Single Pass Fuzzy C Means

Prodip Hore, Lawrence O. Hall, and Dmitry B. Goldgof

***Abstract*— Recently several algorithms for clustering large data sets or streaming data sets have been proposed. Most of them address the crisp case of clustering, which cannot be easily generalized to the fuzzy case. In this paper, we propose a simple single pass (through the data) fuzzy c means algorithm that neither uses any complicated data structure nor any complicated data compression techniques, yet produces data partitions comparable to fuzzy c means. We also show our simple single pass fuzzy c means clustering algorithm when compared to fuzzy c means produces excellent speed-ups in clustering and thus can be used even if the data can be fully loaded in memory. Experimental results using five real data sets are provided.**

I. Introduction

Recently various algorithms for clustering large data sets and streaming data sets have been proposed [2], [4], [5], [6], [7], [8], [12], [13]. The focus has been primarily either on sampling [2], [7], [8], [10], [22] or incrementally loading partial data, as much as can fit into memory at one time. The incremental approach [5], [6], [12], [13] generally keeps sufficient statistics or past knowledge of clusters from a previous run of a clustering algorithm in some data structures and uses them in improving the model for the future. Various algorithms [1], [3], [9], [15], [19] for speeding up clustering have also been proposed. While many algorithms have been proposed for large and very large data sets for the crisp case, not as much work has been done for the fuzzy case. As pointed out in [10], the crisp case may not be easily generalized for fuzzy clustering. This is due to the fact that in fuzzy methods an example does not belong to a cluster completely but has partial membership values in most clusters. More about clustering algorithms can be found in [23].

Clustering large amounts of data takes a long time. Further, new unlabeled data sets which will not fit in memory are becoming available. To cluster them, either sub sampling is required to fit the data in memory or the time will be greatly affected by disk accesses making clustering an unattractive choice for data analysis. Another source of large data sets is streaming data where you do not store all the data, but process it and delete it. There are some very large data sets for which a little labeled data is available and the rest of the data is unlabeled i.e. for example, computer intrusion detection. Semi-supervised clustering might be applied to this type of data [11]. We do not address this specifically in this paper, but the approach here could be adapted. In general, clustering algorithms which can process very large data sets are becoming increasingly important.

Prodip Hore, Lawrence O. Hall, and Dmitry B. Goldgof are with the Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620, USA (Prodip Hore phone: 813-472-3634; email:phore@csee.usf.edu), (Lawrence O. Hall phone: 813-974-4195; email:hall@csee.usf.edu), (Dmitry B. Goldgof phone: 813-974-4055; email:goldgof@csee.usf.edu)

In this paper we propose a modified fuzzy c means algorithm for large or very large data sets, which will produce a final clustering in a single pass through the data with limited memory allocated. We neither keep any complicated data structures nor use any complicated compression techniques. We will show that our simple single pass fuzzy c means algorithm (SP) will provide almost similar clustering quality (by loading as little as 1% of the data) as that of clustering all the data at once using fuzzy c means (FCM). Moreover, we will also show our single pass fuzzy c means algorithm provides significant speed up when compared with clustering a complete data set, which is less than the size of the memory.

II. Related Work

In [1], a multistage random sampling method was proposed to speedup fuzzy c means. There were two phases in the method. In the first phase, random sampling was used to obtain an estimate of centroids and then fuzzy c means (FCM) was run on the full data with these centroids initialized. A speed-up of 2-3 times was reported. In [9], speed up is obtained by taking a random sample of the data and clustering it. The centroids obtained then were used to initialize the entire data set. This method is similar to that in [1]; the difference is they used one random sample where in [1] they may use multiple random samples. In [2], another method based on sampling for clustering large image data was proposed, where the samples were chosen by the chi-square or divergence hypothesis test. It was shown that they achieved an average speed-up of 4.2 times on image data while providing a good final partition using 24% of the total data.

Another speed up technique for image data was proposed in [3]. In this method FCM convergence is obtained by using a data reduction method. Data reduction is done by quantization and speed-up by aggregating similar examples, which were then represented by a single weighted exemplar. The objective function of the FCM algorithm was modified to take into account the weights of the exemplars. However, the presence of similar examples might not be common in all data sets. They showed that it performs well on image data. In summary, the above algorithms attempt to speed up fuzzy c means either by reducing the number of examples through sampling or by data reduction techniques or by providing good initialization points to reduce the number of iterations. However, the above algorithms do not seem to address the issue of clustering large or very large data sets under the constraint of limited memory. Moreover, some of them address the speed up issues for image data only where the range of

features may be limited. Some work on parallel/distributed approaches has been done, where multiple processors could be used in parallel to speed up fuzzy c means [17], [20]. In [21] a parallel version of the adaptive fuzzy Leader clustering algorithm has been discussed, whereas, in [18] an efficient variant of the conventional Leader algorithm known as ARFL (Adaptive rough fuzzy leader) clustering algorithm was proposed.

