental results for the PMPDDP clustering method for both a dense and a sparse large data set. We sub-sampled each data set so that we could directly compare the results of a PDDP clustering with a PMPDDP clustering. We compare the quality of the two clusterings using scatter and entropy as measures.

Recall from §5.1 that PMPDDP uses an estimated scatter value to determine which cluster is split next. To determine the effect on clustering quality of using the estimated scatter, we include results for clustering quality using the computed scatter. Computing the scatter required that the data be reconstructed. To minimize the amount of additional memory, we reconstructed 50 data points at a time. When we used the computed scatter to select which cluster to split next, we did not pre-compute the splits of the leaf clusters.

The algorithms were implemented in MATLAB and the experiments were performed on a AMD XP2200+ computer with 1 GB of memory and 1.5 GB of swap space.

7.1 Data Sets

We used two data sets to evaluate the method, one dense and one sparse. The dense data set was the KDD Cup 1998 data set [2], which consists of network connection data. Since the data set was designed to test classification algorithms, it was labeled. Connection types were either "normal" or some kind of attack. We combined both the training and test data into one large data set. Categorical attributes were converted to binary attributes as needed. Each attribute was scaled to have a mean of zero and a variance of one. Post-processing, the entire data set occupied over 4 GB of memory.

The sparse data set was downloaded from the web. Topics were selected so that a google search on a given topic would return at least 200 hits. We assumed that the top 200 documents returned were relevant to the search topic. Each web page was treated as a document. Any word that only appeared in one document was removed from the dictionary, as were all stop words. The words were also stemmed using Porter's stemming algorithm. Each document vector was scaled to unit length.

The data sets are summarized in Table 4. A more in-depth description of how the data were processed is available in [16].

7.2 Performance Measures

We used two different performance measures to evaluate the comparative quality of the clustering. Those measures were scatter and entropy.

The scatter $s_{\mathtt{C}}$ of a cluster $\mathbf{M}_{\mathtt{C}}$ is defined as:

$$s_{\mathtt{C}} \stackrel{\text{def}}{=} \sum_{j \in \mathtt{C}} (\mathbf{x}_j - \mathbf{w}_{\mathtt{C}})^2 = \|\mathbf{M}_{\mathtt{C}} - \mathbf{w}_{\mathtt{C}} \mathbf{e}^T\|_F^2, \tag{30}$$

Table 4. Summary of the Data Sets Used in the Experiments. The KDD data set is the KDD Cup 1998 data set from the UCI:KDD machine learning repository [2], and the web data was produced at the University of Minnesota.

data set	KDD	web
number of samples m	4898431	25508
number of attributes n	122	733449
number of categories	23	1733

where $\mathbf{w_c}$ is the mean of the cluster, $\mathbf{e}$ is the m-dimensional vector $[1\ 1\ \ldots\ 1]^T$ and $\|\ \|_F$ is the Frobenius norm. For some $n \times m$ matrix $\mathbf{A}$, the Frobenius norm of $\mathbf{A}$ is

$$\|A\|_F = \sqrt{\sum_{\substack{1 \le i \le n \\ 1 \le j \le m}} a_{i,j}^2}, \tag{31}$$

where $a_{i,j}$ is the entry in the $i^{\mathtt{th}}$ row and $j^{\mathtt{th}}$ column of $\mathbf{A}$. A low scatter value indicates good cluster quality. Since scatter is a relative performance measure, it only makes sense to use the scatter to compare clusterings having the same cardinality.

The entropy e_j of cluster j is defined by:

$$e_j \stackrel{\text{def}}{=} -\sum_i \left(\frac{c(i,j)}{\sum_i c(i,j)} \right) \cdot \log \left(\frac{c(i,j)}{\sum_i c(i,j)} \right), \tag{32}$$

where $c(i,j)$ is the number of times label i occurs 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 is positive. The total entropy for a given clustering is the weighted average of the cluster entropies:

$$e_{total} \stackrel{\text{def}}{=} \frac{1}{m} \sum_i e_i \cdot k_i. \tag{33}$$

The lower the entropy, the better the quality. As with the scatter, entropy is a relative performance measure, so the same caveats apply.

7.3 KDD Results

The KDD intrusion detection data were divided into 25 random samples. It was only possible to compute a PDDP clustering of the data up to a combination of the first 5 samples of data. After that, the amount of memory required for data and overhead exceeded the capacity of the computer.

