A Synchronization Based Algorithm for Discovering Ellipsoidal Clusters in Large Datasets

Hichem Frigui
Dept. of Electrical & Computer Eng.
University of Memphis
hfrigui@memphis.edu

Mohamed Ben Hadj Rhouma
Department of Mathematics
Georgia Institute of Technology
rhouma@math.gatech.edu

Abstract

This paper introduces a new scalable approach to clustering based on synchronization of pulse-coupled oscillators. Each data point is represented by an integrate-and-fire oscillator, and the interaction between oscillators is defined according to the relative similarity between the points. The set of oscillators will self-organize into stable phase-locked subgroups. Our approach proceeds by loading only a subset of the data and allowing it to self-organize. Groups of synchronized oscillators are then summarized and purged from memory. We show that our method is robust, scales linearly, and can determine the number of clusters. The proposed approach is empirically evaluated with several synthetic data sets and is used to segment large color images.

1. Introduction

Clustering is an effective technique for exploratory data analysis, and has been studied for several years. It has found applications in a wide variety of areas such as pattern recognition, data mining, and statistical data analysis. Most existing methods can be categorized into the following categories: partitioning methods, hierarchical methods, and locality-based methods. Partitional clusters generate a partition of the data such that objects in a cluster are more similar to each other than they are to objects in other clusters. The k-Means[25], EM[10], and k-medoids[19] are examples of partitional methods. Fuzzy partitional algorithms, such as the Fuzzy C-Means[3], generate a “fuzzy partition” where objects can belong to more than one cluster with different degrees. Hierarchical clustering procedures yield a nested sequence of partitions that corresponds to a graphical representation known as the dendrogram. Hierarchical procedures can be either agglomerative or divisive. Locality-based methods group objects based on local relationships. Some locality-based algorithms are density based, while others assume a random distribution.

Recently, the advent of World Wide Web search engines, the problem of organizing massive multimedia databases, and the concept of “data mining” large databases has lead to renewal of interest in clustering and the development of new algorithms[17]. Some of these methods are evolutionary and introduce enhancements and combination of traditional methods, others are revolutionary and introduce new concepts. In this paper, we describe a new approach that can cluster very large data sets while operating within a limited memory (RAM) buffer. The proposed algorithm is based on the synchronization of pulse-coupled oscillators, and is an extension of our recently introduced clustering algorithm, called Self-Organization of Oscillators Network (SOON)[24], to efficiently cluster huge data sets.

The organization of the rest of the paper is as follows. In section 2, we briefly review related work. In section 3, we describe SOON, and in section 4, we instantiate it for the case of hypo-ellipsoidal clusters. In section 5, we extend SOON to handle very large data sets. In section 6, we evaluate the performance of the proposed algorithm. Finally, section 7 contains the conclusions.

2. Related Work

Recently, few algorithms that can cluster large data sets have been proposed. In [19], Kaufman and Rousseeuw proposed CLARA, which is based on finding k representative objects that minimize the sum of the within-cluster dissimilarities. Ng and Han [23] proposed a variation of CLARA called CLARANS, that makes the search for the k medoids more efficient. CLARANS may require several passes over the database making the run time cost prohibitive for very large databases. The ScaleKM[4] and ScaleEM[5] are two other scalable partitional algorithms. These algorithms are based on loading only a subset of the data into the main memory, then using the K-Means [25] or EM [10] to partition it. Both ScaleKM and ScaleEM inherit the drawbacks.
3. Synchronization of Coupled Oscillators

3.1. Background

Mutual synchronization of coupled oscillators is a widespread phenomenon that manifests itself in many fields [21, 22, 31]. One of the most cited examples is that of the southeastern fireflies that gather in large numbers on trees. First, these insects start flashing in random order. Then, they self-organize and start flashing in total synchrony [7]. A characteristic feature of a population of biological oscillators is that they interact with each other by firing sudden impulses. For example, fireflies communicate through light flashes. The mathematical analysis of the details of such interactions is a complex task. An alternative approach is to neglect the details of the shape of the oscillations and model the population by a set of identical Integrate and Fire (IF) oscillators [2]. In this case, each oscillator is characterized by a state variable, which is assumed to be monotonically increasing toward a threshold. When this threshold is reached, the oscillator fires a pulse to the other oscillators, jumps back to a basal level, and a new period begins.

