MOSAIC: Agglomerative Clustering with Gabriel Graphs

Rachsuda Jiamthapthaksin Computer Science Department, University of Houston, USA
Jiyeon Choo Computer Science Department, University of Houston, USA
Chun-sheng Chen Computer Science Department, University of Houston, USA
Oner Ulvi Celepcikay Computer Science Department, University of Houston, USA
Christian Giusti Department of Mathematics and Computer Science, University of Udine, Italy
Christoph F. Eick Computer Science Department, University of Houston, USA

Abstract. *Representative-based clustering algorithms are quite popular due to their relative high speed and because of their sound theoretical foundation. On the other hand, the clusters they can obtain are limited to convex shapes and clustering results are also highly sensitive to initializations. This paper proposes post-processing techniques to alleviate this problem. In particular, a novel agglomerative clustering algorithm called MOSAIC is proposed which greedily merges neighboring clusters maximizing an externally given fitness function. MOSAIC uses Gabriel graphs to determine which clusters are neighboring and approximates non-convex shapes as the unions of small clusters that have been computed using a representative-based clustering algorithm. We evaluate MOSAIC for traditional unsupervised clustering with k-means and DBSCAN, and also for supervised clustering. The experimental results show that the proposed post-processing techniques lead to clusters of higher quality compared to running a representative clustering algorithm stand-alone. Moreover, given a suitable fitness function, MOSAIC is able to detect arbitrary shape clusters which are comparable to the ones generated by DBSCAN. In addition, MOSAIC is capable of dealing with high dimensional data. We also claim that MOSAIC can be employed as an effective post-processing clustering algorithm to further improve the quality of clustering.*

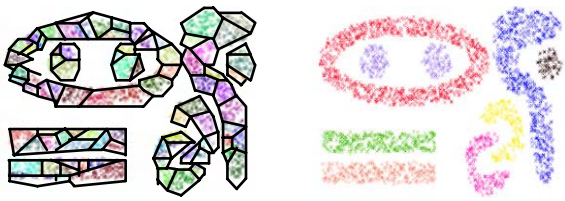
Keywords: *Post-processing, hybrid clustering, finding clusters of arbitrary shape, agglomerative clustering, using proximity graphs for clustering.*

1 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 $O(k \cdot t \cdot n)$, where *n* is the number of objects, *k* is the number of clusters, and *t* is the number of iterations. The clusters generated are always contiguous. However, when using *k*-means the proper number of clusters *k* has to be known in advance, and *k*-means is

very sensitive to initializations and outliers. Another problem of the *k*-means clustering algorithm is that it cannot obtain clusters having non-convex shapes [1]: the shapes that can be obtained by representative-based clustering algorithms are limited to convex polygons.

In theory, agglomerative hierarchical clustering (AHC) [2] 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 merging candidates and therefore often misses high quality solutions. Moreover, its time complexity is $O(n^2)$ or worse. Finally, many variations of AHC obtain non-contiguous clusters. [3]



(a) Input *(b) Output*

Fig. 1. An illustration of MOSAIC's approach

This paper proposes a hybrid clustering technique that combines representative-based with agglomerative clustering trying to maximize the strong points of each approach. A novel agglomerative clustering algorithm called MOSAIC is proposed, which greedily merges neighboring clusters maximizing a given fitness function and whose implementation uses Gabriel graphs [4] to determine which clusters are neighboring. Non-convex shapes are approximated as the union of small convex clusters that have been obtained by running a representative-based clustering algorithm, as illustrated in Fig. 1. Creating mosaics in art is the process of assembling small pieces to get a sophisticated design. Similarly, the proposed MOSAIC algorithm pieces convex polygons together to obtain better clusters.

1. Run a representative-based clustering algorithm to create a large number of clusters.
2. Read the representatives of the obtained clusters C_i.
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:
      Merge-Candidate(C', C) $\Leftrightarrow$
         Merge-Candidate(C_i, C) $\vee$ Merge-Candidate(C_j, C)
   END
RETURN the best clustering X found.

Fig. 2. Pseudo code for MOSAIC

Relying on proximity graphs the MOSAIC conducts a much wider search which leads in clusters of higher quality. Moreover, the expensive, agglomerative clustering algorithm is only run for usually less than 1,000 iterations; therefore, the impact of its high 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. Fig. 2 gives the pseudo code of the proposed MOSAIC algorithm.