There has been research on clustering large or very large data sets [4], [5], [6], [7], [8]. Birch [4] is a data clustering method for large data sets. It loads the entire data into memory by building a CF (Clustering Feature) tree, which is a compact representation of the data using the available memory. Then the leaf entries of the CF tree can be clustered to produce a partition. A hierarchical clustering algorithm was used in their paper. It provides an optional cluster refining step in which quality can be further improved by additional passes over the dataset. However, the accuracy of data summarization depends on available memory. It was pointed out in [5] that depending on the size of the data, memory usage can increase significantly as the implementation of Birch has no notion of an allocated memory buffer. In [5], a single pass hard c means clustering algorithm is proposed under the assumption of a limited memory buffer. They used various data compression techniques to obtain a compact representation of data. In [6], another single pass scalable hard c means algorithm was proposed. This is a simpler implementation of Bradley's single pass algorithm [5], where no data compression techniques have been used. They showed that complicated data compression techniques do not improve cluster quality much while the overhead and book-keeping of data compression techniques slow the algorithm down. Bradley's algorithm compresses data using a primary compression and secondary compression algorithm. In primary compression, data points which are unlikely to change membership were put into a discarded set. The remaining points were then over clustered with a large number of clusters compared to the actual number of clusters. This phase is known as secondary compression and its objective is to save more buffer space by representing closely packed points by their clusters. A tightness criterion was used to detect clusters which can be used to represent points. These sub-clusters were further joined by an agglomerative clustering algorithm provided after merging they are still tightly packed. So, in summary Bradley's implementation has various data compression schemes which involve overhead and book-keeping operations. Farnstorm's single pass [6] algorithm is basically a special case of Bradley's where after clustering, all data in memory is put into a discard set. A discard set is associated with every cluster, and it contains the sufficient statistics of data points in it. The discarded set is then treated as a weighted point in subsequent clustering. Other relevant single pass algorithms for the crisp case can be found in [12] and [13]. Compressing data was also studied in [26], where the number of templates needed by the probalistic neural network (PNN) was reduced by compressing training examples with exemplar. This was done by estimating the probability density functions for the PNN with Gaussian models.

Recently in [10], a sampling based method has been proposed for extending fuzzy and probabilistic clustering to large or very large data sets. The approach is based on progressive sampling and is an extension of eFFCM [2] to geFFCM, which can handle non-image data. However, the termination criteria for progressive sampling could be complicated as it depends upon the features of the data set. They used 4 acceptance strategies to control progressive sampling based on the features of the data. The first strategy (SS1) is to accept the sampled data when a particular feature signals termination. The second one (SS2) is to accept when any one of the features signals termination. The third one (SS3) is to accept when all the features sequentially signal termination and the last one accepts (SS4) when all the features simultaneously signal termination. However, the method could be complicated for a large number of features and the sample size could grow large also.

In [24], two methods of scaling EM [23] to large data sets have been proposed by reducing time spent in E step. The first method, incremental EM, is similar to [28], in which data is partitioned into blocks and then incrementally updating the log-likelihoods. In the second method, lazy EM, at scheduled iterations the algorithm performs partial E and M steps on a subset of data. In [25], EM was scaled in a similar way as they scaled k-means in [5]. In [27], data is incrementally loaded and modelled using gaussians. At the first stage the gaussians models are allowed to overfit the data and then later reduced at a second stage to output the final models. The methods used to scale EM may not generalize to FCM as they are different algorithms with different objective functions.

In [13], a streaming algorithm for hard-c-means was proposed in which data was assumed to arrive in chunks. Each chunk was clustered using a LOCALSEARCH algorithm and the memory was freed by summarizing the clustering result by weighted centroids. This is similar to the method of creating discard set in the single pass hard c means algorithm [6]. Finally, the weighted centroids were clustered to obtain the clustering for the entire stream. We also summarize clustering results in a similar way. The difference between [13], [6] and our approach is in the fact that in fuzzy clustering an example may not completely belong to a particular cluster. Our method of summarizing clustering results involves a fuzzy membership matrix, which does not exist for the crisp cases.

Thus some work has been done for the crisp cases (hard-c-means or hierarchical clustering) and the EM algorithm, but as stated earlier the crisp case may not be easily generalized to fuzzy clustering and to our knowledge no single pass fuzzy c means algorithm has been proposed that takes into account the membership degrees, which describes the fuzziness of each example.