The dynamics of a population of \( N \) coupled oscillators has been investigated by several researchers. It has been shown that a population of \( N \) oscillators can exhibit a chaotic behavior, synchronize, or self-organize into phase-locked sub-groups. The interest in self-organization behavior has increased considerably since evidence that this technique is used by the visual cortex in the mammalian brain have been discovered [11]. Strogatz and Mirollo [22] proved that a population of \( N \) identical concave down IF oscillators with constant excitatory coupling synchronizes for almost all initial conditions. The effect of delay, inhibitory coupling, and local coupling on similar IF models have been examined by several authors [6, 29, 18].

In [24], we introduced a model that combines synchronization and clustering concepts and resulted in an efficient and robust clustering approach. In addition to helping the model self-organize into stable structured groups, the synergy between clustering and synchronization reduces the computational complexity significantly. This is because the number of competing oscillators shrinks progressively as synchronized oscillators get summarized by a single oscillator. A brief description of this model is presented in the following section.

3.2. Self-Organization of Oscillators Network

Let \( Y = \{ y_j | j = 1, \ldots, N \} \) be a set of \( N \) objects, where each object, \( y_j \in \mathbb{R}^p \). We represent each object \( y_j \) by an oscillator \( (O_j) \) which is characterized by a phase variable \( \phi_j \) and a state variable \( x_j \), which evolves according to:

\[
x_j = f(\phi_j) = \frac{1}{b} \ln \left[ 1 + (e^b - 1)\phi_j \right].
\]

In (1), \( b \) is a constant that measures the extent to which \( f \) is concave down. The function \( f \) plays the role of an amplitude function, while \( \phi_i \in [0, 1] \) represents the phase, which in case of no coupling with other oscillators corresponds to the normalized time elapsed since the last firing of \( O_i \). Whenever \( x_i \) reaches a threshold at \( x_i = 1 \), the \( e^b \) oscillator fires (i.e., excites/inhibits other oscillators) and \( \phi_i \) and...
where $\epsilon_i(\phi_j)$ is a coupling function that is positive if $O_i$ and $O_j$ are similar, and negative otherwise. We use

$$
\epsilon_i(\phi_j) = \begin{cases} 
C_E [1 - d_{ij}^2/\delta_0] & \text{if } d_{ij}^2 \leq \delta_0 \\
-C_I [\frac{d_{ij}^2 - \delta_0}{\delta_1 - \delta_0}] & \text{if } \delta_0 < d_{ij}^2 \leq \delta_1 \\
-C_I & \text{otherwise}
\end{cases}
$$

$d_{ij}$ is the dissimilarity between $O_i$ and $O_j$, $C_E$ and $C_I$ (typically $\in [0.01, 0.1]$) are the maximum excitatory and inhibitory coupling. Eq. (3) states that a firing oscillator pulls similar ones closer by an amount proportional to their degree of similarity, and pushes non-similar oscillators farther by an amount proportional to the degree of dissimilarity. $\delta_0$ can be regarded as a resolution parameter, and $\delta_1$ (typically $\delta_1 = 2 \times \delta_0$) is a constant that is used to indicate that if an oscillator is too far, then it should simply be maximally inhibited. The SOON algorithm is summarized below. The mathematical analysis and a proof of convergence for the case of two oscillators can be found in [24].

![Self-Organization of Oscillators Network (SOON)](image)

**Self-Organization of Oscillators Network (SOON)**

Select a dissimilarity measure $d(\cdot)$;  
Initialize phases $\phi_i$ randomly for $i = 1, \ldots , N$;  
Repeat  
Identify next oscillator to fire = $\{O_i : \phi_i = \max_{j=1}^{N} \phi_j\}$;  
Compute $d_{ij}$ for $j = 1, \ldots , N, j \neq i$;  
Bring $\phi_i$ to threshold, and adjust other phases: $\phi_j = \phi_j + (1 - \phi_j)$ for $j = 1, \ldots , N$;  
For all oscillators $O_j$ ($j \neq i$) Do  
Compute state variable $x_j = f(\phi_j)$ using (1);  
Compute coupling $\epsilon_i(\phi_j)$ using (3);  
Adjust state variables using (2);  
Compute new phases using $\phi_i(t^+) = f^{-1}(x_j(t^+))$;  
Identify synchronized oscillators;  
Update the parameters of the synchronized group;  
Reset phases of oscillators that synchronized in this iteration;  
Until (Synchronized groups stabilize);