In summary, MOSAIC merges pairs of neighboring clusters maximizing an externally given fitness function *q*, and this process is continued until only one cluster is left. Finally, the best clustering is determined and returned. Using cluster representatives obtained from 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 this proximity graph. When clusters are merged, this merge-candidate relation is updated incrementally without any need to regenerate proximity graphs.

The main contributions of this work are:

- It introduces a hybrid algorithm that combines strong features of representative-based clustering and agglomerative clustering.
- The algorithm provides flexibility by enabling to plug-in any fitness functions and is not restricted to any specific cluster evaluation measure.
- The algorithm conducts a much wider search, compared to traditional agglomerative clustering algorithms, by considering neighboring clusters from a proximity graph as merge candidates.
- It is beneficial to apply MOSAIC as a post-processing to identify clusters with complex shapes, for instance a cluster residing in another cluster.

The organization of our paper is as follows: Section 2 describes MOSAIC in more detail, and explains cluster evaluation measures used in traditional and supervised clustering. The time complexity of MOSAIC is also analyzed. Section 3 reports the results of an experimental evaluation of MOSAIC on traditional clustering and supervised clustering problems. Related work is reviewed in Section 4, and a conclusion is given in Section 5 respectively.

2 POST-PROCESSING WITH MOSAIC

This section discusses MOSAIC in more detail. First, proximity graphs are introduced and their role in agglomerative clustering is discussed. Next, cluster evaluation measures will be discussed that will serve as fitness functions in the experimental evaluation. Finally, MOSAIC's complexity is discussed

2.1 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 [5], with Delaunay graphs [6] (DG) being the most popular ones. The Delaunay graph generated from a set of cluster representatives tells us which clusters of a representative-based clustering are neighboring and the shapes of these clusters can be computed by using Voronoi cells, the dual to Delaunay graphs.

Delaunay triangulation (DT) [7] 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 computational complexity of $O(n^{d/2})$ (when $d > 2$), where d is the number of dimensions of a data set and n is the number of graph vertices. Therefore, our implementation of MOSAIC uses another proximity graph called Gabriel graphs (GG) [4] instead, which is a sub-graph of the DG. Two points are said to be Gabriel neighbors if their diametric sphere does not contain any other points. 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 [9]. Constructing GG has a time complexity of $O(dn^3)$ but faster approximate algorithms (with time complexity $O(dn^2)$) to construct GG

exist [8]. The pseudo code of an algorithm that constructs the GG for a given set of objects is given in Fig. 3.

MOSAIC 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. The illustration of the Gabriel graph construction in MOSAIC is shown in Fig. 4 in which clusters that have been obtained by a representative algorithm are visualized using black polygons, cluster representatives are depicted as red dots, and neighboring (e.g. Merge-Candidates) are depicted as yellow segments.

Let $R = \{r_1, r_2, ..., r_n\}$, be a set of cluster representatives.

FOR each pair of representatives $(r_i, r_j) \in R^2$:

IF for each representative r_k, the following inequality holds

$$d(r_i, r_j) \leq \sqrt{d^2(r_i, r_k) + d^2(r_j, r_k)}$$

where $k \neq i, j$ and $r_k \in R$,

THEN r_i and r_j are neighboring.

$d(r_i, r_j)$ denotes the distance of representatives r_i and r_j.

Fig. 3. Pseudo code for constructing Gabriel graphs.

Fig. 4. Gabriel graph for clusters generated by a representative-based clustering algorithm

2.2 Cluster Evaluation Measures for Traditional Clustering

Many cluster evaluation measures have been proposed in the literature [2]. In this paper, we use Cohesion, Separation, and Silhouettes [10] which are evaluation measures that have been widely used to assess cluster quality. Cohesion measures the tightness of a cluster while Separation measures how well-separated a cluster is from other clusters. The formula of both evaluation measures are 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 $C_i \cap C_j = \varnothing$ 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,

$$Cohesion(X) = \frac{Sumintra(X)}{Intra(X)}$$

$$Separation(X) = \frac{Suminter(X)}{Inter(X)}$$

where,

$$Sumintra(X) = \sum_{i<j,\, o_i \text{ and } o_j \text{ belong to the same cluster}} d_{i,j}$$

$$Suminter(X) = \sum_{i<j,\, o_i \text{ and } o_j \text{ belong to different clusters}} d_{i,j}$$

