A HYBRID CLUSTERING TECHNIQUE THAT COMBINES REPRESENTATIVE-BASED AND AGGLOMERATIVE CLUSTERING

Ji Yeon Choo, Rachsuda Jiamthapthaksin,
Chun-sheng Chen, Oner Ulvi Celepcikay, Christian Giusti, and
Christoph F. Eick

Department of Computer Science
University of Houston
Houston, TX, 77204, USA
http://www.cs.uh.edu

Technical Report Number UH-CS-06-13

December 15, 2006

Keywords: Post-processing, hybrid clustering, finding clusters of arbitrary shape, agglomerative clustering, supervised clustering.

Abstract

Representative-based clustering algorithms form clusters by assigning objects to the closest cluster representative. On the one hand, they are quite popular due to their relative high speed and due to the fact that they are theoretically well understood. On the other hand, the clusters they can obtain are limited to spherical shapes and clustering results are also highly sensitive to initializations. In this paper, a novel agglomerative cluster post-processing technique is proposed, which merges neighboring clusters greedily maximizing a given objective function and uses Gabriel graphs to determine which clusters are neighboring. Non-spherical shapes are approximated as the union of small spherical clusters that have been computed using a representative-based clustering algorithm. We claim that this technique leads to clusters of higher quality compared to running a representative clustering algorithm stand-alone. Empirical studies were conducted to support this claim; for both traditional and supervised clustering significant improvements in clustering quality were observed for most datasets. Moreover, as a byproduct, the paper also introduces and evaluates internal cluster evaluation measures and sheds some light on technical issues related to representative clustering algorithms in general.

A HYBRID CLUSTERING TECHNIQUE THAT COMBINES REPRESENTATIVE-BASED AND AGGLOMERATIVE CLUSTERING

Ji Yeon Choo[1], Rachsuda Jiamthapthaksin[1], Chun-sheng Chen[1],
Oner Ulvi Celepcikay[1], Christian Giusti[2], and Christoph F. Eick[1]

Abstract

Representative-based clustering algorithms form clusters by assigning objects to the closest cluster representative. On the one hand, they are quite popular due to their relative high speed and due to the fact that they are theoretically well understood. On the other hand, the clusters they can obtain are limited to spherical shapes and clustering results are also highly sensitive to initializations. In this paper, a novel agglomerative cluster post-processing technique is proposed, which merges neighboring clusters greedily maximizing a given objective function and uses Gabriel graphs to determine which clusters are neighboring. Non-spherical shapes are approximated as the union of small spherical clusters that have been computed using a representative-based clustering algorithm. We claim that this technique leads to clusters of higher quality compared to running a representative clustering algorithm stand-alone. Empirical studies were conducted to support this claim; for both traditional and supervised clustering significant improvements in clustering quality were observed for most datasets. Moreover, as a byproduct, the paper also introduces and evaluates internal cluster evaluation measures and sheds some light on technical issues related to representative clustering algorithms in general.

Index Terms

Post-processing, hybrid clustering, finding clusters of arbitrary shape, agglomerative clustering, supervised clustering.

I. Introduction

Representative-based clustering algorithms form clusters by assigning objects to the closest cluster representative. K-means is the most popular representative-based clustering algorithm; it uses cluster centroids as representatives and iteratively updates clusters and centroids until no change in the clustering occurs. K-means is a relatively fast clustering algorithm with a complexity of $O(k \cdot t \cdot n)$ with typically k, t << n. The clusters generated are always contiguous. However, when using K-means the number of clusters k has to be known in advance, and K-means is very sensitive to initializations and outliers. Figure 1 demonstrates the initialization problem of K-means. The original 9Diamonds dataset contains 9 natural clusters as shown in Figure 1 (a). However, if initial representatives of clusters are not properly chosen, it fails to identify the nine natural clusters, as illustrated in Figure 1 (b). Another problem of K-means

[1]Computer Science Department, University of Houston,
Houston, TX 77204-3010

[2]Department of Mathematics and Computer Science, University of Udine
Via delle Scienze, 33100, Udine, Italy.

clustering algorithm is that it cannot obtain clusters that have non-spherical shapes [1]: the shapes that can be obtained by representative-based clustering algorithms are limited to convex polygons.

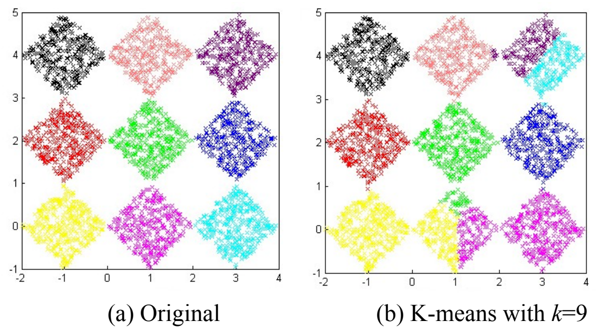

(a) Original (b) K-means with *k*=9

Fig. 1. Applying K-means to the 9Diamonds dataset.

In theory, agglomerative hierarchical clustering (AHC) is capable of detecting clusters of arbitrary shape. However, in practice, it performs a very narrow search, merging the two closest clusters without considering other merge candidates and therefore often misses high quality solutions. Moreover, its time complexity of $O(n^2)$ limits its application to small and medium-sized data sets. Furthermore, clusters obtained by AHC are not necessarily contiguous, as illustrated in Fig. 2: a hierarchical clustering algorithm that uses average linkage[2] would merge clusters C3 and C4, although the two clusters are not neighboring. This example motivates the need to disallow merging of non-neighboring clusters in agglomerative clustering.

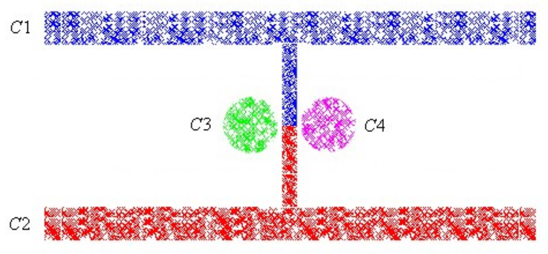


Fig. 2. Merging elongated clusters

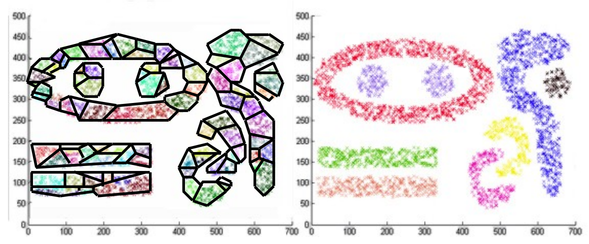


Fig. 3. An illustration of Post-Processing technique

[2] Average linkage uses the average distance between the members of two clusters as its distance function.

This paper proposes a hybrid clustering technique that combines representative-based with agglomerative clustering trying to maximize the strong points of each approach. In particular, a novel agglomerative cluster post-processing technique is proposed, which merges neighboring clusters greedily maximizing a given fitness function and uses Gabriel graphs [2] to determine which clusters are neighboring. Non-spherical shapes are approximated as the union of small spherical clusters that have been computed using a representative-based clustering algorithm, as illustrated in Fig. 3. By using Gabriel graphs the agglomerative clustering algorithm conducts a much wider search which, we claim, results in clusters of higher quality. Moreover, the expensive, agglomerative clustering algorithm is only run for usually less than 500 iterations; therefore, the impact of its higher complexity on the overall run time is alleviated, particularly for very large data sets. Furthermore, the proposed post-processing technique is highly generic in that it can be used with any representative-based clustering algorithm, with any proximity graph and with any cluster evaluation function. Figure 4 gives the pseudo code of the proposed post-processing technique.

```
1. Run a representative-based clustering algorithm to
   create a large number of clusters.
2. Read the representatives of the obtained clusters.
3. Create a merge candidate relation using proximity
   graphs.
4. WHILE there are merge-candidates ($C_i$ ,$C_j$) left
     BEGIN
       Merge the pair of merge-candidates ($C_i$,$C_j$), that
       enhances fitness function $q$ the most, into a new
       cluster $C'$
       Update merge-candidates:
       $\forall C$ Merge-Candidate($C'$,$C$) $\Leftrightarrow$ Merge-
           Candidate($C_i$,$C$) $\vee$ Merge-Candidate($C_j$,$C$)
     END
5. RETURN the best clustering $X$ found.
```

Fig. 4. Pseudo code of the Post-Processing Technique

The proposed post-processing technique merges neighboring clusters greedily: the pair of clusters whose merging maximizes an externally given the fitness function q is merged, and this process is continued until only one cluster is left. Finally, the best clustering is returned. Using cluster representatives that have been obtained by running a representative-based clustering algorithm as an input, a proximity graph is generated to determine which of the original clusters are neighboring and a merge-candidate relation is constructed from the proximity graph. When clusters are merged, this merge-candidate relation is updated incrementally without any need to regenerate proximity graphs.