III. Single Pass Fuzzy c means Algorithm

Suppose we intend to cluster a large or very large data set present on a disk. We assume for large or very large data sets the data set size exceeds the memory size. As in [6], we assumed the data set is randomly scrambled on the disk. We can only load a certain percentage of the data based on the available memory allocation. If we load 1% of the data into memory at a time then we have to do it 100 times to scan through the entire data set. We call each such data access a partial data access (PDA). The number of partial data accesses will depend on how much data we load each time. In our approach, after the first PDA, data is clustered into c partitions using fuzzy c means. Then the data in memory is condensed into c weighted points and clustered with new points loaded in the next PDA. We call them "weighted" points because they are associated with weights, which are calculated by summing the membership of examples in a cluster. This is the key difference from the crisp clustering case [6], where a fuzzy membership matrix is not present. In each PDA new singleton points are loaded into memory and clustered along with the past c weighted points obtained from the previous clustering. We will call this partial data clustering (PDC). After clustering these new singleton points along with the past c weighted points, they are condensed again into c new higher weighted points and clustered with examples loaded in the next PDA. This continues until all the data has been scanned once. The objective function of fuzzy c means was modified in a fashion similar to that in [3] to accommodate the effect of weights. We will discuss the calculation of weighted points and the modification of the objective function of fuzzy c means in detail later.

As an example, consider a large or very large data set of n examples. If n_1 examples are fetched in the first PDA and clustered into c partitions then all these n_1 examples in memory are condensed into c weighted points, whose weights will sum up to n_1. Condensation of n_1 examples into c weighted points frees the buffer. Next n_2 examples are then loaded into memory in the next PDA. These new n_2 examples are then clustered along with the c weighted points. So, after the second PDA there will be $n_2 + c$ examples in memory for clustering, out of which c are weighted points and n_2 examples have weight one (singletons). We will call the new fuzzy c means which takes into account the weights of c weighted points the weighted fuzzy c means (WFCM). After clustering these $n_2 + c$ examples in memory using WFCM they are condensed again into c new weighted points. This time the weight of the c points sum up to $n_1 + n_2$ and thus they have more weight than before. This is because there were already c weighted points, summed up to n_1, present when n_2 new singleton examples were loaded in the second PDA. Similarly, after completion of clustering in the third PDA, the weight of the new condensed c points will sum up to $n_1 + n_2 + n_3$. This means after the m^{th} PDA there will be n_m singleton points loaded in the memory along with c weighted points from a previous PDC, whose weights sum up to $n_1 + n_2 + n_3 + ... + n_{m-1}$. So, if the last PDA loads n_l examples, it essentially clusters the whole data set, where $n - n_l$ examples remain as c condensed weighted points and n_l as singleton points. Thus, our simple single pass fuzzy c means will partition the whole data in a single pass through the whole data set. To speed up clustering, we initialize each PDC with the final centroids obtained from the previous PDC. This knowledge propagation allows for faster convergence.

A. Weighted point calculation

The objective function (J_m) minimized by FCM is defined as follows:

$$J_m\left(U,V\right) = \sum\nolimits_{i=1}^{c} \sum\nolimits_{k=1}^{n} U_{ik}^{m} D_{ik}\left(x_k, v_i\right) \tag{1}$$

$U$and V can be calculated as:

$$U_{ik} = \frac{D_{ik}\left(x_k, v_i\right)^{\frac{1}{1-m}}}{\sum_{j=1}^{c} D_{jk}\left(x_k, v_j\right)^{\frac{1}{1-m}}} \tag{2}$$

$$v_i = \frac{\sum\limits_{j=1}^{n} \left(u_{ij}\right)^m x_j}{\sum_{j=1}^{n} \left(u_{ij}\right)^m} \tag{3}$$

Where,

U_{ik} : is the membership value of the k^{th} example, x_k, in the i^{th} cluster.

v_i : is the i^{th} cluster centroid.

n : is the number of examples

c: is the number of clusters

$D_{ik}\left(x_k, v_i\right) = \left\|x_k - v_i\right\|^2$: is the norm. We have used the Euclidean distance.

We will discuss weighted point calculation in this section. We will assume we have a weighted FCM algorithm (WFCM) that takes into account the weights of examples and clusters data into c partitions. The details of WFCM will be discussed in the next section.

Consider n_d examples which are loaded in memory in the d^{th} PDA.