Fig. 1 illustrates the evolution of the phases for the objects of a simple 2-D data set. The $L_1$ norm is used and $\delta_0$ was set to 0.2 (the choice of $\delta_0$ will be discussed later). Fig. 1(b) displays the initial random phases. Fig. 1(c), which displays the state of the system after 10 iterations, 5 groups have formed: $G_1 = \{y_{12}, y_{14}, y_{15}, y_{17}, y_{18}\}$, $G_2 = \{y_8, y_9\}$, $G_3 = \{y_{10}, y_{11}, y_{13}\}$, $G_4 = \{y_2, y_3, y_6\}$, and $G_5 = \{y_4, y_7\}$. Oscillators 1, 6, and 16 are not assigned to any group yet. As the system evolves further, new groups keep forming, and existing groups keep getting bigger by bringing other oscillators to threshold along with them, and absorbing them. Moreover, a group can also bring another group to threshold, and the two groups will be merged into one. After a total of 25 iterations, only two groups (the 2 actual clusters) are present. The iteration process is stopped when the constant phase difference is detected.

### 3.3. Discussion

At first glance, SOON seems reminiscent of the standard Hierarchical Agglomerative Clustering (HAC) [27] and other neural networks based clustering algorithms such as VQ[14], SOM[20], and ART[8]. However, there are two major properties that distinguish SOON. First, SOON provides an efficient selection mechanism for candidate groups to be updated in every iteration. Most similar algorithms will either select all points in a sequential or in a random order. These simple selection schemes are inefficient since they do not consider the distribution of the data. The selection process in SOON is data driven and relies on the historical behavior (accumulated in the phase). For instance, objects that are similar and are expected to belong to the same cluster will excite each other often, and thus will be selected (i.e., reach the threshold and fire) more frequently. On the other hand, noise objects will get inhibited very often and will rarely reach the threshold.

The second property that distinguishes SOON is that the decision to merge objects is not based on a simple thresholding of the inter-point distances. In SOON, points are merged based on their phase values which are accumulated over several iterations (since $C_E < 1$). This property prevents the “bridging effect” which is a common drawback.
of hierarchical clustering. Fig. 2 illustrates this property. Let $\delta$ be the threshold used in HAC to decide if two points should be merged. If $d(x_i, x_j) < \delta$, then points from the two clusters can be lumped in the same cluster. On the other hand, if $\delta > d$, then no groups can form. SOON does not suffer from this drawback. This is because the decision to merge two points depends not only on the distance between them, but also on the behavior of the neighboring points in the previous iterations. For instance, if $d$ is less than $\delta$, then $x_1$ will receive excitatory pulses from $x_2$ and $x_3$, and inhibitory pulses from all the other points. On the other hand, $x_2$ will receive excitatory pulses from $x_1$ and most of the points in the left cluster. Thus, it is expected that after a few iterations the states of $x_1$, $x_2$ and $x_3$ will be different (even if they had similar initial values), and therefore, it is unlikely for these points to be lumped together.

\[
\begin{array}{c}
\text{Figure 2. Bridging Effect of HAC}
\end{array}
\]

The SOON clustering framework is generic and can be used to cluster feature vectors composed of numerical as well as categorical attributes. Moreover, it can incorporate a multitude of similarity measures including subjective and non-metric ones. In this paper, we instantiate and numerically illustrate SOON for the case of hyper-ellipsoidal clusters. We also introduce an incremental version of SOON to cluster very large data that cannot be fully loaded into the main memory.

4. Identification of Hyper Ellipsoidal clusters

We assume that each cluster, $\mathcal{C}_k$, has a multivariate normal distribution with mean $\mu_k$ and covariance matrix $\Sigma_k$. The synchronization process involves computing the distance between two oscillators, where each oscillator represents a cluster. We use a modified Mahalanobis distance:

\[
d^2_{ij} = \min \{ (c_i - c_j)^T \Sigma_i^{-1} (c_i - c_j), (c_i - c_j)^T \Sigma_j^{-1} (c_i - c_j) \},
\]

where $c_i$, $c_j$, $\Sigma_i$, $\Sigma_j$ are the centers and covariance matrices of oscillators (clusters) $O_i$ and $O_j$. By using a different covariance matrix for each cluster, the above distance can be used to detect ellipsoidal clusters of various shapes and orientations. The shape of each cluster $k$ is generated by the eigenstructure of the matrix $\Sigma_k$. If the clusters are assumed to come from a multivariate Gaussian distribution, then the within-cluster distances have a $\chi^2$ probability distribution with $p$ (dimensionality of the data) degrees of freedom. This desirable feature will (i) automate the choice of $\delta_0$; and (ii) make the neighborhood of the excitatory region dynamic and cluster dependent. We use $\delta_0 = \chi^2(\alpha, p)$, that is, the $\alpha^{th}$ percentile of the $\chi^2$ probability distribution. In other words, if the probability that a given point belongs to the firing cluster is greater than $(1 - \alpha)$, then its phase would be incremented. We use $\alpha = 95\%$.