The remainder of the paper explains the post-processing technique in more detail. We claim that this technique leads to clusters of higher quality compared to running a representative-based clustering algorithm stand-alone. Empirical studies were conducted to support this claim; for both traditional and supervised clustering significant improvements in clustering quality were observed for most data sets. The paper is organized as follows. Section 2 describes our post-processing algorithm in more detail, and applies it to traditional clustering problems. In Section 3, we introduce a post-processing framework for supervised clustering. Related work is reviewed in Section 4, and a conclusion is given in Section 5.

II. Post-Processing for Traditional Clustering

In this section, first proximity graphs are introduced and their role in cluster post-processing is discussed. Next, internal cluster evaluation measures will be discussed that will serve as fitness functions for the post-processing algorithm. Finally, results of experimental studies will be discussed that evaluate the proposed post-processing algorithm.

A. Using Gabriel Graphs for Determining Neighboring Clusters

Different proximity graphs represent different neighbor relationships for a set of objects. There are various kinds of proximity graphs [3], with Delaunay graphs [4] (DG) being the most popular ones. The Delaunay graph for a set of cluster representatives tells us which clusters of a representative-based clustering are neighboring and the shapes of representative-based clusters are limited to Voronoi cells, the dual to Delaunay graphs. Voronoi cells are always convex polygons, but there are convex shapes that are different from Voronoi cells.

Delaunay triangulation (DT) [5] is the algorithm that constructs the Delaunay graphs for a set of objects. Unfortunately, using DT for high dimensional datasets is impractical since it has a high complexity of $O\left(n^{d/2}\right)$ (when d>2), where *d* is the number of dimensions of a data set. Therefore, the proposed post-processing algorithm uses another proximity graph called Gabriel graphs (GG) [2] instead, which is a subgraph of the DG. Two points are said to be Gabriel neighbors if their diametric sphere does not contain any other points. The pseudo code of an algorithm that constructs the GG for a given set of objects is given in Figure 5; its time complexity is $O\left(dn^3\right)$. Faster, approximate algorithms $\left(O\left(dn^2\right)\right)$ to construct GG exist [6].

```
Let $R = \{r_1, r_2, \ldots, r_k\}$, be a set of cluster representatives
FOR each pair of representatives $(r_i, r_j)$,
  IF for each representative $r_p$, the following inequality
     $d(r_i,r_j) \leq \sqrt{d^2(r_i,r_p) + d^2(r_j,r_p)}$
     where $p \neq i, j$ and $r_p \in R$, is true,
THEN $r_i$ and $r_j$ are neighboring.
$d(r_i,r_j)$ denotes the distance of representatives $r_i$ and $r_j$.
```

Fig. 5. Pseudo code for constructing Gabriel graphs.

Gabriel graphs are known to provide good approximations of Delaunay graphs because a very high percentage of the edges of a Delaunay graph are preserved in the corresponding Gabriel graph. Our proposed post-processing algorithm constructs the Gabriel graph for a given set of representatives, e.g. cluster centroids in the case of K-means, and then uses the Gabriel graph to construct a boolean merge-candidate relation that describes which of the initial clusters are neighboring. This merge candidate relation is then updated incrementally when clusters are merged.

B. Cluster Evaluation Measures for Traditional Clustering

The purpose of the cluster evaluation is to assess their quality. Cluster evaluation measures are classified into unsupervised, supervised and relative [7]. In this paper, we only consider internal evaluation measures. One of the popular methods for representative-based algorithms, such as K-means or K-medoids, is the Mean Square Error (MSE) that is defined as follows:

$$MSE = \sum_{o \in C_i} d\left(o, r_i\right)^2$$

where *d(o,r_i)* is the distance between object *o* and representative of cluster *i*, r_i.

Our post processing technique is similar to agglomerative hierarchical clustering algorithms in that it iteratively merges two candidates. However, it differs from a hierarchical clustering algorithm that it merges neighboring clusters that enhance a given fitness function the most, and does not blindly merge clusters that are closest to each other. Two fitness functions $q1(X)$ and $q2(X)$ that use cohesion, separation, and silhouette coefficient are introduced for this purpose in the following.

The fitness function $q1(X)$ combines the cohesion and separation. Cohesion measures the tightness of a cluster while separation measures how distinct or well-separated a cluster is from other clusters. The fitness function $q1(X)$ is defined as:

Let
$O=\{o_1,\ldots,o_n\}$ be the dataset to be clustered,
d_{ij} be the distance between objects o_i and o_j,
$X=\{C_1,\ldots,C_k\}$ be a clustering of O with
$C_i \subseteq O$ and $C_i \cap C_j = \emptyset$ for $i \neq j$
$Intra(X)$ be the number of intra-cluster distances and
$Inter(X)$ be the number of inter-cluster distances in a clustering X.

$$q1(X)=\frac{Separation(X)^{\delta}}{Cohesion(X)^{(2-\delta)}}$$

where,

$$Cohesion(X)=\frac{Sumintra(X)}{Intra(X)},\ Separation(X)=\frac{Suminter(X)}{Inter(X)}$$

$$Sumintra(X)=\sum\nolimits_{\text{i<j, } o_i \text{ and } o_j \text{ belong to a same cluster}} d_{ij}$$

$$Suminter(X)=\sum\nolimits_{\text{i<j, } o_i \text{ and } o_j \text{ belong to a different cluster}} d_{ij}$$

and

$$\forall X\left(Inter(X)+Intra(X)=\frac{n(n-1)}{2}\right)$$

Fitness function $q2(X)$ uses the popular silhouette coefficient [8]; for each object o_i belonging to cluster C_k its silhouette coefficient is defined as follows:

$$s_i=\frac{(a_i-b_i)}{max(a_i,b_i)}$$

where,

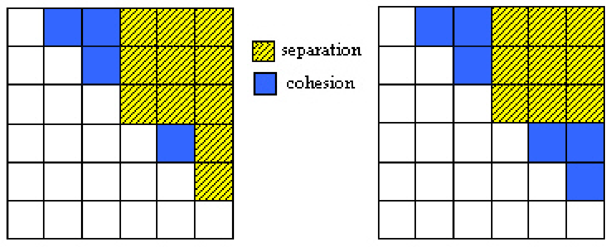
$$a_i=min_m\left(\frac{1}{|C_m|}\sum_{o_j\in C_m} d_{ij}\right),\quad m\neq k$$

$$b_i=\frac{1}{|C_k|}\sum_{o_j\in C_k} d_{ij}$$

In the above formula, b_i is average dissimilarity of an object o_i to all other objects o_j in the same cluster. a_i is minimum of average dissimilarity of an object o_i to all objects o_j in another cluster (the closest cluster). To measure quality not for one object but entire clustering, we use average of silhouette over whole dataset. The fitness function $q2(X)$ is defined as follows:

$$q2(X)=\frac{1}{n}\sum\nolimits_{i=1}^{n} s_i$$

where n is the number of objects in a dataset.
In our approach we directly use fitness functions $q1(X)$ and $q2(X)$ to determine which cluster is merged next.



(a) Before merging (b) After merging

Fig. 6. Swapping of distances when clusters are merged.

In general, it is computationally expensive to compute cohesion and separation from the scratch each time clusters are merged. As it turns out, cohesion and separation can be computed incrementally from their old value relying on the following formulas. Let *X'* be the clustering that has been obtained by merging clusters C_p and C_q in the clustering *X*; in this case, the cohesion and separation of the new clustering *X'* can be computed incrementally by just maintaining 4 variables (see also Figure 6).

$$Cohesion(X') = \frac{Sumintra(X) + d(C_p, C_q)}{Intra(X) + (|C_p| \cdot |C_q|)}$$

$$Separation(X') = \frac{Suminter(X) - d(C_p, C_q)}{Inter(X) - (|C_p| \cdot |C_q|)}$$

$$Intra(X') = Intra(X) + (|C_p| \cdot |C_q|)$$

$$Inter(X') = Inter(X) - (|C_p| \cdot |C_q|)$$

where $d\left(C_p, C_q\right)$ is the sum of inter-cluster distances with respect to C_p and C_q.