1) Case1: d=1: If d is equal to one i.e. the first PDA, there will be no previous weighted points. In this case WFCM will be the same as FCM. After applying clustering, let v_i be the cluster centroids obtained, where $1 \leq i \leq c$ Let u_{ij} be the membership values, where $1 \leq i \leq c$ and $1 \leq j \leq n_d$. Let $\overrightarrow{w}$ be the weights of the points in memory. In this case all n_d points have weight 1 because no weighted points from previous PDC exist. Now, memory is freed by condensing the clustering result into c weighted points, which are represented by the c cluster centroids v_i, where $1 \leq i \leq c$ and their weights are computed as follows:

$$w_i^{'} = \sum\nolimits_{j=1}^{n_d} \left(u_{ij}\right) w_j,$$

$$1 \leq i \leq c, \quad w_j = 1, \forall 1 \leq j \leq n_d.$$

The weight of the c points, after condensing the clustering results, in memory is as follows:

$$w_i = w_i^{'}, \quad 1 \leq i \leq c.$$

It should be noted that when n_d new singleton points (weight one) are loaded in all subsequent PDA (d>1), their indices associated with $\overrightarrow{w}$ will begin at $c+1$ and end at $n_d + c$ i.e.

$$w_j = 1, \forall c < j \leq n_d + c.$$

2) Case2: d>1: In this case, clustering will be applied on singleton points freshly loaded in the d^{th} PDA along with c weighted points obtained after condensation from the $(d-1)^{th}$ PDC. So, there will be $n_d + c$ points in the memory for clustering using WFCM. The new n_d singleton points have weight one. After clustering, the data in memory (both singletons and weighted points) is condensed into c new weighted points. The new weighted points are represented by the c cluster centroids v_i, where $1 \leq i \leq c$ and their weights are computed as follows:

$$w_i^{'} = \sum\nolimits_{j=1}^{n_d+c} (u_{ij})\, w_j, \quad 1 \leq i \leq c.$$

Then memory is freed up and the weight of the condensed clustering results in memory is updated as follows:

$$w_i = w_i^{'}, \quad 1 \leq i \leq c.$$

B. Weighted FCM (WFCM)

We modified the objective function of FCM (similar to [3]) to take into effect the weighted points. It must be noted that except for the first PDC all subsequent clustering will have c weighted points along with freshly loaded examples. Let us consider that n_d examples (weight one) are loaded from the disk in the d^{th} PDA and there are c weighted points obtained from condensation of examples after the previous PDC and their union constitutes the set $X^{'}$. The cluster centroids and membership matrix for the WFCM are calculated as:

$$v_i = \frac{\sum_{j=1}^{n_d+c} w_j (u_{ij})^m x_j^{'}}{\sum_{j=1}^{n_d+c} w_j (u_{ij})^m}, \quad 1 \leq i \leq c, \quad x_j^{'} \in X^{'}.$$

The weight of each of the n_d examples loaded will be 1. The c weighted points have weight calculated from condensation after the previous PDC.

$$u_{ij} = \left[\sum_{l=1}^{c} \left(\frac{\left\| x_j^{'} - v_i \right\|}{\left\| x_j^{'} - v_l \right\|} \right)^{\frac{2}{m-1}} \right]^{-1}, \quad 1 \leq i \leq c \text{ and } 1 \leq j \leq n_d + c.$$

The modification of objective function does not change the convergence property of FCM because a weighted point can be thought of as many singleton points of identical value. It should be noted that if we have enough memory to load all the data then our simple single pass fuzzy clustering algorithm will reduce to FCM.

IV. Data Sets

There were five real data sets used. They are Iris, ISOLET6, Pen Digits, and two magnetic resonance image data sets (MRI-1 and MRI-2).

The Iris plant data set consists of 150 examples each with 4 numeric attributes [14] and 3 classes of 50 examples each. One class is linearly separable from the other two. We clustered this data set into 3 clusters.

The ISOLET6 Data set is a subset of the ISOLET spoken letter recognition training set and has been prepared in the same way as done in [17]. Six classes out of 26 were randomly chosen and it consists of 1440 instances with 617 features [14]. We clustered this data set into 6 clusters.

The Pen-Based Recognition of Handwritten Digits Data set (Pen Digits) consists of 3498 examples, 16 features, and 10 classes [14]. The classes are pen-based handwritten digits with ten digits 0 to 9. We clustered this data set into 10 clusters.

The MRI-1 data set was created by concatenating 45 slices of MR images of the human brain of size 256X256 from modalities T1, PD, and T2. The magnetic field strength was 3 Tesla. After air was removed, there are slightly above 1 million examples (1,132,545 examples, 3 features). We clustered this data set into 9 clusters.

The MRI-2 data set was created by concatenating 2 Volumes of human brain MRI data. Each Volume consists of 144 slices of MR images of size 512X512 from modalities T1, PD, and T2. The magnetic strength is 1.5 Tesla. For this data set air was not removed and the total size of data set is slightly above 75 million (75,497,472 examples, 3 features). We clustered this data set into 10 clusters..

The values of m used for fuzzy clustering were m=2 for Iris, MRI-1, and MRI-2, m=1.2 for ISOLET6, and m=1.5 for the pen digits data set. The different values enabled reliable partitions with the full data to be obtained.