The parameters and results for the KDD intrusion detection data set are summarized in Table 5. We show the results for the combinations through the first five pieces of data and the results for the entire data set. The clustering

quality from PMPDDP is comparable to PDDP in both the scatter and entropy performance measures. The memory savings are significant. The costs are higher for PMPDDP, but the majority of the time is spent computing the factored representation of the data. Once the factored representation is available, clusterings of different sizes can be computed relatively inexpensively.

Table 5. Comparison of a PDDP clustering with a PMPDDP clustering of the KDD data for various sample sizes. It was not possible to compute a PDDP clustering past the largest sample size shown since the data would not fit into memory. It was not possible to compute the scatter for PMPDDP for the entire data set, since it wouldn't fit into memory. Also including are results for a modified PMPDDP clustering method which uses the computed scatter (c.s.) rather than using the estimated scatter. See Table 2 for a definition of the parameters, and §7.1 for a description of the data.

data set	KDD					
m	195937	391874	587811	783748	979685	4898431
k_s	5	10	15	20	25	125
k_c	392	392	392	392	392	392
k_z	3	3	3	3	3	3
k_f	36	36	36	36	36	36
Normalized Scatter Values, lower is better						
PDDP	3.179e-04	3.279e-04	3.276e-04	3.290e-04	3.288e-04	won't fit
PMPDDP	3.257e-04	3.236e-04	3.271e-04	3.250e-04	3.275e-04	N.A.
PMPDDP c.s.	3.271e-04	3.258e-04	3.250e-04	3.245e-04	3.255e-04	N.A.
Entropy Values, lower is better						
PDDP	.127	.130	.129	.124	.120	won't fit
PMPDDP	.0590	.0585	.0546	.120	.114	.113
PMPDDP c.s.	.125	.127	.126	.112	.120	.113
Time taken by experiments, in seconds, on XP 2200+						
PDDP	39.72	89.68	140.45	204.87	282.44	won't fit
Compute **CZ**	216.66	450.41	674.49	872.15	1108.56	5652.59
Cluster **CZ**	15.63	32.71	52.67	69.07	89.62	492.81
Cluster **CZ** c.s.	20.20	45.18	73.59	96.42	120.69	447.36
PMPDDP totals	232.29	483.12	727.16	941.22	1198.18	6145.40
Memory occupied by representation, in MB						
M	191.2	382.4	573.6	764.9	956.2	**4780**
CZ	8.24	16.48	24.72	39.00	48.75	**206**

Examining the results for the PMPDDP clustering using the computed scatter as compared to PMPDDP using the estimated scatter, we can see that the scatter values are similar for the two approaches. The entropy is better for the smaller sample sizes when using the estimated scatter, but this could be a results of some bias in the data set. A few labels are assigned to most

of the data points, while the remaining labels are assigned to relatively few data points. A small number of data points can move and change the entropy results significantly. In any case, once more data are present, the advantage in entropy values for the estimated scatter is no longer present. As such, this anomaly probably does not represent any significant advantage in general to using the estimated scatter. Also, we can see that the clustering times are close enough that there is no advantage in speed when using the estimated scatter for this dense data set.

The entire KDD intrusion detection data set in its original representation would occupy 4.78 GB of memory, beyond the limits of most desktop workstations. The factored representation of the data only requires about 206 MB of memory for the PMPDDP parameters selected, leaving plenty of memory space for clustering computation overhead on even a 512 MB workstation.

The time taken as the number of data items increased is shown in Figure 9. For this data set, PMPDDP costs scale linearly with the number of data items. This agrees with the complexity analysis in §6.1.

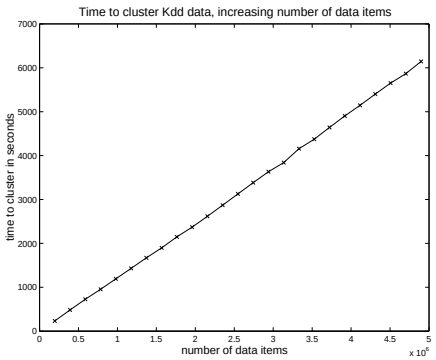

Fig. 9. Time taken for a PMPDDP clustering of the KDD data set with an increasing number of data items. The parameters used in the experiments are shown in Table 5.