C. Complexity

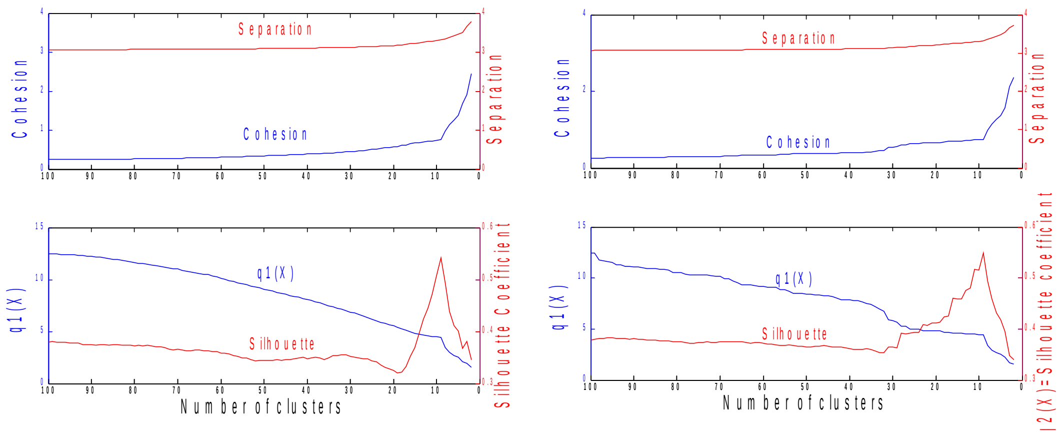
The time complexity of the post-processing algorithm is the sum of the cost of K-means, constructing Gabriel graphs and post-processing. We use the K-means algorithm supported in MATLAB in our implementation. K-means clustering algorithms run relatively efficient in $O(k \cdot t \cdot n)$ time where *k* is the number of clusters, *t* is the number of iterations and *n* is the number of objects. We use the function for Gabriel graphs bundled in the library of MATLAB [Strauss06] which has a cost of $O(dk^3)$. If distances, merge-candidates and fitness values after merging clusters are computed from scratch, the complexity of the post processing technique is $O(k^2 \cdot n^2)$. We claim that this complexity can be improved to $O(n^2)$ by updating cohesion and separation or silhouette coefficients in an incremental way. In summary, the complexity of the current implementation of the post-processing technique is $O(k \cdot n \cdot t + k^3 + n^2)$. Since *n* >> *t* and *k*, we can approximate the two complexity functions, obtaining respectively $O(n^2)$. For more details about incremental update of fitness function, see the Appendix 1. For instance, in our experiments the post-processing algorithm spends 367 seconds constructing Gabriel graphs and performing post processing for *k*=100.

D. Experiments

We conducted our experiments on a Dell Inspiron 600m laptop with a Intel(R) Pentium(R) M processor 1.6GHz and 512 MB of RAM. The proposed post-processing algorithm was tested with the two different fitness functions $q1(X)$ and $q2(X)$ for eight datasets whose properties are described in TABLE I. The experimental results of the post processing algorithm were compared to the results generated by the K-means clustering algorithm for the same number of clusters, k. In particular, for each k we run K-means 10 times and then we compute the maximum and the average of the two fitness functions.

TABLE I
INFORMATION OF DATASETS

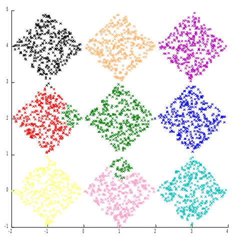
Dataset	Number of objects	Number of dimensions	Number of classes
9Diamonds	3,000	2	9
Volcano	1,533	2	2
Binary Complex 8	2,551	2	8
Binary Complex 9	3,031	2	9
Earthquakes 1%	3,161	2	3
Arsenic 20%	2,385	2	2
Vehicle	8,469	19	4
Ionosphere	351	34	2
Diabetes	768	8	2



(a) $q1(X)$ (b) $q2(X)$

Fig. 7. Evaluation graphs of clustering using $q1(X)$ and $q2(X)$

Figure 7 depicts how separation, cohesion, $q1(X)$ and $q2(X)$ evolve in two run of the post processing algorithm for the 9Diamonds dataset; in Figure 7 (a) $q1(X)$ was used as the fitness function and in Figure 7 (b) $q2(X)$ was used as the fitness function. As the number of clusters decreases, both separation and cohesion increase, a pattern we observed for many other data sets. $q1(X)$ goes down all the time in both runs, whereas the Silhouette coefficient has the tendency to go up for small k values. As expected, both runs have highest silhouette coefficient when k reaches 9. The visualization shown in Figure 9 and 10 provide the visual evidences with respect to high value of Silhouette coefficient in two datasets. For 9Diamonds dataset, the post-processing with $q2(X)$ discovers the natural clusters (Figure 8 (b)) while the post-processing with $q1(X)$ has some misclassification in the middle (Figure 8 (a)), but does better than K-means whose result was depicted in Figure 1 (b). For the Volcano dataset, K-means separates several natural chains. We captured a part of the Volcano dataset as shown in Figure 9 (a) to demonstrate this deficiency. In Figure 9 (b) the post-processing with $q1(X)$ obtains the perfect clustering: it captures all connected chains. In Figure 9 (c) the post-processing with $q2(X)$ fails to completely separate the upper chain from the left chain.


(a) $q1(X)$

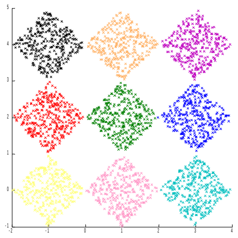
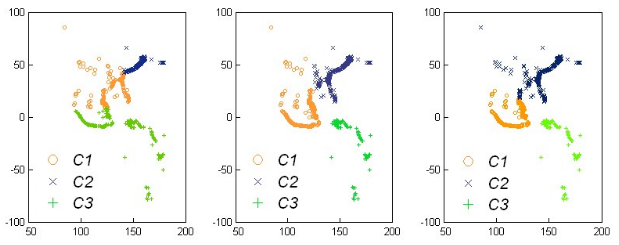

(b) $q2(X)$

Fig. 8. Post-processing results using $q1(X)$ and $q2(X)$ with k=9



(a) K-means (b) $q1(X)$ (c) $q2(X)$

Fig. 9. Comparison of experimental results of our post-processing with the K-means clustering algorithm

We conducted experiments of post-processing using $q1(X)$ and $q2(X)$ on 8 datasets. The post-processing using $q1(X)$ is using Separation/Cohesion and has a tunable parameter, δ, to adjust the influence of each to the overall quality function. In order to evaluate the impact of δ, we have tested on 8 datasets with 5 different δ values setting; 0, 0.5, 1, 1.5, and 2. The experimental results of using $\frac{Separation(X)^{\delta}}{Cohesion(X)^{(2-\delta)}}$ as the fitness function suggest that our post-processing algorithm performs best when δ is 1.5 and quite badly when δ is 2, suggesting that using solely separation as an evaluation function is not a good choice. We conducted a large number of experiments using $\frac{Separation(X)}{Cohesion(X)}$ as the fitness function which lead to significant improvements in cluster quality in 5 of the datasets tested. In the remaining of the chapter we assume that $q1(X)$ is always used with δ set to 1. Detailed results and visual representations of each experiment of $q1(X)$ with different δ values can be found at [21].

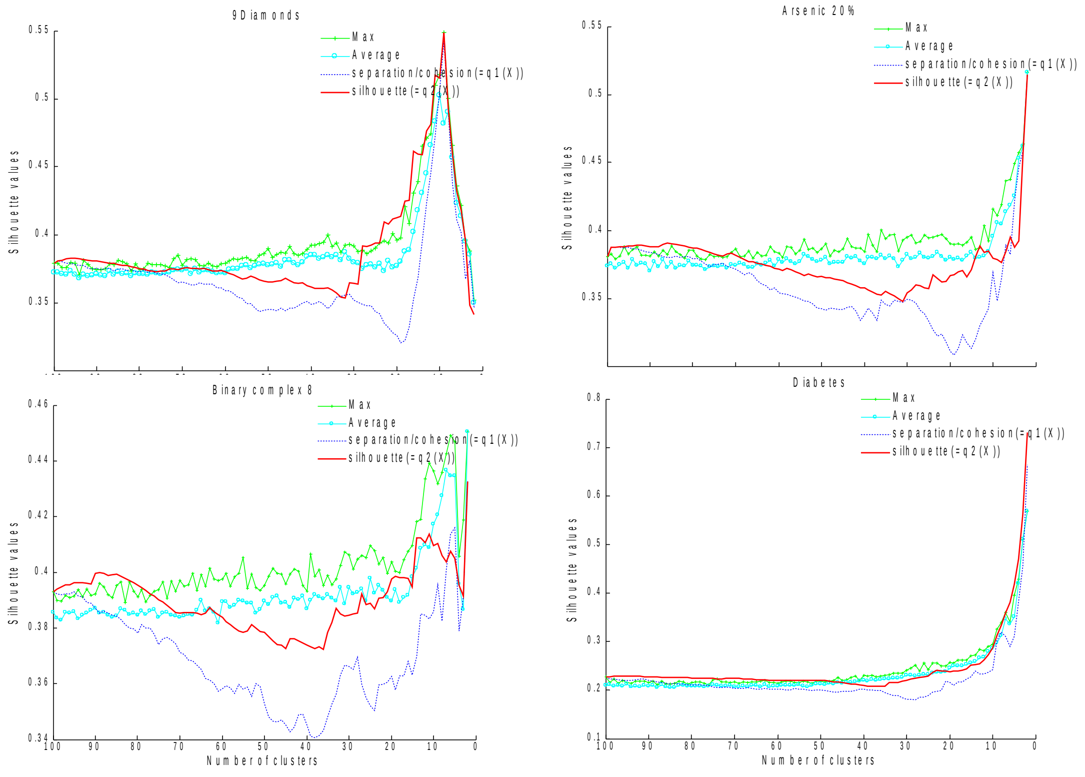
Figure 10 compares the maximum and average silhouette coefficients of K-means to the silhouette coefficient of post-processing technique that uses $q2(X)$ as the fitness function for some of the data sets tested; moreover, we also depict the $q1(X)$ values for the clusters that are created by the post-processing algorithm. In general, we observed four different patterns in the 8 datasets tested. In the first group that contains the Earthquakes 1%, Volcano, and Ionosphere datasets, the post-processing technique works well as it gives higher silhouette values than K-means in almost every iteration. In the second group (9Diamonds, Arsenic, and Diabetes dataset), our post-processing technique outperforms K-means at the beginning and at the end of the runs. For the Vehicle dataset, our post-processing technique slightly outperforms K-means until k=50 and then outperforms K-means significantly. Finally for Binary Complex 8 dataset, the post-processing works well only for the initial generations.