V. Experimental Setup

In [16], a reformulated optimization criteria R_m , which is mathematically equivalent to J_m (equation 1) was given.

$$R_m(V) = \sum_{k=1}^{n} \left(\sum_{i=1}^{c} D_{ik}\,(x_k, v_i)^{\frac{1}{(1-m)}} \right)^{(1-m)} \tag{4}$$

The new formulation has the advantage that it does not require the U matrix and can be directly computed from the final cluster centroids. For large data sets, where the whole data cannot be loaded into memory, R_m can be computed by incrementally loading examples from the disk.

Each experiment (except MRI-2 data set) was conducted with 50 random initializations. Both FCM and SP were initialized with the same centroids. Each data set was clustered using FCM, loading all the data into memory, and single pass fuzzy c means (SP), loading a chosen percentage of data into memory. For measuring quality, we compute the mean R_m value of FCM and SP. Generally this is the criteria minimized in FCM. In single pass hard c means ([6]), a similar metric was used to compute quality, that is, sum of squared distance between each example and the cluster centroid it belongs. This criteria is minimized in hard c means. Both for SP and FCM, the R_m value of the data set can be computed using the final V matrix (equation 4). We compare quality by computing the difference between the mean R_m value of FCM and SP and then expressing it as a percentage. For

example, if we denote the mean R_m value of experiments of FCM as m_1 and mean R_m value of experiments of SP as m_2 then

Difference in Quality (DQ) is:

$$DQ = \left(\frac{m_2 - m_1}{m_1}\right) * 100 \quad (5)$$

So, a positive value means average value of R_m of FCM is better (lower) while negative value means R_m of SP is better.

We also compare the speed-up obtained by SP compared with FCM. For FCM, as stated earlier, we load the entire data set into memory before clustering. Thus the speed-up reported in this paper is the minimum speed up SP can achieve compared to FCM. This is because for very large data sets the time required by FCM will become more and more due to disk accesses per iteration. Thus we will show that even if we have enough memory to load all the data SP will be significantly faster than FCM while providing almost the same quality partition.

All experiments were run on UltraSPARC-III with 8 processors each of 750 MHz. There was 32GB of main memory. None of the programs were multithreaded, so each program used only one processor at a time.

VI. Results and Discussion

In a single pass experiment, if we load $n\%$ of the data in each PDA, it is denoted by SPn. Experimental results on 4 real data sets are shown in Table I. For small data sets i.e. Iris, Pen digits, and Isolet6 with SP10 (10% data loaded), we got excellent quality partitions over all 50 experiments with different random initializations i.e. on average the difference in quality from FCM is 0.05%, 0.24%, 0.28% for Iris, Pen Digits, and Isolet6 respectively. We also performed experiments on these small data sets under a more stringent condition i.e. loading as little as 5% (loading only 8 of 150 examples each time) for Iris and 1% for Pen digits (loading only 35 of 3498 examples each time) and Isolet6 (loading only 15 of 1440 examples each time). As seen in Table 1 for Pen Digits and Isolet6 we observed on average acceptable differences in quality of 4.80% and 3.18% respectively from FCM over 50 experiments. For Iris with 5% (SP5) data loaded, the difference in quality compared to FCM is a little higher. After we examined the distribution of extrema, we found that out of 50 random experiments 42 (84%) are excellent partitions whose difference in quality with mean R_m value of FCM is only 0.09% and the other 8 got stuck in some not so good extrema whose R_m value is high. We show this in Figure 1 and Figure 2, which shows the extrema distribution both for FCM (Figure 1) and SP5 (Figure 2) on the Iris data set over the 50 random experiments. In Figure 1 and Figure 2, the x-axis indicates the experiment number and y-axis the R_m value. It clearly shows that FCM in all 50 experiments got one type of extremum whose R_m value is near 60, while SP5 found 8 poorer partitions ($R_m$value near 105) and the other 42 were very similar R_m (about 60) values compared to FCM. In Figure 3, we have visualized this poorer partition using 2 out of the 4 features of Iris. The two features used were petal length and petal width as they contain most of the information about the classes. It looks from Figure 3 that the two overlapped classes of Iris came out as a single cluster and the separable one got split into 2 clusters.

SP appears quite stable even under stringent conditions, which are not very realistic. However, it can break if a "too small" percentage is loaded. In summary, on small data sets with 10% data loaded we got excellent partitions, with average quality almost the same as FCM, on all data sets and generally acceptable partitions under stringent conditions.

As our single pass fuzzy c means algorithm is mainly meant for large or very large data sets, results on the magnetic resonance image data set MRI-1, which contains 1,132,545 examples will help us better assess our algorithm. The results (Table I) show that SP achieved excellent partitions whose average quality difference with FCM on all the 50 experiments with different random initializations was only 0.12% and 0 .02% respectively for 1% (SP1) and 10% (SP10) data loaded. So, we believe for medium/large data sets 1% data loaded may be enough to get an excellent partition, that is, with average quality almost the same as FCM.