7.4 Web

The web data were divided into 8 random samples. It was not possible to compute a PDDP clustering for more than a combination of 6 samples of the data, since after that point the program terminated abnormally due to lack of swap space.

The parameters and results are shown in Table 6, including the results for the entire data set. The PMPDDP and PDDP clusterings again have similar quality with respect to the scatter and entropy. Note that the memory savings

are not as significant as those for the dense data set. The sparse data already uses a representation which saves a considerable amount of memory. Plus, the **C** matrix is considerably more dense than the original representation of the data. This is due to the fact that the **C** matrix is comprised of cluster centroids. A centroid of any given cluster must contain a word entry for any data point in the cluster which has that word as an element. Therefore, the centroid of a cluster of sparse data is usually denser than any given item in the cluster. The higher density also accounts for some of the additional expense incurred during clustering, since the greater density is associated with an increase in the number of multiplications required to obtain the principal direction of the cluster.

Table 6. Comparison of a PDDP clustering and a PMPDDP clustering of the web data set for various sample sizes. The entire data set wouldn't fit into memory, so it wasn't possible to perform a PDDP clustering for the entire data set. Since the data wouldn't fit, it was not possible to compute the scatter of PMPDDP for the entire data set. Also including are results for a modified PMPDDP clustering method which uses the computer scatter (c.s.) rather than using the estimated scatter. See Table 2 for the parameter definitions and §7.1 for a description of the data.

data set	Web						
m	40688	81376	122064	162752	203440	244128	325508
k_s	5	10	15	20	25	30	40
k_c	81	81	81	81	81	81	81
k_z	3	3	3	3	3	3	3
k_f	200	200	200	200	200	200	200
Normalized Scatter Values, lower is better							
PDDP	.7738	.7760	.7767	.7778	.7789	.7787	won't fit
PMPDDP	.7762	.7792	.7802	.7809	.7819	.7829	N.A.
PMPDDP c.s.	.7758	.7788	.7796	.7800	.7812	.7817	.7820
Entropy Values, lower is better							
PDDP	5.043	5.483	5.692	5.836	5.922	5.990	won't fit
PMPDDP	5.168	5.624	5.843	6.004	6.094	6.175	6.869
PMPDDP c.s.	5.127	5.598	5.842	5.987	6.088	6.161	6.253
Time taken by experiments, in seconds, on an XP 2200+							
PDDP	1461	2527	3421	4359	5277	6286	won't fit
Compute **CZ**	5909	11783	17648	23508	29414	35288	47058
Cluster **CZ**	9174	17654	26278	34565	43591	51992	68416
Cluster **CZ** c.s.	15192	30208	44908	59634	74542	89614	119762
PMPDDP total	15083	29437	43926	58073	73005	87279	115475
Memory occupied by representation, in MB							
M	115.3	227.3	339.9	451.4	563.6	674.6	**897.3**
CZ	43.54	84.72	126.0	167.0	208.5	248.9	**330.3**

It was not possible to cluster the entire data set using the original representation of the data, which occupied 897 MB of memory, while the LMFR at 330 MB left sufficient memory space for clustering overhead.

The comparison between clustering quality and cost between standard PMPDDP and PMPDDP using the computer scatter is much more pronounced than for the dense data. Using the estimated scatter saves a significant amount of time during the clustering process, even though it requires computing twice the number of splits. The scatter values when clustering using the computed scatter are slightly better than those for the estimated scatter. The entropy values are better as well. However, the amount of time saved when using the estimated scatter is enough that we would still recommend using it over the computed scatter.

The time taken as the number of data items increased is shown in figure 10 (a). As with the KDD data set, the time taken to compute a complete PMPDDP clustering of the web data is linear in the number of data items. This agrees with the complexity analysis in §6.1.

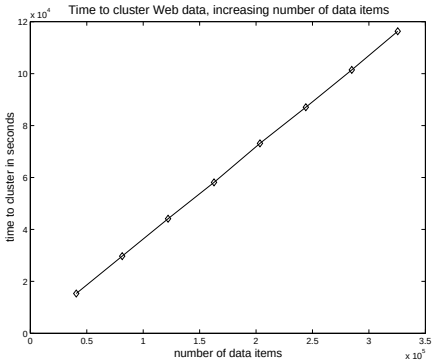

Fig. 10. Time taken for a PMPDDP clustering of the web data with an increasing number of data items. The results are a graphical representation of the times from Table 6.