On the other hand, Silhouettes is a popular cluster evaluation technique that takes both cohesion and separation into consideration. The definition of a Silhouette Coefficient s_i for each object o_i belonging to cluster C_k is as follows:

$$s_i = \frac{(a_i - b_i)}{\max(a_i, b_i)} \qquad \textbf{(1)}$$

where,

$$a_i = \min_m \left(\frac{1}{|C_m|} \sum_{o_j \in C_m} d_{ij} \right), \quad m \neq k \quad \text{and} \qquad \textbf{(2)}$$

$$b_i = \frac{1}{|C_k|} \sum_{o_j \in C_k} d_{ij} \,. \qquad \textbf{(3)}$$

In the formula (1), 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 Silhouettes over whole dataset. The fitness function $q(X)$ is defined as follows:

$$q(X) = \frac{1}{n} \sum_{i=1}^{n} s_i \qquad \textbf{(4)}$$

where n is the number of objects in a dataset and X is the dataset partition. We simply call the fitness function, Silhouettes through the rest of the paper.

2.3 A Cluster Evaluation Measure for Supervised Clustering

Due to the fact that MOSAIC supports plug-in fitness functions, it can be used as a supervised clustering algorithm. In general, supervised clustering is applied to 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. Fig. 5 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 Fig. 5 (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.

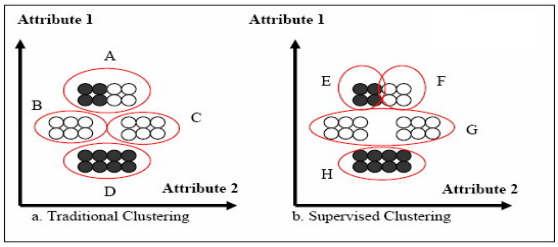


Fig. 5. Differences between traditional clustering and supervised clustering.

A supervised clustering algorithm that maximizes class purity, on the other hand, would split cluster A into two clusters E and F (Fig. 5 (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.

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:

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

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 and cst are 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.

To clarify the reward-based fitness function, we generate an example to illustrate the calculation as follow: Let us assume a clustering X has to be evaluated with respect to a class of interest "Poor" that contains 1,000 examples. Suppose that the generated clustering X subdivides the dataset into three clusters c_1, c_2, c_3 with the following characteristics. $|c_1| = 250$, $|c_2| = 200$, $|c_3| = 550$; $purity_{Poor}(c_1) = 130/250$, $purity_{Poor}(c_2) = 20/200$, $purity_{Poor}(c_3) = 50/550$. Moreover, the following parameters used in the fitness function are as follows: $\beta = 1.1$, 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 c_2 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 c_1 is 11/35 × (250)1.1 since $purity_{Poor}(c_1) = 52\%$ is greater than hotspot which is 30%, 11/35 is obtained by applying the function $purity_\gamma(c)$, thus we get $purity_{Poor}(c_1)$ = ((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:

$$q_{Poor}(X) = \frac{\frac{11}{35} \times 250^{1.1} + 0 + \frac{1}{11} \times 550^{1.1}}{1{,}000^{1.1}} = 0.115$$

2.4 Complexity

The time complexity of our proposed hybrid clustering algorithm depends on two factors: the complexity of the representative-based clustering algorithm and the complexity of the MOSAIC algorithm itself. Analyzing MOSAIC's complexity, we already discussed that the cost for constructing the Gabriel Graph over an n-element dataset with k representatives is equal to $O(k^3)$. After that, we have to merge the k vertices of the Gabriel Graph. Basically, a Delaunay Graph is a planar graph; since a Gabriel Graph is a connected subset of a Delaunay Graph, we have that the number e of edges of our GG is k-1 ≤ e ≤ 3k-6. This means that the number of edges e in the graph is always linear with respect to the number of vertices of the graph: $e = O(k)$. The merge-candidates are given by the edges of the Gabriel Graph; this means that, at each time, we have $O(k)$ merge-candidates. In each iteration, the number k of vertices (i.e. the number of representatives) is decreased by one. Thus, at the i^{th} iteration, $O(k-i)$ merge-candidates have to be evaluated, which adds up to $O(k^2)$ fitness function evaluations: $\left(O(k-1) + O(k-2) + \text{K} + 1\right)$. Putting this all together, we obtain a time complexity for MOSAIC algorithm equal to: $O\left(k^3 + k^2 \cdot O(q(X))\right)$ where $O(q(X))$ is the time complexity of the fitness function. A lower complexity for MOSAIC can be obtained if the fitness of a particular clustering can be computed incrementally during the merging stages based on results of previous fitness computations.