For Earthquakes 1%, Ionosphere, Vehicle, and Volcano dataset, our post-processing technique using $q2(X)$ obtains silhouette coefficient values higher than average silhouette coefficient values of K-means in most of iterations (except 1-2 iterations). Moreover it also obtains higher silhouette coefficient than the maximum silhouette values of K-means in most of iterations (except under 15 iterations). For other datasets such as 9Diamonds, Arsenic 20%, Binary Complex 8, and Diabetes, our post-processing outperforms K-means at the beginning of iterations and at the end of iterations. The complete experimental results are summarized in Figure 10 and TABLE II, respectively. In general our post-processing algorithms work efficiently in high-dimensional datasets such as Ionosphere, Diabetes, and Vehicle as discussed above.

TABLE II
ITERATIONS ON WHICH *q*2(*X*) OUTPERFORM K-MEANS CLUSTERING ALGORITHM

Dataset	*q*2(*X*) vs. K-Means Average	*q*2(*X*) vs. K-means maximum
9Diamonds	100-60, 28-6, 4	100-86, 28-15, 13-11
Arsenic 20%	100-64,14-11	100-72,13
Binary Complex 8	100-67, 63-61, 21-16, 14-11,3	100-77
Diabetes	100-45,24-21,10-2	100-48, 6-2
Earthquakes 1%	100-3	100-19, 9-5
Ionosphere	100-3	100-5
Vehicle	100-7, 5-2	100-8, 3-2
Volcano	100-5, 3	100-13,10-8

To demonstrate the effectiveness of our post-processing algorithm, we construct the graphs for the first 15 iterations as shown in Figure 11. The overall performance of the post-processing using *q*2(*X*) outperforms K-means clustering for all datasets in every iteration. In particular the silhouette coefficient of the post-processing is always higher than the maximum silhouette coefficient of K-means clustering algorithm.



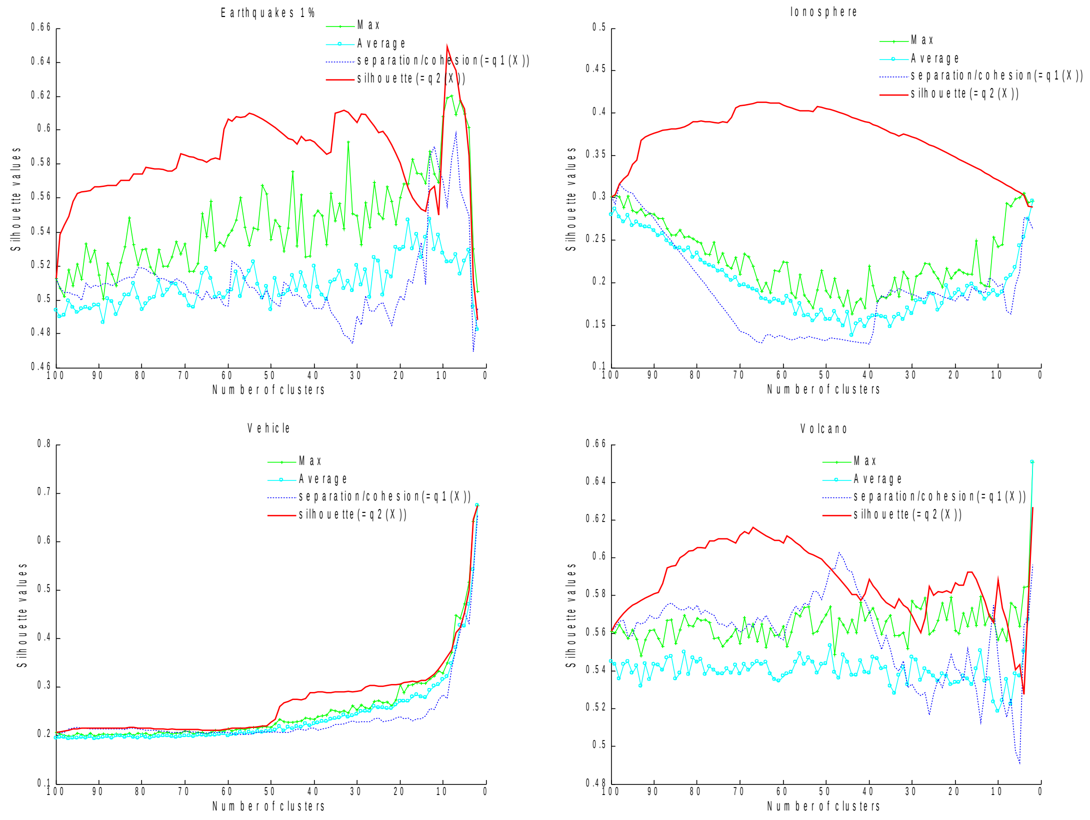
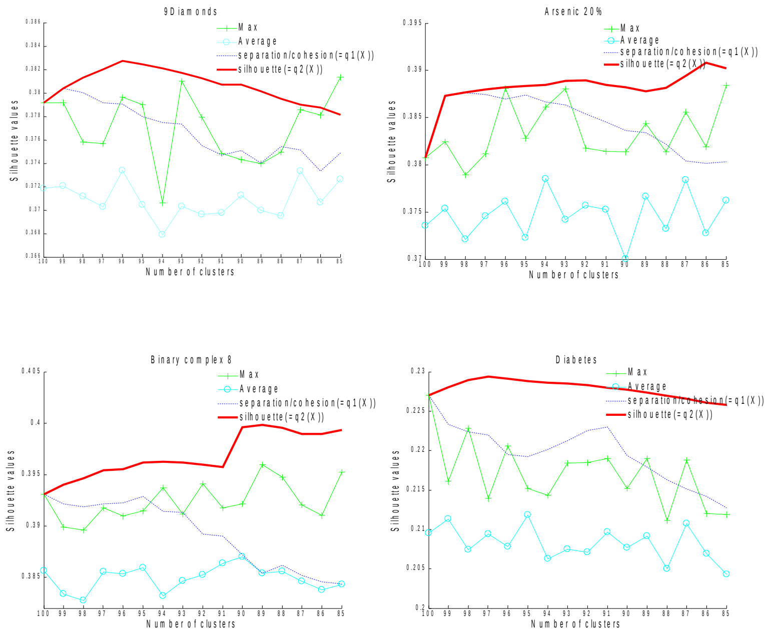


Fig. 10. Experimental results for the post-processing technique and K-means



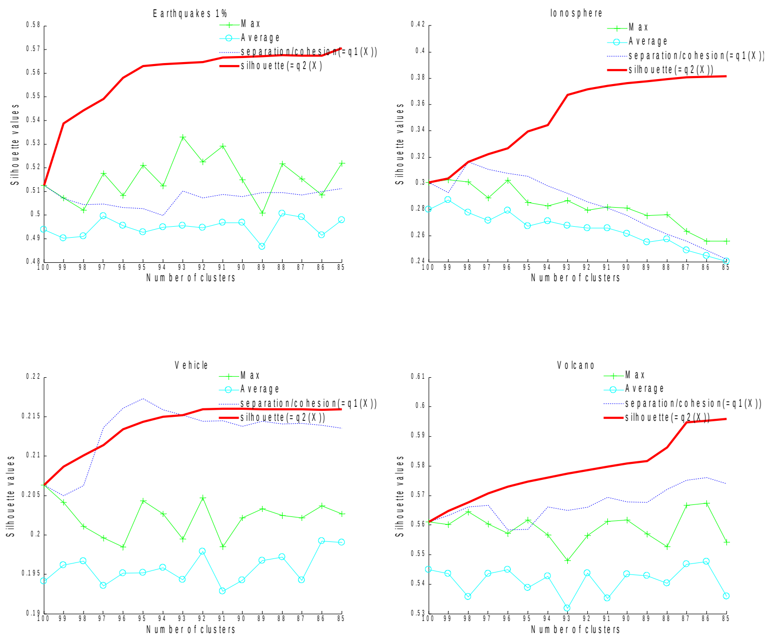


Fig. 11. Comparison of experimental results of our algorithm to K-means clustering algorithm for 15 iterations

TABLE III
AVERAGE SILHOUETTE VALUES OF MAX OF K-MEANS AND $q2(X)$ FOR THE FIRST 15 ITERATIONS (k=100-85)

Dataset	Average of max K-means	Average of $q2(X)$
9 Diamonds	0.37685	0.3808
Arsenic 20%	0.38308	0.38796
Binary complex 8	0.39224	0.39667
Diabetes	0.21748	0.22796
Earthquakes 1%	0.51505	0.55736
Ionosphere	0.28427	0.35076
Vehicle	0.2022	0.21374
Volcano	0.55951	0.57751

We also depicted the average of maximum silhouette coefficient of K-means and average silhouette coefficient of $q2(X)$ for the first 15 iterations in TABLE III. This table shows that our post-processing algorithm is much more efficient than K-means on datasets for the first 15 merges performed.