8 How to Apply PMPDDP in Practice

We realize that while the experiments demonstrate that PMPDDP is a scalable extension of PDDP, they do not give much intuition on how to select the parameters. There are much more extensive experiments in [16], and using those results, we can provide some guidelines on how to apply PMPDDP in practice.

The LMFR construction is the point where the parameter choices are most relevant. Results indicate that even for an LMFR with the exact same memory

footprint, minimizing the number of sections is beneficial. In other words, it is best to process the original data in pieces which are as large as can be fit into memory while still allowing room for overhead and room for the new matrices being created. Realize that it is not necessary for the sections to have the same size, so if there is some remainder data, it can be used to construct a smaller section, with the remaining parameters adjusted accordingly.

There is a trade-off when selecting the values of k_c and k_z, since those are the two user-selectable parameters which control the size of the LMFR. The experimental results indicated that increasing k_z to numbers above 7 or 8 does not increase clustering quality, and values of 3 to 5 provide good results in most cases.

The single most important parameter is k_c, the number of representatives produced per section. The accuracy of the clustering, and the approximation accuracy of the LMFR, are both strongly influenced by the value of k_c. Therefore, k_c should be chosen such that it will maximize the memory footprint of the global LMFR.

Using the above information, we recommend starting with a small value for k_z and then selecting k_c to maximize memory use. If the data set in question is sparse, it would be advisable to test the parameter choice on a small piece of data to determine the increase in density so some estimate of the final memory footprint of the LMFR could be obtained.

Note that applying the above technique may produce an LMFR which is much larger than the size necessary to obtain good clustering quality. For instance, the experimental results in §7.3 were good with an LMFR that did not occupy as much memory as possible. However, it is difficult to know beforehand how much reduction a given data set can tolerate. The optimum memory reduction, if there is such a result, would be strongly data dependent. Therefore, it is difficult to do other than suggest making the LMFR as large and, correspondingly, as accurate a representation of the original data as available memory allows.

9 Summary

In this chapter we presented a method to extend the Principal Direction Divisive Partitioning (PDDP) clustering method to data sets which cannot fit into memory at once. To accomplish this, we construct a Low-Memory Factored Representation (LMFR) of the data. The LMFR transparently replaces the original representation of the data in the PDDP method. We call the combination of constructing an LMFR and clustering it using PDDP PieceMeal PDDP (PMPDDP).

The LMFR is comprised of two matrices. The LMFR is computed incrementally using relatively small pieces of the original data called sections. Each section of data is clustered, and the centroids of the clusters form the

first matrix $\mathbf{C}_j$. The centroids are then used to construct a least-squares approximation to each data point. The data loadings from the least-squares approximation are used to construct the second matrix $\mathbf{Z}_j$. Memory is saved since only a small number of centroids are used to approximate each data item, making $\mathbf{Z}$ very sparse. $\mathbf{Z}$ must be represented in sparse matrix format in order to realize the memory savings. Once a $\mathbf{C}_j$ and $\mathbf{Z}_j$ is available for each section, they are assembled into a global representation of all the data, $\mathbf{CZ}$. The matrices $\mathbf{CZ}$ can then be used in place of the original representation of the data. The product $\mathbf{CZ}$ is never computed explicitly, since the product would take up at least as much space as the original data. Unlike many other approximating techniques, the LMFR provides a unique representation of each data item.

We provided a complexity analysis of the cost of constructing the LMFR. This provides a useful guide when determining how to choose the parameters if the time of computation is critical. In the process, we showed that PDDP is theoretically linear in the number of data items and the number of attributes, which has been shown to be the case experimentally.

We then described the PMPDDP clustering algorithm. PMPDDP uses the LMFR in place of the original data to obtain a clustering of the data. Since each original data item has a corresponding column in the LMFR, mapping the clustering of the LMFR to the original data is trivial. Therefore, a clustering of the LMFR is a clustering of the original data.

PDDP is uniquely suited to clustering the LMFR since PDDP does not require similarity measures to determine the clustering. Therefore, no data need to be reconstructed and no full or even partial products of $\mathbf{CZ}$ are computed. To avoid reconstructing the data, an estimate of the scatter is used in place of the computed scatter when determining which cluster in the PDDP tree is split next.