2.5 Incremental Update of Silhouette Function

In this section, a detailed description about the technique that can be used for incrementally updating the Silhouette function will be provided:

The silhouette function is computed by using the following values for each object $o_i \in C_k$:

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

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 $a_{i,m} = \frac{1}{|C_m|} \sum_{o_j \in C_m} d_{i,j}$ (3) and then find out the minimum amongst them.

Since the procedure merges only two clusters at a time, not all the values 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. 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 by using objects belonging to external clusters, the cluster minimizing the eq. (1) remains the same, therefore, in this case $a_i(p) = a_i(p-1)$, where *p* is the current step of computation.

b. 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 index of the cluster 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 one of the two clusters which have just been merged together by the merging procedure. Even this case must be divided in two sub-cases similarly to the case 1:

a. Let C_l and C_r be the two clusters that have just been merged and let *q* be the index of the cluster that minimized eq. (1) at step *p*–1 with $q \neq l, r$. 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 compute two matrices ($n \times k$), where n is the number of objects and k the number of clusters such as:

$A_{i,j} = \left(\sum_{o_m \in C_j} d_{i,m} \right)$ 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, an update of the two matrices has to be performed in order to manage the new situation. Since cluster C_l and cluster C_r have been merged together and have become a single cluster C_{l+r}, we have to compute:

$A_{i,l+r} = A_{i,l} + A_{i,r}$ for each object o_i.

The same update has to be performed on the matrix B. This operation takes time equal to $O(n)$ where n is the cardinality of the set of objects o_i.

If we analyze the computational complexity, it is simple to see that the computation of a_i during MOSAIC's execution takes time $O(k \cdot n \cdot m)$, 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 (this is the worst case because not always we have to perform the case 2b). 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 the Silhouette fitness function by using an incremental technique reduces to:

$O(k \cdot n \cdot m) + O(n^2)$.

At this time we are exploring the possibility to compute $q(X)$ locally, in order to reduce the intrinsic computational complexity introduced by this fitness function.

3 EXPERIMENTS

We set up experiments that evaluate MOSAIC for traditional clustering and supervised clustering. In the traditional clustering experiment, clustering results generated by MOSAIC are compared with ones generated by DBSCAN and *k*-means. In the supervised clustering section, we demonstrate how MOSAIC is effective in discovering arbitrary shape clustering; MOSAIC is used as a post-processing algorithm to agglomerate initial clusters generated by Supervised Clustering with Evolutionary Computing algorithm (SCEC).

3.1 Experiments on Traditional Clustering

We compare MOSAIC using the Silhouettes fitness function with DBSCAN and *k*-means[1]. Due to space limitations we are only able to present a few results; a more detailed

[1] In general, we would have preferred to compare our algorithm also with CHAMELEON and DENCLUE. However, we sadly have to report that executable versions of these two algorithms no longer exist.

experimental evaluation can be found in [3]. We conduct our experiments on a Dell Inspiron 600m laptop with a Intel(R) Pentium(R) M 1.6GHz processor with 512 MB of RAM. We set up three experiments that use the following datasets: an artificial dataset called **9Diamonds**[11] consisting of 3,000 objects with 9 natural clusters, **Volcano**[11] containing 1,533 objects, **Diabetes**[12] containing 768 objects, **Ionosphere**[12] containing 351 objects**,** and **Vehicle**[12] containing 8,469 objects.

Experiment 1: The experiment analyses Cohesion, Separation and Silhouettes over a MOSAIC run. The goal of this experiment is to understand how fitness with respect to the three fitness functions changes as MOSAIC merges clusters.