Table II indicates the average speed-up of SP compared to FCM on the data sets. We excluded Iris as it is quite small. Although, SP is meant for large scale data we reported speed-up for Pen Digits and Isolet6 also. Similar to [6], it seems in general loading and clustering 1% of the data (SP1) is faster than loading 10% of the data (SP10) in each PDA. As shown in Table II, MRI-1 achieved an excellent speed-up (above 43 times with SP1). On the small data sets, the speed up achieved was also significant. It should be noted that FCM loaded all data in memory, so the speed-up reported here for SP is the minimum. For large data sets (assuming it won't fit in memory), FCM will require multiple disk accesses making it slower. The results indicate that SP can be used for speeding up fuzzy c means even if the data can be loaded fully into memory.

The MRI-2 experiment consists of 75,497,472 examples and it takes about 4 to 5 days for FCM to complete each experiment. We clustered the whole data 10 times using FCM. So, we compare it with the first 10 experiments of SP and thus the average results are computed on 10 experiments only. On this data set we loaded only 1% of the data for SP. The average quality difference of FCM with SP1 is 0.0053%. FCM on average took 102.72 hours, above 4 days, to partition the whole data set, while SP took only 1.47 hours. Thus, the speed up obtained is 69.87 times. The quality difference and speed up obtained on this 75 million example data set were also excellent.

In summary, fuzzy c means is widely used in image segmentation and for other data partitioning purposes. Looking at the excellent quality and speed-up achieved by SP, besides using it for clustering large generic data sets, it can also be used to accurately segment, compared to FCM, large volumes of image data quickly. For example MRI-1, which is an entire

volume of MRI data of the human brain, is segmented into 9 clusters accurately in less than a minute on average with our not so fast processor. So, our SP algorithm could help segment the huge amounts of data involved in 3D modeling. For example, to our knowledge segmenting the whole volume of MRI at once using multiple features (generally it is done slice by slice) is done rarely, if ever, but with our SP this can be explored/studied.

TABLE I

DIFFERENCE IN QUALITY OF FCM WITH SP. ALL VALUES EXPRESSED IN PERCENTAGES

Data Sets	Quality Difference
Iris(SP5)	11.96 %
Iris(SP10)	0.05 %
Pen Digits(SP1)	4.80 %
Pen Digits(SP10)	0.24 %
Isolet6(SP1)	3.18 %
Isolet6(SP10)	0.28 %
MRI-1(SP1)	0.12 %
MRI-1(SP10)	0.02 %

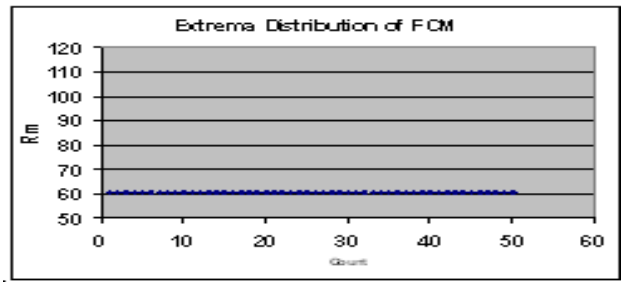

Fig. 1. Extrema distribution of FCM on the Iris data set

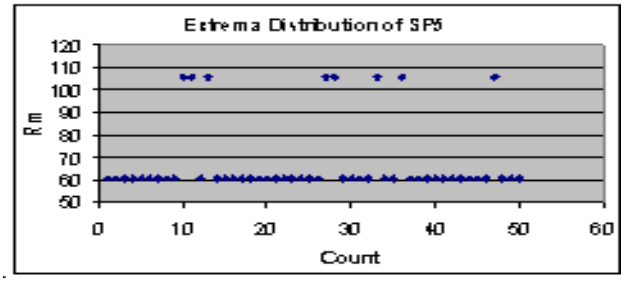

Fig. 2. Extrema distribution of SP5 on the Iris data set

VII. COMPLEXITY ANALYSIS

In [6], time, memory, and disk input/output complexity of single pass hard c-means and the hard-c-means algorithm were discussed. Similarly, we discuss the time, memory, and disk input/output complexity of our single pass fuzzy c means and FCM algorithm. We use the following notation in our discussion.

i number of iterations of FCM over the full data set
$i^{'}$ average number of iterations of SP in each PDC
n number of examples/data points

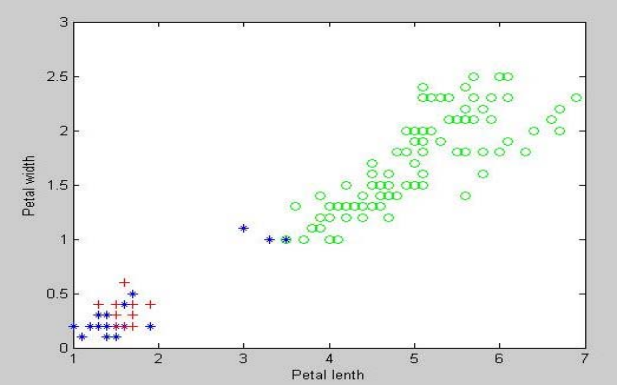

Fig. 3. Plotting of the poor extrema obtained on Iris with SP5 using the dimensions petal length and petal width. Symbols "o", "*", and "+" represent the three clusters.