With the complexity analysis, we showed PMPDDP is linear in the number of data items and the number of attributes. Thus, PMPDDP extends PDDP to large data sets while remaining scalable.

Next, we provided some experimental results. The experiments demonstrated that it is possible to produce a PMPDDP clustering which has quality comparable to a PDDP clustering while saving a significant amount of memory. Therefore, it is possible to cluster much larger data sets than would otherwise be possible using the standard PDDP method. Additional data sets were shown to be clustered successfully using PMPDDP in [16].

We also showed the effect of replacing the estimated scatter, as used in PMPDDP, with the computed scatter, when determining which cluster to split next. For the dense data set, the difference was neutral with respect to clustering quality and clustering time. However, for the sparse data set, a significant amount of time can be saved during clustering by using the estimated scatter. Clustering quality was slightly inferior, but the reductions in clustering times more than made up for the differences in clustering quality.

Finally, we described how we would expect PMPDDP would be applied by a casual user. While we cannot guarantee that our suggestions provide an optimal balance between memory used and clustering quality, we believe that using our suggestions would provide the best clustering quality obtainable considering the amount of memory available on the workstation being used.

10 Acknowledgment

This work was partially supported by NSF grant 0208621.

References

1. K. Alsabti, S. Ranka, and V. Singh. An efficient k-means clustering algorithm. In *Proceedings of IPPS/SPDP Workshop on High Performance Data Mining*, 1998.
2. C. Blake, E. Keogh, and C. J. Merz. UCI repository of machine learning databases. *http://www.ics.uci.edu/∼mlearn/MLRepository.html*, 1998.
3. D. L. Boley. Principal direction divisive partitioning. *Data Mining and Knowledge Discovery*, 2(4):325–344, 1998.
4. P. Bradley and U. Fayyad. Refining initial points for k-means clustering. In J. Shavlik, editor, *Proceedings of the Fifteenth International Conference on Machine Learning (ICML), San Francisco, CA*, pages 91–99. AAAI Press, 1998.
5. P. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Proceedings Fourth International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1998.
6. 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 *ACM SIGIR*, 1992.
7. I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1):143–175, January 2001. Also appears as IBM Research Report RJ 10147, July 1999.
8. Pedro Domingos and Geoff Hulten. A general method for scaling up machine learning algorithms and its application to clustering. In *Proceedings of the 18th International Converence on Machine Learning*, pages 106–113. Morgan Kaufmann, 2001.
9. R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.
10. Chris Fraley. Algorithms for model-based Gaussian hierarchical clustering. *SIAM Journal on Scientific Computing*, 20(1):270–281, 1999.
11. Earl Gose, Richard Johnsonbaugh, and Steve Jost. *Pattern Recognition and Image Analysis*. Prentice Hall, 1996.
12. 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.
13. Hillol Kargupta, Weiyun Huang, Krishnamoorthy Sivakumar, and Erik L. Johnson. Distributed clustering using collective principal component analysis. *Knowledge and Information Systems*, 3(4):422–448, 2001.

14. G. Karypis, E.-H. Han, and V. Kumar. Multilevel refinement for hierarchical clustering. Technical Report 99-020, 1999.
15. T. Kurita. An efficient agglomerative clustering algorithm using a heap. *Pattern Recognition*, 24(3):205–209, 1991.
16. David Littau. *Using a Low-Memory Factored Representation to Data Mine Large Data Sets.* PhD Dissertation, University of Minnesota, Department of Computer Science, 2005.
17. David Littau and Daniel Boley. Using low-memory representations to cluster very large data sets. In D. Barbará and C. Kamath, editors, *Proceedings of the Third SIAM International Conference on Data Mining*, pages 341–345, 2003.
18. David Littau and Daniel Boley. Streaming data reduction using low-memory factored representations. *Journal of Information Sciences, Special Issue on Mining Stream Data*, to appear.
19. D. Pelleg and A. Moore. Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the 5th ACM SIGKDD*, pages 277–281, San Diego, CA, USA, 1999.
20. M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *6th ACM SIGKDD, World Text Mining Conference*, Boston, MA, USA, 2000.
21. T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, Montreal, Canada, 1996.