III. Post-Processing for Supervised Clustering

A. Motivation

Supervised clustering is applied on classified examples with the aim of producing clusters that have high probability density with respect to individual classes. Moreover, in supervised clustering, we also like to keep the number of clusters small, and examples are assigned to clusters using a notion of closeness with respect to a given distance function. Figure 12 illustrates the differences between a traditional and a supervised clustering. Let us assume that the black examples and the white examples represent objects belonging to two different classes. A traditional clustering algorithm would, very likely, identify the four clusters depicted in Figure 12 (a). This clustering would not be very attractive in the case of supervised clustering because cluster **A** has low purity of 50%, containing examples of two classes; moreover, the white examples are subdivided into two separate clusters **B** and **C**, although they are neighboring.

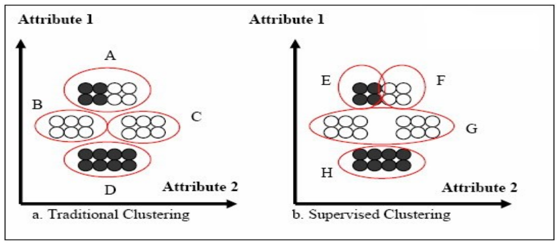


Fig. 12. Differences between Traditional Clustering and Supervised Clustering.

A supervised clustering algorithm that maximizes class purity, on the other hand, would split cluster **A** into two clusters **E** and **F** (Figure 12 (b)). Another characteristic of supervised clustering is that it tries to keep the number of clusters low. Consequently, clusters **B** and **C** would be merged into a single cluster without compromising class purity while reducing the number of clusters.

B.Evaluation Measures and Algorithms

In the experiments, we evaluate our post-processing technique using a reward-based fitness function. In particular, the quality $q(X)$ of a clustering X is computed as the sum of the rewards obtained for each cluster $c\in X$. Cluster rewards are computed as the product of interestingness of a cluster and the size of a cluster. More specifically, the evaluation function $q(X)$ is defined as follows:

$$q(X) = \sum_{c\in X} \mathrm{Re}\,ward(c) = \sum_{c\in X} \frac{i(c)\cdot(|c|)^{\beta}}{n^{\beta}}$$

with the interestingness $i(c)$ of a cluster c be defined as follows:

$$i(c) = \begin{cases} \left(\dfrac{(purity_Y(c) - hst)}{(1 - hst)}\right)^{\eta} & if \;\; purity_Y > hst \\ \left(\dfrac{(cst - purity_Y(c)}{cst}\right)^{\eta} & if \;\; purity_Y < cst \\ 0 & otherwise \end{cases}$$

where *hst a*nd *cst a*re hotspot and coolspot purity thresholds and $purity_Y(c)$ is the percentage of examples in cluster c that belong to the class of interest Y.

In general, we are interested in finding larger clusters if larger clusters are at least equally interesting than smaller clusters. Consequently, our evaluation scheme uses a parameter β with $\beta > 1$; that is, fitness increases nonlinearly with cluster-size dependent on the value of β, favoring clusters c with more objects. Selecting larger values for the parameter β usually results in a smaller number of clusters in the best clustering X. The measure of interestingness i relies on a class of interest Y, and assigns rewards to regions in which the distribution of class Y significantly deviates from the prior probability of class Y in the whole dataset.

The parameter η determines how quickly the reward function grows to maximum reward of 1. If η is set to 1 it grows linearly, if it is set to 2 a quadratic function would be used that grows significantly slower initially. In general, if we are interested in giving higher rewards to purer clusters, it is desirable to choose large values for η: e.g. η=8.

Let us assume a clustering *X* has to be evaluated with respect to a class of interest "Poor" that contains 1000 examples. Suppose that the generated clustering *X* subdivides the dataset into three clusters *c1, c2, c3* with the following characteristics. |*c1*| = 250, |*c2*| = 200, |*c3*| = 550; $purity_{poor}(c1)$ = 130/250, $purity_{poor}(c2)$ = 20/200, $purity_{poor}(c3)$ = 50/550. Moreover, the following parameters used in the fitness function are as follows: β= 1.1, η =1. A coolspot (*cst*=0.1) is defined as a cluster that contains less than 10% and a hotspot (*hst*=0.3) is a cluster that has more than 30% of instances of the class "Poor". Due to the settings clusters that contain between 10% and 30% instances of the class "Poor" do not receive any reward at all; therefore, no reward is given to cluster *c2* in the example. The remaining clusters received rewards because the distribution of class "Poor" in the cluster is significantly higher or lower than the corresponding threshold. Consequently, the reward for the first cluster c1 is 11/35 x $(250)^{1.1}$ since $purity_{poor}(c1)$ = 52% is greater than *hotspot* which is 30%, 11/35 is obtained by applying the function $purity_y(c)$, thus we get $purity_{poor}(c1)$ = ((0.52-0.3)/(1-0.3)) * 1 = 11/35. Rewards of other clusters are computed similarly and the following overall reward for *X* is obtained:

$$q_{Poor}(X) = \frac{\frac{11}{35} * 250^{1.1} + 0 + \frac{1}{11} * 550^{1.1}}{1000^{1.1}} = 0.115$$

C. Supervised Clustering with Evolutionary Computing (SCEC)

SCEC [9] is a representative-based supervised clustering algorithm that employs evolutionary computing to seek for the "optimal" set of representatives by evolving population of solutions over a fixed number of generations. The size of the population is fixed to a predetermined number when running SCEC. The initial generation is created randomly. The subsequent generations are generated by applying three different genetic operators to members of the current generation that are selected based on the principles of *survival of the fittest*. Figure 13 presents a flowchart of the SCEC algorithm, whose key features include:

1. *Chromosomal Representation*: A solution consists of a set of representatives that are a subset of the examples to be clustered.

2. *Genetic Operators*:

Mutation: replaces a representative by a non-representative.

Crossover: take 2 "parent" solutions and creates an offspring as follows:

A. Include all representatives that occur in both parents in the offspring

B. Include representatives that occur in a single parent with a probability of 50%.

Copy: Copy a member of the current generation into the next generation.

3. Selection: K-tournament selection is used to select solutions for generating the next generation through mutation, crossover, and copying. K-tournament randomly selects *K* solutions from the current population, and uses the solution with the highest *q*(*X*) value to be added to the mating pool for the breeding of the next generation.

4. Transformation of the Chromosomal Representation into Clusters and Evaluation:

A. Create clusters by assigning the remaining examples in the dataset to the closest representative.

B. Evaluate the so obtained clustering *X* using *q*.

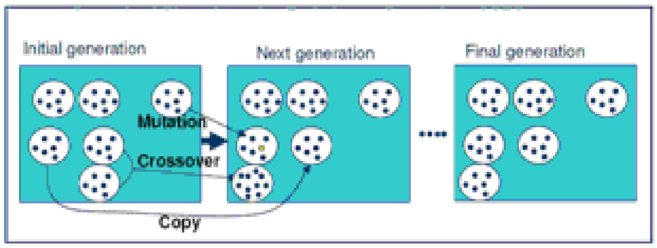


Fig. 13. Key features of the SCEC algorithm

D. Datasets and Experiments

In order to study the performance of the post-processing technique, we have conducted experiments on a benchmark consisting of five spatial datasets and one high dimensional dataset that were described in section 2. Two separate experiments were conducted. The first experiment compared the SCEC and the SCEC with post-processing. The second experiment compared SCEC with post-processing technique with three supervised clustering algorithms: SCAH, SCHG and SCMRG. SCAH is an agglomerative, hierarchical supervised clustering algorithm, SCHG is an agglomerative, hierarchical grid-based clustering method, and SCMRG is a divisive clustering algorithm that employs multi-resolution grids. More details descriptions of these algorithms and experimental results of the algorithms for 6 datasets can be found in [10].