Initially, each data point is represented by one oscillator and constitute a cluster by itself. The initial center of the oscillator (i.e., cluster) is the point itself, and the covariance matrix is initialized to $\epsilon \cdot I_{p \times p}$, where $I_{p \times p}$ is the identity matrix, and $\epsilon$ is a constant that depends on the dynamic range of the data. The center and the covariance matrix of each cluster will be updated using the features of the synchronized oscillators.

\[
\begin{array}{c}
\text{Figure 3. Detection of ellipsoidal clusters. Results at the end of (a) 5, (b) 500, and (c) 1200 iterations.}
\end{array}
\]

\[
\begin{array}{c}
\text{Figure 4. Detection of ellipsoidal clusters in a noisy data set. Results at the end of (a) 5, (b) 500, and (c) 1700 iterations.}
\end{array}
\]

Fig. 3 illustrates the evolution of SOON using a 2-D synthetic Gaussian mixture. Fig. 3(a) displays the results after 5 iterations where 4 small groups have formed. The center of each group is indicated by the “+” sign and the ellipses enclose points within the $\delta_0$ neighborhood. The remaining points indicate oscillators that did not synchronize yet. Fig. 3(b) displays the results after 500 iterations, where most os-
oscillators have synchronized and formed groups of various sizes. Fig. 3(c) shows the result after 1200 iterations where the remaining groups are phase locked. Fig. 4 illustrates the robustness of SOON to noise. As can be seen, noise points will either form very small clusters or will not synchronize at all. This is because noise points are located in non-dense regions and they get inhibited by most of the other points. Even if these points reach the threshold, they will excite a few if any other oscillators.

The previous two examples might indicate that SOON is not efficient since it takes more than 1500 iterations to obtain the final partition, and that a simple K-means or EM algorithm can converge in less than few hundred iterations. However, this is not the case. Unlike the K-means and EM which compute the distance between all points and all clusters in every iteration, SOON computes the distance to only one of the clusters (the one that fires) in every iteration. Thus, to provide a fair comparison, the number of iterations in SOON should be divided by the number of clusters. Moreover, K-means and EM would need many more iterations if the number of clusters is not known, and the computation in each iteration gets more complex for K-means type algorithms that are robust to noise [13].

To provide a quantitative indication of the efficiency of our approach, we compare the running time with two other algorithms. The first algorithm is the Fuzzy C-Means (FCM) [3] using the Mahalanobis distance which is similar to the EM algorithm. The FCM is very simple, however, it cannot handle noisy data, and requires the specification of the number of clusters. The second algorithm is the Robust Competitive Agglomeration (RCA) [13]. The RCA can cluster noisy data and find the optimal number of clusters. It only requires the specification of an over-specified number of clusters ($C_{max}$). Table 1 displays the average CPU time of the three algorithms over several runs using the data sets in Fig. 3 and Fig. 4. We should note here that the FCM is sensitive to the initial parameters, and that there are few instances where the small cluster was not detected. Both the RCA and SOON are not as sensitive because they start with a much larger number of initial prototypes. Notice that for the data in Fig. 4, the FCM cannot generate a meaningful partition, and thus the CPU time is not recorded.