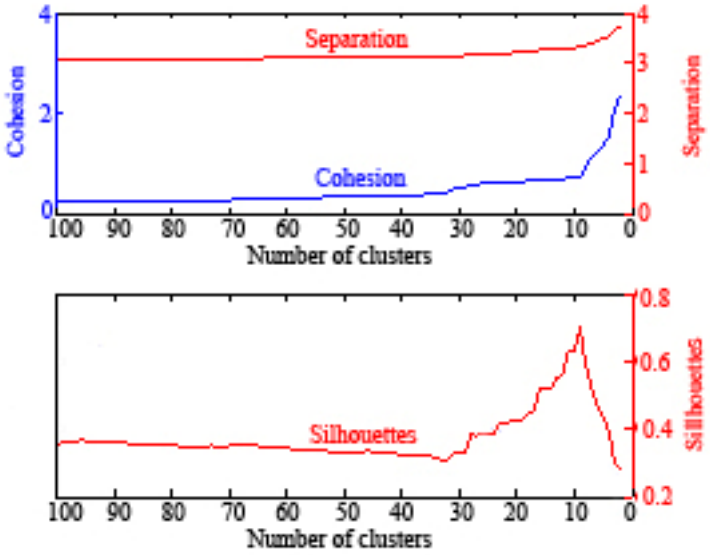
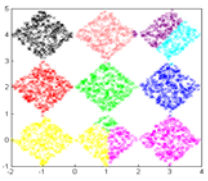


Fig. 6. Evaluation Graphs of Clustering Using Cohesion, Separation and Silhouettes

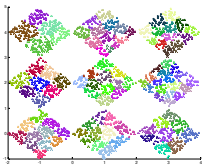
Discussion: Figure 6 depicts how Separation, Cohesion and Silhouettes evolve in one run of the post-processing algorithm for the 9Diamonds dataset. The top graph of the figure shows that as the number of clusters decreases, both Separation and Cohesion increase; the same pattern was observed for many other data sets. In other words, there is the trade off between the size of clusters, and Cohesion and Separation. In addition, Separation and Cohesion have the tendency increase more quickly for small *k* values. On the other hand, we observed [3], that the Silhouettes curve takes many different forms for different datasets. As shown in the bottom graph of the figure, the experimental result depicts a highest value of the Silhouette function when *k* reaches 9, which is number of *natural clusters* in the dataset.

Experiment 2: The experiment compares the clustering results generated by running *k*-means with k=9 with MOSAIC for the 9Diamonds dataset.

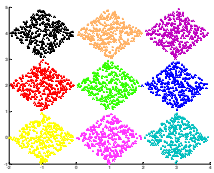
Discussion: As shown in Fig. 7 (a), *k*-means is not able to discover the natural clusters. MOSAIC, on the other hand, is able to discover the natural clusters by iteratively merging the sub-clusters that have been depicted in Fig. 7 (b) by maximizing the Silhouettes fitness function: the clustering with the highest fitness value is displayed in Fig. 7 (c).



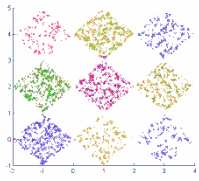
(a) Clusters created by k-means with k = 9



(b) MOSAIC clusters' input generated by k-means with k = 100



(c) MOSAIC output

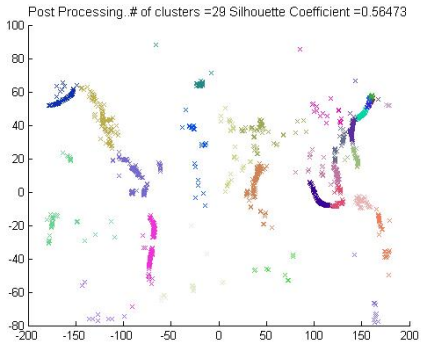


(d) DBSCAN with MinPts = 35 and ε = 0.05

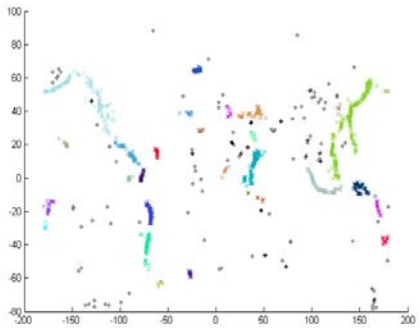
Fig. 7. Experimental results for the 9Diamonds dataset

Experiment 3: The experiment compares the clustering results generated by MOSAIC and DBSCAN for two two-dimensional datasets: 9Diamonds and Volcano.