The common parameters for SCEC of both experiments are listed in TABLE IV. In the first experiment, we set the fitness function parameters β=1.0001, and η=7. for SCEC to produce a large number of initial clusters. The post-processing algorithm used 4 different parameter settings: (β=1.0001, η=7), (β=1.01, η=6), (β=1.3, η=1), and (β=3, η=1). In the second experiment, we compared SCEC with post-processing, SCAH, SCHG, and SCMRG algorithms with 3 evaluation function parameter settings: (β=1.01, η=6), (β=1.3, η=1), and (β=3, η=1). After finishing the clustering for each algorithm, the value of evaluation function $q(X)$, the purity and the number of clusters are reported.

TABLE IV
PARAMETERS FOR SCEC

Initial Crossover rate	0
Initial mutation rate	0.95
Copy rate	0.05
Population size	400
Number of generations	1500
K for tournament	2

Figure 14 are the visualization of clustering results of Binary Complex 8, Binary Complex 9, Volcano (fraction), Earthquake (fraction) and Arsenic datasets. The clusters created by SCEC are shown on the left and the graphs; on the right are the results after applying our post-processing technique. These visualized results show that our post-processing technique is capable of merging neighboring clusters into larger continuous clusters. It is also capable of discovering arbitrary shape clusters.

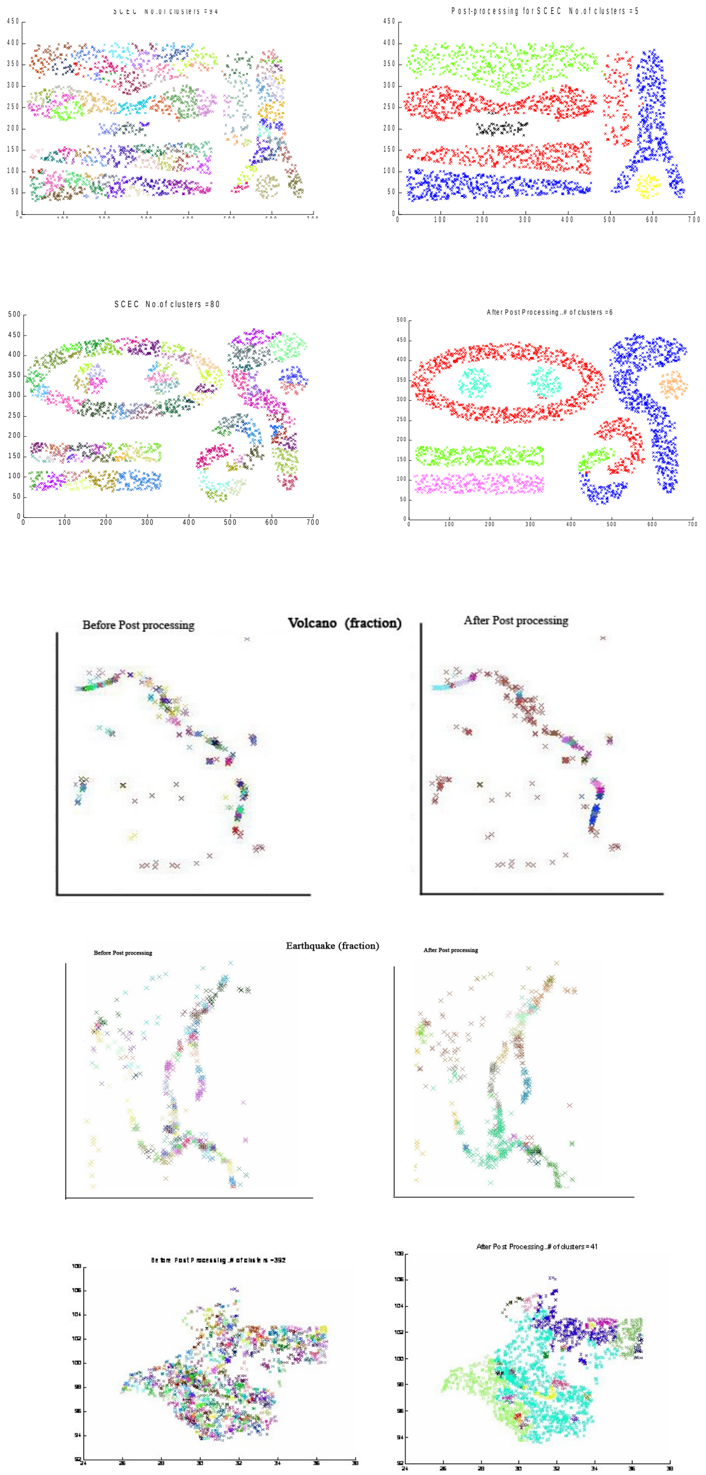


Fig. 14. (a) Before applying Post-Processing technique (left) and (b) After applying Post-Processing technique (right)

The experimental results listed in TABLE V indicate that post-processing technique considerably improves the performance of SCEC. We can draw a conclusion from the comparison between SCEC and SCEC with post-processing that the post-processing enhances or keeps the purity and the quality, and decreases the number of clusters on Binary Complex 8, Binary Complex 9, and Arsenic20% datasets for all parameter settings. For the Volcano, Earthquake 1% and Diabetes datasets, the post-processing improves the quality in most cases and reduces the number of clusters significantly when β is small. From a quantitative point of view, Table 5 shows that the post-processing technique improved average purity by 2.8%, average cluster quality by 13.3% and decreased the average number of clusters by 51.4%.

TABLE V
SCEC AND SCEC WITH POST-PROCESSING

Dataset	Parameters	SCEC			Post-processing		
		Purity	Quality	Clusters	Purity	Quality	Clusters
Binary Complex 8	β=1.0001, η=7	0.995	0.951	94	0.995	0.952	12
	β=1.01,η=6	0.990	0.901	90	0.995	0.940	12
	β=1.3, η=1	0.916	0.418	8	0.995	0.682	5
	β=3 , η=1	0.886	0.050	4	0.995	0.115	5
Binary Complex 9	β=1.0001, η=7	0.998	0.989	80	0.998	0.989	9
	β=1.01,η=6	0.998	0.937	98	0.998	0.973	9
	β=1.3, η=1	0.937	0.339	22	0.998	0.607	8
	β=3 , η=1	0.830	0.032	4	0.997	0.075	6
Volcano[3]	β=1.0001, η=7	0.790	0.332	402	0.780	0.332	230
	β=1.01,η=6	0.780	0.322	402	0.780	0.316	230
	β=1.3, η=1	0.789	0.068	372	0.787	0.110	176
	β=3 , η=1	0.607	4.65E-4	7	0.786	0.002	117
Earthquake1 % [1]	β=1.0001, η=7	0.903	0.610	400	0.884	0.610	147
	β=1.01,η=6	0.895	0.575	404	0.884	0.599	147
	β=1.3, η=1	0.846	0.272	5	0.858	0.412	76
	β=3 , η=1	0.846	0.061	4	0.842	0.141	34
Arsenic 20%[1]	β=1.0001, η=7	0.836	0.402	392	0.836	0.402	172
	β=1.01,η=6	0.834	0.391	396	0.836	0.392	72
	β=1.3, η=1	0.779	0.105	11	0.808	0.253	90
	β=3 , η=1	0.774	0.021	6	0.779	0.065	41
Diabetes	β=1.0001, η=7	0.770	0.315	94	0.742	0.315	17
	β=1.01,η=6	0.790	0.289	94	0.742	0.310	17
	β=1.3, η=1	0.740	0.228	5	0.753	0.208	9
	β=3 , η=1	0.726	0.050	2	0.764	0.018	9
Average		0.844	0.361	141.5	0.868	0.409	68.75

We further compare the SCEC with post-processing to SCAH, SCHG, and SCMRG. The same set of parameter settings of β and η were applied to each algorithm with Binary Complex 8, Binary Complex 9, Volcano, and Earthquake datasets. The experimental results of the three supervised clustering algorithms have been reported in [10]. Considering the quality of the clustering, our representative-based clustering algorithm, SCEC with post-processing, did not perform as well as other algorithms for β=1.01. SCHG had the highest quality value on Complex8 dataset and SCAH performed best on the other 5 datasets. However, if the evaluation

[3] The initial number of clusters for SCEC is 400

function prefers the size of the cluster to be medium size or big, SCEC outperforms all other algorithms tested. For β=1.3, SCEC with post-processing outperformed all other supervised clustering algorithms on all datasets except the volcano dataset in terms of quality. For β=3, SCEC with post-processing has the best quality of clustering among all supervised clustering algorithm on all datasets.

IV. Related Work

In many domains, such as hot spot detection, region discovery and spatial data mining, it important to find regions that have arbitrary shapes. There is significant amount of research centering on this topic. Jiang [1] proposes spatial clustering techniques for generalization processes in GIS. The paper claims that hierarchical clustering accompanied by tree-like diagrams provides a powerful tool for visualizing cluster hierarchies at different levels of detail. The paper also stresses the limitations of K-means clustering that cannot effectively detect network skeletons and claims that hierarchical clustering algorithms is suitable for this task. Heckel [11] proposes a hierarchical classification and visualization technique that utilized the hierarchical clustering and classical techniques in statistics and computer visualization. It recursively splits the clusters in such a way that an eigenvector is used as the center of split clusters. It then applies a local reclassification scheme based on the Gabriel graph to improve the quality of the classification. Anders [12] developed an unsupervised graph-based clustering algorithm, called *Hierarchical Parameter-free Graph CLustering* (HPGCL) for spatial data analysis. Various proximity graphs are used to define coarse-to-fine hierarchical segmentation of a data.