### Table 1. CPU time of 3 clustering algorithms

<table>
<thead>
<tr>
<th>Image</th>
<th>Algorithm</th>
<th>FCM ($C=6$)</th>
<th>RCA ($C_{max}=25$)</th>
<th>RCA ($C_{max}=50$)</th>
<th>SOON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 3</td>
<td>2 Sec.</td>
<td>6 Sec.</td>
<td>10 Sec.</td>
<td>3 Sec.</td>
<td></td>
</tr>
<tr>
<td>Fig. 4</td>
<td>*****</td>
<td>7 Sec.</td>
<td>11 Sec.</td>
<td>4 Sec.</td>
<td></td>
</tr>
</tbody>
</table>

## 5 Incremental Clustering using SOON

During the evolution of SOON, once a group of oscillators synchronize, they will share the same center and covariance matrix. Thus, a group of synchronized oscillators can be treated as a single oscillator. Our goal is to exploit this fact, and design an algorithm that can efficiently cluster huge data sets while operating within a limited memory (RAM) buffer in one scan of the data set. Our approach, called ScaleSOON, is outlined as follows:

1. Get a sample from the data set, and fill the memory buffer.
2. Apply the algorithm to the data contents in the buffer.
3. Summarize each synchronized group by a single oscillator with equivalent sufficient statistics, and purge the synchronized oscillators from the buffer.
4. If there are any points that have not been previously loaded, go to step 1.

### 5.1. Data Compression and Sufficient Statistics

Let $G$ be a group of synchronized oscillators. If the distance defined in (4) is used, then the sufficient statistics for $G$ are the triplet $(N_G, S_G, SS_G)$, where $N_G$ is the number of oscillators in group $G$, $S_G = \sum_{y \in G} y$, and $SS_G = \sum_{y \in G} yy^T$. Note that $SS_G$ is symmetric, and there is no need to store the entire matrix. Moreover, when the dimensionality of the feature space is large, it is common in practice to assume diagonal covariance matrices. In this case, the shape of $SS_G$ reduces to a vector. Initially, each feature vector (or oscillator) $y_j$ has its initial sufficient statistics, i.e., $(N_{y_j}, S_{y_j}, SS_{y_j}) = (1, y_j, y_j y_j^T)$.

### 5.2. Purging and Filling the Memory Buffer

When the data set resident in the memory buffer reaches a stable state, each group of synchronized oscillators, $G_k$, will be compressed as follows:

1. **Summarization**: Add a new oscillator with sufficient statistics:

   \[
   \begin{align*}
   N_{G_k} &= \sum_{y \in G_k} N_y \\
   S_{G_k} &= \sum_{y \in G_k} S_y \\
   SS_{G_k} &= \sum_{y \in G_k} SS_y.
   \end{align*}
   \]

2. **Purging**: Purge the oscillators that belong to $G_k$ from the memory buffer.
In (5), \( y \) can represent a single oscillator or a group of oscillators that have been synchronized and summarized in previous iterations. After summarizing and purging all of the synchronized groups, the memory buffer is filled with feature vectors that have not been previously loaded. Since the features of the purged oscillators are not available in the main memory, the center and covariance matrix should be updated using the sufficient statistics of the group they belong to. That is, we use

\[
    c_k = \frac{1}{N_{G_k}} S_{G_k} \quad \text{and} \quad C_k = \frac{1}{N_{G_k}} S S_{G_k} - c_k c_k^T
\]

6. Experimental Results

The performance of ScaleSOON is evaluated with several data sets. We demonstrate the scalability of ScaleSOON with respect to the number of records and the number of attributes using several synthetically generated data sets. All experiments were performed on an Ultra Sparc Ii 300 Mhz workstation with 256 MB RAM. Several data sets were generated by sampling from \( k=10 \) multivariate Gaussian distributions. Gaussian means and diagonal covariance matrices were chosen uniformly on \([0, 10] \), and \([0.8, 1.2] \) respectively. The number of attributes varied from 10 to 100, and the number of records varied from 10,000 to 1,000,000.

6.1. Effect of the Buffer Size

Initially, ScaleSOON was designed to cluster very large data sets that cannot be entirely loaded into the main memory, and we planned on setting the buffer size to be close to the size of the main memory. However, after running ScaleSOON on several data sets with different buffer sizes, we have discovered that it runs faster with smaller buffer sizes. Fig. 5 shows the running time for an increasing sequence of buffer sizes varying from 100 to 10,000. The data used in this experiment has 25 attributes and 100K records. There are two reasons that explain this behavior. First, the algorithm exhibits temporal locality and the frequency of cache misses would be minimized when the buffer size is less than the cache size. Second, we have noticed that ScaleSOON reaches a stable state in fewer iterations when the buffer size is smaller.