TABLE II

SPEED UP (SU) OF SP COMPARED TO GC. TIME COMPUTED IN SECONDS

	Global Time(S)	SP10 Time(S)	SP1 Time(S)	SU (SP10)	SU (SP1)
Pen Digits	10.38	1.60	0.81	6.48	12.81
Isolet6	7.06	2.12	1.47	3.33	4.80
MRI-1	2408.60	181.84	54.93	13.24	43.48

f number of dimensions
p fraction of data loaded in each PDA
d number of PDAs required by SP
c number of clusters

The time complexity of FCM is $\mathrm{O}\left(nfc^2i\right)$. For the SP, time complexity in each PDC will be $\mathrm{O}\left(pnfc^2i^{'}\right)$. As there will be d PDCs, the time complexity to cluster the whole data set is $\mathrm{O}\left(pnfc^2i^{'}d\right)$. Because $d = \frac{1}{p}$, it reduces to $\mathrm{O}\left(nfc^2i^{'}\right)$. As stated in [6], we also found $i^{'}$ to be smaller compared to i because after the first PDC we get an approximate model (centroids), and as we pass this knowledge to subsequent PDCs by initializing with previous cluster centroids, the clustering of freshly loaded singleton points along with past weighted points converges quickly. It has been shown before that initializing with "good" centroids helps FCM converge quickly [1].

The memory complexity of FCM is $\mathrm{O}\left(nf + nc\right)$, where nf is the size of data set and nc the size of U matrix. For SP, the memory complexity is $\mathrm{O}\left(pnf + pnc\right)$. As $p < 1$, the memory complexity of SP is less than FCM.

For data sets which cannot be loaded into memory, FCM will have disk accesses every iteration. Thus the disk input output complexity will be $\mathrm{O}\left(nfi\right)$. It is likely that for those data sets the U matrix cannot be kept in memory too. Thus, it will increase the disk input/output complexity further. In case of SP, input/output complexity in each PDA is $\mathrm{O}\left(pnf\right)$. Thus overall complexity for clustering the whole data set is $\mathrm{O}\left(pnfd\right)$, which is $\mathrm{O}\left(nf\right)$.

VIII. Conclusions

In this paper we have proposed a single pass fuzzy c means algorithm for large or very large data sets, which will produce a final clustering in a single pass through the data with limited memory allocated. We neither keep any complicated data structures nor use any complicated compression techniques, yet achieved excellent quality partitions, with an average quality almost the same as FCM, by loading as little as 1% of the data for medium/large data sets and 10% for small data sets. Moreover, our simple single pass fuzzy c means algorithm provides significant speed up even when compared to clustering all the data at once in memory. So, besides using it for clustering large data sets it can also be used for speed up purposes even for data that can be fully loaded into memory. As SP would enable segmenting/partitioning a large amount of data accurately and quickly, it might open new opportunities for fuzzy 3D modeling/segmentation. Future work could be done by adapting it for fuzzy clustering of streaming data. In the future, we also plan to compute quality difference of single pass hard c means from regular hard c means on all these data sets, and compare with the quality difference we reported here.

IX. Acknowledgements

This research was partially supported by the National Institutes of Health under grant number 1 R01 EB00822-01 and by the Department of Energy through the ASCI PPPE Data Discovery Program, Contract number: DE-AC04-76DO00789.

References

[1] Tai Wai Cheng, Dmitry B. Goldgof, and Lawrence O. Hall, *Fast Fuzzy clustering*, Fuzzy Sets and Systems, V. 93, pp. 49–56, 1998.

[2] Nikhil R. Pal and James C. Bezdek, *Complexity Reduction for "Large Image" Processing*, IEEE Transactions on Systems, Man, and Cybernetics, Part B 32(5):pp. 598–611, 2002.

[3] Steven Eschrich, Jingwei Ke, Lawrence O. Hall and Dmitry B. Goldgof, *Fast Accurate Fuzzy Clustering through Data Reduction*, IEEE Transactions on Fuzzy Systems, V. 11, 2, pp. 262–270, 2003.

[4] Zhang Raghu Ramakrishnan Miron Livny, *BIRCH: An Efficient Data Clustering Method for Very Large Databases*, Tian ACM SIGMOD International Conference on Management of Data, pp. 103–114, 1996.