Density-based clustering methods [13, 14, 15, and 16] are considered to be efficient in discovering dense arbitrary shaped cluster, such as 9Diamonds. Sander et al. generalize DBSCAN to GDBSCAN: that can cluster point spatial objects as well as complex spatial objects. In real world spatial and biological applications, the information in datasets are sometimes vague and uncertain. Kriegel and Pfeifle propose a fuzzy distance functions to measure the similarity between fuzzy objects and enhance the DBSCAN algorithm to deal with the fuzzy objects. Hinneburg and Keim proposed the DenClue algorithm, a clustering method based on the density estimation techniques. Although density-based algorithm are effective in detecting dense arbitrary shape cluster, they usually face problems with parameters tuning, with datasets having low or varying density, with varying size cluster sizes, and are not suitable for high dimensional datasets.

There is significant research centering on hybrid clustering. Lin and Zhong [17 and 18] propose a hybrid clustering algorithm that combines representative-based clustering and hierarchical clustering methods which are similar to our approach. However they employ the different merging criteria and merge clusters greedily by considering only single pair of merging clusters. On the other hand, our approach uses Gabriel graph to construct the clusters neighboring. The benefit is that the post-processing can consider more merging candidates and consequently maximizes the fitness function for merging clusters. We also investigate other researches in hybrid clustering approaches that are also different from our approach. Surdeanu [19] extracts the top-ranking initial clusters candidates from a hierarchical agglomerative clustering algorithm and then input to Expectation Maximization (EM) algorithm. Fern [20] uses K-means and random subsampling to generate cluster ensembles and merge them by using bipartite graph partitioning.

V. Conclusion

This paper proposes a novel clustering approach that approximates arbitrary-shape clusters through unions of small spherical clusters. Gabriel graphs are used to identify neighboring clusters. Using proximity graphs increases the number of merge candidates considerably over traditional agglomerative algorithms that only consider "closest" clusters for merging, resulting in clusters of higher quality. The proposed post-processing technique is quite general and can be used with any representative-based clustering algorithm, proximity graph, and internal cluster evaluation measure. Furthermore, it is scalable since the merging is conducted in an incremental fashion, and the running of the expensive, agglomerative algorithms is limited to less than a few hundred iterations. Moreover, as a by-product, the paper also contributes a formula to compute cluster separation and cohesion incrementally.

Our proposed post processing technique has also some similarity agglomerative grid-based clustering algorithms; both approaches employ micro-clusters which are grid-cells in their approach and convex polygons in our approach and greedily merge neighboring clusters. However, our approach has much more general by supporting more shapes and by allowing convex polygons of different sizes. On the other hand, for a given grid structure it easy to determine which clusters are neighboring, which is not the case in our approach.

We conducted two sets of experiments to evaluate our post-processing technique: one for traditional clustering algorithms and another for supervised clustering algorithms. The experimental results of using silhouette coefficients as the fitness function showed that our post-processing technique always obtains higher silhouette coefficients in the first 15 iterations than the maximum silhouette value observed in 10 runs of K-means for all data sets. After 15 iterations, the results of the post-processing technique continued to improve for five of the eight data sets tested. We also presented visual evidence that our post-processing approach discovers the "natural" clusters in 9Diamonds and Earthquakes datasets which K-means mostly fails to discover. We also observed a gap between the average silhouette coefficient and the maximum silhouette coefficient for different runs of K-means. This suggests that K-means should be run multiple times (i.e. 50 times) in order to get reasonable clustering results.

The experimental results of applying our post-processing technique to the results of a Supervised Clustering with Evolutionary Computing algorithm (SCEC) showed significant improvement over the stand-alone version of SCEC. For the benchmark tested, the post-processing technique improved purity by 2.8% and cluster quality by 13.3% and decreased the number of clusters by 51.4%. We also compared these results with the performance of other supervised clustering algorithms. When the objective is to obtain medium or large size clusters, the post-processing technique outperformed the other supervised clustering algorithms significantly. In general, the post processing technique showed quite significant improvement for supervised clustering; our explanation for this fact is that the class-label provides a quite strong feedback (that usually is not found in traditional clustering) that makes it easier for the clustering algorithm to make correct merging decisions.

In future research, we plan to investigate internal cluster evaluation measures that are more suitable for evaluating clusters with non-spherical shapes. Furthermore, as mentioned in Section 2, the time complexity of the current post-processing implementation is $O(n^2)$ by using *q1(x)* or *q2(x)* function. We plan to investigate approximate Gabriel Graph algorithm and to derive formulas for computing Silhouette coefficients incrementally to reduce this time complexity to $O(k^2)$.

References

[1] B. Jiang, "Spatial Clustering for Mining Knowledge in Support of Generalization Processes in GIS", in *ICA Workshop on Generalisation and Multiple representation, Leicester*, 20-21 August 2004.

[2] K. Gabriel and R. Sokal, "A New Statistical Approach to Geographic Variation Analysis", *Systematic Zoology,* Vol. 18, No. 3, September 1969, pp. 259-278.

[3] G. Toussaint "The relative neighborhood graph of a finite planar set", *Pattern Recognition*, Vol. 12, 1980, pp. 261-268.

[4] D. Kirkpatrick, "A note on Delaunay and optimal triangulations", *Information Processing Letters 10*, 1980, pp. 127-128.

[5] A. Okabe, B. Boots, and K. Sugihara, "Spatial Tessellations: Concepts and Applications of Voronoi Diagrams", *Wiley*, New York, 1992.

[6] B. Bhattacharya , R. Poulsen, and G. Toussaint, "Application of Proximity Graphs to Editing Nearest Neighbor Decision Rule", in *International Symposium on Information Theory*, Santa Monica, 1981.

[7] M. Tan, M. Steinbach, and V. Kumar, "Introduction to Data Mining", *Addison Wesley*, 2005.

[8] P.J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis", *Journal of Computational and Applied Mathematics 20*, Elsevier Science Publishers B.V., North-Holland, 1987, pp. 53-65.

[9] C. Eick, N. Zeidat, Z. Zhao, “Supervised Clustering --- Algorithms and Benefits”, in *Proceedings of International Conference on Tools with AI (ICTAI), Boca Raton, Florida*, November 2004.

[10] C. Eick, B. Vaezian, D. Jiang, and J. Wang, “Discovery of Interesting Regions in Spatial Datasets Using Supervised Clustering”, in *Proceedings of PKKD Conference*, Berlin, Germany, September 2006.

[11] B. Heckel, and B. Hamann, “Visualization of Cluster Hierarchies”, in *Proceeding of SPIE Volume 3298, SPIE, The International Society for Optical Engineering,* Bellingham, Wasington, January 1998, pp. 162-171.

[12] K.H. Anders, “A Hierarchical Graph-Clustering Approach to find Groups of Objects”, (Technical Paper), in *ICA Commission on Map Generalization, Fifth Workshop on Progress in Automated Map Generalization*, IGN, Paris, 28-30 April 2003.

[13] Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu, “Density-Based Spatial Clustering of Applications with Noise”, in *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining*, 1996.

[14] J. Sander, M. Ester, H.P. Kriegel, and X. Xu “Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and its Applications*”, Data Mining and Knowledge Discovery , An International Journal* 2(2): 169-194, June 1998. Kluwer Academic Publishers, Norwell, MA.

[15] H.P. Kriegel and M. Pfeifle, “Density-Based Clustering of Uncertain Data”, in *Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, Chicago, Illinois, USA, 2005, pp. 672-677.

[16] A. Hinneburg, and D. Keim, “An Efficient Approach to Clustering in Large Multimedia Databases with Noise”, in *Proceedings of KDD Conference*, 1998.

[17] C. Lin and M. Chen, “A Robust and Efficient Clustering Algorithm based on Cohesion Self-Merging”, in *Proceeding of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, Edmonton, Alberta, Canada, 2002, pp. 582-587.

[18] S. Zhong, and J. Ghosh, “A unified framework for model-based clustering”, *The Journal of Machine Learning Research Vol.4*, MIT Press Cambridge, MA, USA, 2003, pp. 1001-1037.

[19] M. Surdeanu, J. Turmo, and A. Ageno, “A hybrid unsupervised approach for document clustering”, in*, Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, Chicago, Illinois, USA, 2005, pp. 685-690.

[20] X. Z. Fern and C. E. Brodley, “Solving clustering ensemble problems by Bipartite graph partitioning”, in *ACM International Conference Proceeding Series; Vol. 69, Proceedings of the twenty-first international conference on Machine learning*, Banff, Alberta, Canada, 2004, pp. 36.

[21] J. Y. Choo, “Using Proximity Graphs to Enhance Representative-based Clustering Algorithms”, Master Thesis, Department of Computer Science, University of Houston, TX, February 2007.