When the buffer size becomes very small, the running time does not decrease any further. This is probably due to the more frequent accesses to the secondary storage, and also because not all of the available cache memory is used. In addition to inefficient use of cache memory, very small buffer sizes cannot maintain a variety of sub-groups, and may lead to an unsatisfactory partition. Ideally, the buffer size should be much larger than the expected number of clusters to allow a smooth hierarchical agglomeration. In the remainder of this section, we will assume that the number of clusters is always much smaller than 500, and let the buffer size be fixed to 500 oscillators.

6.2. Scalability

To illustrate the scalability of ScaleSOON with respect to the number of records, we use data sets that have 25 attributes, 10 clusters, and 10K, 100K, 300K, 500K, and 1M records. For all data sets, ScaleSOON found 10 clusters and identified their parameters correctly. Fig. 6(a) displays the running time versus the number of records. As expected, ScaleSOON scales linearly. To illustrate the scalability with respect to the number of attributes, we use data sets that have 10 clusters, 500K records, and 10, 25, 50, and 100 attributes. For all data sets, ScaleSOON found 10 clusters and identified their parameters correctly. Fig. 6(b) displays the running time versus the number of records. As can be seen, ScaleSOON scales linearly with respect to the number of attributes.

6.3. Effect of the Initial Phases

ScaleSOON assigns a random number (initial phase) to each record when it gets loaded. To study the sensitivity of the algorithm with respect to these random phases, we ran ScaleSOON on the same data set with several different
initial phases (using different seed values). The data used in this experiment has 25 attributes, 100K records, and 10 clusters. Fig. 7(a) displays the total distance between the true Gaussian means and the means of the identified clusters for the 10 different initializations. As can be seen, the total distance does not exceed 0.11, which means that for all 10 initializations, ScaleSOON was able to identify the 10 correct clusters. Moreover, as shown in Fig. 7(b), the variation in the run time for these 10 different runs is small (less than 3%). This indicates that the total number of iterations required for the algorithm to reach a stable state is not sensitive to the initial phases.

Figure 7. Effect of the Initial phases. (a) Total Euclidean distance from the true Gaussian means. (b) Running time.

6.4. Applications to Color Image Segmentation

ScaleSOON has been used to segment large color images. Each image contains $512 \times 768$ pixels, and each pixel is mapped to an 8-D vector consisting of 3 color, 3 texture, and 2 position features [9]. Typically, when using a clustering algorithm to segment large images, sampling is used to reduce the number of pixels. Unfortunately, sampling may result in the loss of small and/or thin objects. Fig. 8(a) shows 2 color images from the Corel image database. ScaleSOON was applied to each image (without sampling) to segment it into several homogeneous regions by clustering the feature vectors mapped from the image pixels. Fig. 8(b) shows the edges of the clustered objects.

Figure 8. Color image segmentation. (a) Original images, (b) Edges of detected objects

ScaleSOON achieves scalability by loading only a subset of the data into main memory at a time, and allowing it to self-organize. Each group of oscillators is summarized by a single oscillator with sufficient statistics and are purged from memory. New data points are then loaded into memory and self-organize with the existing groups. Our empirical evaluation has shown that ScaleSOON scales linearly with respect to the number of records and the number of attributes. Our experiments have also indicated that even if large memory is available to hold the entire data, it is more efficient to process it incrementally. This is because the problem is much simpler when fewer oscillators are interacting, and also the cache memory is used more efficiently. ScaleSOON does not require the specification of the actual number of clusters. This number is determined automatically depending on the resolution parameter $\delta_0$. We have shown that using the Mahalanobis distance, $\delta_0$ can be fixed by exploiting the $\chi^2$ probability distribution of the distances in each cluster. Thus, each cluster can have its own adaptive resolution. The synchronization mechanism of ScaleSOON makes it robust to noise and outliers. Oscillators that represent noise points will synchronize with few if any other oscillators. Thus, they can be simply identified as those points that did not synchronize, or that belong to very small groups.

In this paper, we have instantiated our clustering framework using the Mahalanobis distance. This, however, does not constitute a limitation of our method. In fact, since ScaleSOON does not explicitly optimize an objective function, it can incorporate non-metric similarity measures. Moreover, it can be easily adapted to cluster discrete-valued measurements. This represents a very desirable features in
many data mining applications where discrete data is very common and the most appropriate dissimilarity measures tend to be rather subjective, and thus non-metric.

Acknowledgment

Partial support of this research was provided by the Woodrow W. Everett, Jr. SCE EE Development Fund in cooperation with the Southeastern association of electrical engineering department heads.

References