Discussion: To use DBSCAN, we have to choose values for two parameters: *MinPts* and *ε*. One challenge of this experiment is to find proper values for those parameters. First, we use the procedure that has been proposed in the original paper [13] to select values for *MinPts* and *ε*. Unfortunately, this procedure does not work very well: DBSCAN just creates a single cluster for both datasets tested. Therefore, we created an interactive procedure to generate parameters for DBSCAN; we initially randomly generate a few parameter settings, visualize clustering results generated by DBSCAN for the different parameter settings, obtain feedback from the user about clustering quality, and then generate further parameter settings based on the user's feedback. After a couple of iterations, the parameters selected by the second procedure lead to much better results. We observe that *ε* values that produce better clustering results are much smaller than those suggested by analyzing the sorted *k*-dist graph. Fig. 7 (d) depicts one of the best clustering results obtained for DBSCAN for the 9Diamonds dataset. MOSAIC correctly clusters the dataset while DBSCAN reports a small number of outliers in the left corner of the bottom left cluster.



(a) MOSAIC



(b) DBSCAN with MinPts = 5 and ε = 0.02

Fig. 8. Experimental results of MOSAIC and DBSCAN on Volcano dataset

Volcano is a real world dataset that contains chain-like patterns with various densities. In general, DBSCAN and MOSAIC produce results of similar quality for this dataset. Fig. 8 depicts a typical result of this comparison: MOSAIC does a better job in identifying the long chains in the left half of the display (Fig. 8 (a)), whereas DBSCAN correctly identifies the

long chain in the upper right of the display (Fig. 8. (b)). DBSCAN and MOSAIC both fail to identify all chain patterns.

Experiment 4: This experiment compares MOSAIC and *k*-means on three high dimensional datasets: Vehicle, Ionosphere, and Diabetes. The quality of clustering results is compared using the Silhouettes cluster evaluation measure. MOSAIC's input are 100 clusters that have been created by running *k*-means. Next, MOSAIC is run and its Silhouette values are averaged over its 98 iterations. These Silhouette values are compared with the average Silhouette values obtained by running *k*-means with $k = 2,\ldots,99$. Table 1 summarizes the findings of that experiment.

Table 1. Information for the high dimensional datasets and experimental results

Dataset	Number of objects	Number of dimensions	Average Silhouette coefficient of *k*-means	Average Silhouette coefficient of MOSAIC
Vehicle	8,469	19	0.20013	0.37157
Ionosphere	351	34	0.2395	0.26899
Diabetes	768	8	0.23357	0.24373

Discussion: MOSAIC outperforms *k*-means quite significantly for the Vehicle dataset and we see minor improvements for the Ionosphere and Diabetes datasets.

3.2 Supervised Clustering Experiments

In traditional clustering section, *k*-means is used to generate initial clusters for MOSAIC. In this section, we also run a supervised clustering algorithm, namely SCEC, to generate a set of small input clusters for MOSAIC. SCEC [14] 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: *Mutation*, *Crossover*, and *Copy*, to members of the current generation that are selected based on the principles of survival of the fittest. Figure 9 shows 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 $q(X)$.