[5] P.S. Bradley, Usama Fayyad, and Cory Reina, *Scaling Clustering Algorithms to Large Databases*, In Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining, KDD-1998, pp., 9–15, 1998.

[6] Fredrik Farnstrom, James Lewis, and Charles Elkan, *Scalability for Clustering Algorithms Revisited*, ACM SIGKDD Explorations, V. 2, pp. 51–57, 2000.

[7] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim, *CURE: An Efficient Clustering Algorithm for Large Databases*, In Proceedings of ACM SIGMOD International Conference on Management of Data, pp. 73–84, 1998.

[8] Raymond T. Ng and Jiawei Han, *CLARANS: A Method for Clustering Objects for Spatial Data Mining*, IEEE Transactions on Knowledge and Data Engineering, 14(5): pp. 1003–1016, 2002.

[9] David Altman, *Efficient Fuzzy Clustering of Multi-spectral Images*, FUZZ-IEEE , 1999

[10] Richard J. Hathaway and James C. Bezdek, *Extending Fuzzy and Probabilistic Clustering to Very Large Data Sets*, Journal of Computational Statistics and Data Analysis, 2006, accepted.

[11] A. Bensaid, J. Bezdek, L.O. Hall, and L.P. Clarke, *Partially Supervised Clustering for Image Segmentation, Pattern Recognition*, V. 29, No. 5, pp. 859-871, 1996.

[12] Chetan Gupta, Robert Grossman, *GenIc: A Single Pass Generalized Incremental Algorithm for Clustering*, Proceedings of the Fourth {SIAM} International Conference on Data Mining (SDM 04), pp. 22–24, 2004.

[13] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, andR. Motwani, *Streaming-Data Algorithms for High-Quality Clustering*, Proceedings of IEEE International Conference on Data Engineering, March 2002.

[14] C.J. Merz and P.M. Murphy. *UCI Repository of Machine Learning Databases Univ. of CA*. Dept. of CIS, Irvine, CA., http://www.ics.uci.edu/∼mlearn/MLRepository.html

[15] John F. Kolen and Tim Hutcheson, *Reducing the Time Complexity of the Fuzzy C-Means Algorithm*, IEEE Transactions on Fuzzy Systems. V. 10, pp. 263–267, 2002.

[16] Richard J. Hathaway and James C. Bezdek, *Optimization of Clustering Criteria by Reformulation*, IEEE Transactions on Fuzzy Systems, V. 3, pp. 241–245, 1995.

[17] Prodip Hore, Lawrence Hall, and Dmitry Goldgof, *A Cluster Ensemble Framework for Large Data sets*, IEEE International Conference on Systems, Man and Cybernetics, 2006, accepted.

[18] S. Asharaf, M. Narasimha Murty, *An adaptive rough fuzzy single pass algorithm for clustering large data sets*, Pattern Recognition, V.36, 2003.

[19] Christian Borgelt and Rudolf Kruse, *Speeding Up Fuzzy Clustering with Neural Network Techniques*, Fuzzy Systems, V. 2, pp. 852–856,2003.

[20] S. Rahmi, M. Zargham, A. Thakre, D. Chhillar, *A Parallel Fuzzy C-Mean Algorithm for Image Segmentation*, Fuzzy Information, V. 1, pp. 234–237,2004.

[21] Alfredo Petrosino and Mauro Verde, *P-AFLC: a parallel scalable fuzzy clustering algorithm*, ICPR , V. 1, pp. 809–812, 2004.

[22] Domingos, P. and Hulten, G.,*A General Method for Scaling Up Machine Learning Algorithms and its Application to Clustering*,Proc. Eighteenth Int'l. Conf. on Machine Learning, pp. 106–113, 2001.

[23] AK Jain and RC Dubes, *Algorithms for Clustering Data. Prentice Hall*, Englewood Cliffs NJ, USA, 1988.

[24] Bo Thiesson, Christopher Meek, and David Heckerman, *Accelerating EM for Large Databases*, Machine Learning Journal, V. 45, pp. 279–299, 2001.

[25] Bradley, P.S., Fayyad, U.M., and Reina, C.A., *Clustering very large databases using EM mixture models*, ICPR, vol 2, pp. 76–80, 2000.

[26] Traven, H. G. C., *A neural network approach to statistical pattern classification by semiparametric estimation of probability density functions*, IEEE Transcations on Neural Networks, V. 2, pp. 366–377, 1991.

[27] Karkkainen and Franti, *Gradual model generator for single-pass clustering*, ICDM , pp. 681–684, 2005.

[28] R. Neal and G. Hinton, *A View of the EM Algorithm that Justifies Incremental, Sparse, and other Variants*, Learning in Graphical Models, pp. 355–368, 1998.