Appendix 1: Incremental Update of the Investigated Cluster Evaluation Measure.

A. Incremental Update of Cohesion and Separation.

In this section, a detailed description about the technique that can be used for incrementally updating the *q1(x)* function will be provided:

Let us suppose to use two matrices *A* and *B* with dimensions *k* x *k* (where *k* is the number of clusters and thus of nodes of the Gabriel Graph).

Let *A* be: $A_{i,j}=\left(\sum_{o_l\in C_i}\sum_{o_m\in C_j} d(o_l,o_m)\right)$ and *B* be $B_{i,j}=\left(\sum_{o_l\in C_i}\sum_{o_m\in C_j} 1\right)$ where o_l and o_m are objects of the dataset and C_i and C_j are clusters.

It is clear that if $i\neq j$ then $A_{i,j}$ is the sum of the distances of all the pairs of objects between clusters C_i and C_j. If $i=j$ then $A_{i,i}$ is the sum of the distances of all the pairs of objects inside the *i*-th cluster. $B_{i,j}$ stores the number of pairs of objects between the *i*-th and *j*-th clusters.

The time complexity for the computation of matrices *A* and *B* is $O(n^2)$, because we need to sum all the possible pairs of objects in our dataset (*n* is the number of objects).

Using *A* and *B,* it is simple to compute cohesion and separation, in fact the values:

$$Coh(0)=\frac{Sumintra(0)}{Intra(0)}=\frac{\sum_{1\le i\le k} A_{i,i}}{\sum_{1\le i\le k} B_{i,i}} \text{ and } Sep(0)=\frac{Suminter(0)}{Inter(0)}=\frac{\sum_{1\le i\le k}\sum_{1\le j\neq i\le k} A_{i,j}}{\sum_{1\le i\le k}\sum_{1\le j\neq i\le k} B_{i,j}}$$

are cohesion and separation at the step number 0, that is before starting the merging process. Obviously, the complexity is $O(k^2)$.

During the merging process, it is very easy to update cohesion and separation values, in fact, at each generic step p:

$$Sumintra(p)=Sumintra(p-1)+A_{i,j},$$
$$Suminter(p)=Suminter(p-1)-A_{i,j},$$
$$Intra(p)=Intra(p-1)+B_{i,j},$$
$$Inter(p)=Inter(p-1)-B_{i,j}$$

if we are merging respectively cluster *i*-th with cluster *j*-th. And cohesion and separation are computed as follows:

$$Coh(p)=\frac{Sumintra(p)}{Intra(p)} \quad Sep(p)=\frac{Suminter(p)}{Inter(p)}$$ (time complexity *O*(1)).

This operation has to be performed once for each edge of the graph. Since the graph is a Gabriel Graph (thus is a connected planar graph), the number of edges is $k-1\le e\le 3\mathrm{k}-6$. This means that the number of edges *e* is linear with respect to the number of nodes, that is $e\simeq O(k)$. Since at the generic step *p*, the number of remaining nodes is *k-p*, such an operation has to be performed $O(k-p)$ times for each step *p*.

Once computed cohesion and separation, we have to update matrices *A* and *B*, in order to manage the new situation. Since cluster C_i and cluster C_j have been merged together and now they have become a single cluster $C_{(i+j)}$, we have to compute:

$$A_{(i+j),(i+j)}=A_{i,i}+A_{j,j}+A_{i,j}$$

and, for each other cluster *l*, we have to compute

$$A_{(i+j),l}=A_{i,l}+A_{j,l}.$$

The same update has to be performed on *B*. This operation takes time equal to $O(k-p)$.

In conclusion, we have to perform the following operations:

- Compute the matrices *A* and *B*: $O(n^2)$.
- Compute for the first time Cohesion, Separation, Sumintra, Suminter, etc. : $O(k^2)$.
- Execute $O(k)$ merging steps (one for each cluster), where each step has a time complexity equal to $O(k-p)+O(k-p)$ where *p* is the number of the current step.

In the end, we obtain a total complexity equal to $O(n^2+k^2)\simeq O(n^2)$ because *n* >> *k.*

B. Incremental Update of Silhouette Function.

In this section, a detailed description about the technique that can be used for incrementally updating the *q2(x)* function will be provided:
The silhouette function is computed using the following values:

$$a_i = \min_{m\neq k}\left(\frac{1}{|C_m|}\sum_{o_j\in C_m} d_{i,j}\right) \text{ (1) and } b_i = \left(\frac{1}{|C_k|}\sum_{o_j\in C_k} d_{i,j}\right) \text{ (2)}$$

for each object $o_i \in C_k$, where $d_{i,j}$ is the euclidean distance between objects *i-th* and *j-th.*

Thus, at each merging step *p*, for each object o_i, we should compute each possible $a_{i,m} = \left(\frac{1}{|C_m|}\sum_{o_j\in C_m} d_{i,j}\right)$ (3) and then find out the minimum amongst them.
Since the procedure merges only two cluster at a time, not all the value a_i and b_i will be changed at each computational step *p*. If we analyze deeper the way the silhouette values change, we are able to detect two main cases, each one can be decomposed in two sub-cases:

1. Object o_i belongs to a cluster that has just been merged with another cluster.
2. Object o_i does not belong to a cluster that has been merged with another cluster during the computational step *p*.

Case 1:
This case is applied when the object $o_i \in C_k$ that we are taking into account belongs to a cluster which has just been merged by the merging procedure. This case must be divided in two sub-cases:

a. in the first sub-case , the cluster C_k has not been merged with the cluster that minimized the eq. (1) at the step *p*-1. Since the value of a_i is computed only using objects belonging to external clusters, the cluster minimizing the eq. (1) remain the same, therefore, in this case $a_i(p)=a_i(p-1)$, where *p* is the current step of computation.
b. In the second sub-case, the cluster C_k has been merged with the cluster that minimized the eq. (1) at the step *p*-1. The value of a_i is computed using objects belonging to external clusters, but in this case the cluster minimizing the eq. (1) cannot remain the same, because it has just been merged with the cluster C_k. This means that at step *p*, the cluster index that minimizes eq. (1) is the second best value for the eq. (1) at the previous step:

$a_i(p) = second\, best(a_i(p-1))$.

Case 2:
This case is applied when the object $o_i \in C_k$ that we are taking into account doesn't belong to the two clusters which have just been merged together by the merging procedure. Even this case must be divided in two sub-cases similar to case 1:

a. Let C_l and C_r be the two clusters that have just been merged and let q the index of the cluster that minimized eq. (1) at step p-1. In this case, the index of the cluster that minimizes the eq. (1) at step p must be chosen between index q (the previous best) and the index of the new cluster (l+r), thus:
$a_i(p)=min\left(a_i(p-1), a_{i,l+r}(p)\right)$ but it's easy to prove that:
$a_i(p)=min\left(a_i(p-1), a_{i,l+r}(p)\right)=a_i(p-1)$.

b. Let C_l and C_r be the two clusters that have just been merged and let l or r the index of the cluster that minimized eq. (1) at step p-1. In this case, the cluster that minimized the eq. (1) at the previous step does not exist anymore, thus the index that minimizes the equation must be chosen between the previous second best value and the new cluster $C_{(l+r)}$:
$a_i(p)=min\left(second\,best\,a_i(p-1), a_{i,l+r}(p)\right)$.

It is clear that this is the only case we have a computational complexity that is different from O(1). Thus, in order to compute the complexity, we have to estimate the number of times this case occurs.

Now it remains to compute incrementally the value of b_i at each step p. In order to maintain such a value updated, without recomputing it every time, we can use the same method as in the incremental update of cohesion and separation functions. Thus, we must compute two matrices n x k, where n is the number of objects and k the number of clusters such as:
$A_{i,j}=\left(\sum_{o_m \in C_j} d(o_i, o_m)\right)$ and
$B_{i,j}=\left(\sum_{o_m \in C_j} 1\right)$.
If at step p, clusters C_l and C_r have been merged together, we can recompute all the new values of b_i (for each $i \in C_{l+r}$), simply by computing $\frac{A_{i,l}+A_{i,r}}{B_{i,l}+B_{i,r}}$.
In the end, like in the appendix A, an update of the matrices has to be performed.
If we analyze the computational complexity, it is simple to see that for the computation of a_i during the post-processing is equal to $O(k^2 \cdot n \cdot m)$, where k is the number of clusters, n the number of objects in the dataset and m the computational cost of the case 2b at each step p. For the computation of b_i, the greatest cost is due to the initialization of matrices A and B, which takes time equal to $O(n^2)$ because all the distances between all the possible couples of objects must be computed.
Summarizing, the computational complexity of post-process technique is $O(k^2 \cdot n \cdot m + n^2) \simeq O(n^2)$ because n >> k and m.
At this time we are exploring the possibility to compute *q1(x)* and *q2(x)* locally, in order to reduce the intrinsic computational complexity introduced by these two fitness functions.