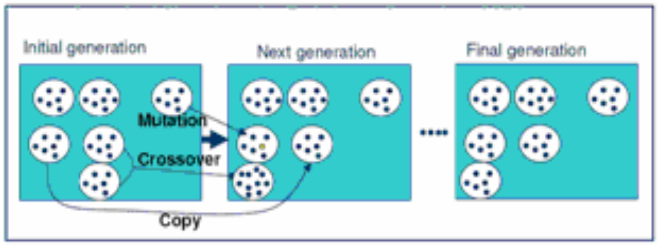


Fig. 9. Key features of the SCEC algorithm

We set up two experiments that use the following datasets: an artificial dataset called **Binary Complex 8** [11] consisting of 2,551 objects, **Binary Complex 9** [11] containing 3,031objects, **Earthquakes 1%** [11] containing 3,161 objects, **Volcano** [11] containing 1,533 objects**, Arsenic 20%** [11] containing 2,385 objects and **Diabetes** [12] containing 768 objects. All of the tested datasets contain objects belonging to two different classes except Earthquakes1% which contains 3 classes. The common parameters for SCEC of both experiments are listed in Table 2. In this paper the experiments on supervised clustering focus on discovering large size of high purity clusters. So we use the reward-based fitness function whose interestingness is the purity function as discussed in Section 2.3. The parameters settings for the fitness function are as follows: β=1.0001 and η=7—the parameters are chosen to instruct SCEC to produce a large number of initial clusters.

Table 2. Parameters for SCEC

Dataset	Number of objects
Initial crossover rate	0
Initial mutation rate	0.95
Copy rate	0.05
Population size	400
Number of generations	1,500
K for tournament	2

Experiment 5: The experiment evaluates post-processing SCEC clusters with MOSAIC for discovering arbitrary shape clusters for three two-dimensional datasets: Binary Complex 8, Binary Complex 9, and Earthquake (with sampling rate 1 percent) datasets. For MOSAIC we set β=1.3 and η=1 for Earthquakes dataset and β=3 and η=1 for the other datasets.

Discussion: The datasets contain clusters of varying in shape, density and distribution. We claim that MOSAIC is capable of merging neighboring clusters into larger continuous clusters while maintaining purity. The high purity and arbitrary shape clusters discovered by MOSAIC are displayed in Fig. 10. The clusters created by SCEC are shown on the middle panel and the clusters on the right are generated by MOSAIC. For Binary Complex 8 MOSAIC can merge the three red elongated clusters which belong to the same class. It can also agglomerate the two blue elongated shapes belonging the same class. The green, black and yellow clusters are not further merged because there is no neighboring edge of Gabriel Graph connecting them. For Binary Complex 9, MOSAIC can merge the two cyan circles, merge the red ellipse cluster and a red C-shape cluster, merge the green elongated and half of another green C-shape cluster, and merge the remaining half of blue C-shape cluster and the blue question mark shape cluster. The 2 halves of C-shape cluster in green and blue are not merged together because there exists an intervening cluster, generated by SCEC, with a different majority class (the tiny red part at the middle of the 2 halves of C-shape cluster is

agglomerated to the red C-shape cluster). Since Gabriel Graph only creates edges from the flawed cluster to them, MOSAIC does not merge because it is required to merges one half with the red C-shape cluster resulted in decreasing fitness value. For Earthquakes1% dataset, MOSAIC is able to form chain-like clusters. The clustering result on the rightmost is very comparable to original clusters residing on the leftmost. It also should be emphasized that MOSAIC can be employed as a post-processing clustering algorithm for any representative clustering algorithm to enhance the clustering result.

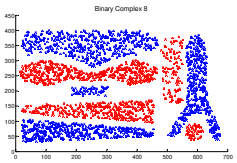
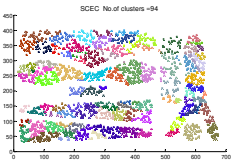
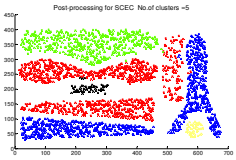
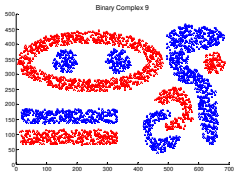
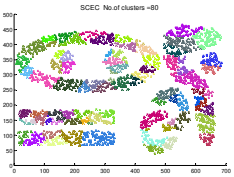
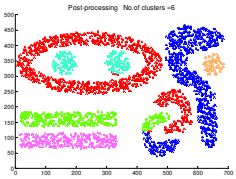
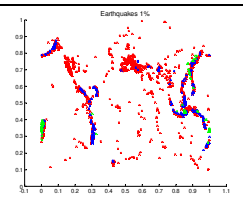
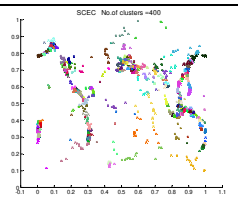
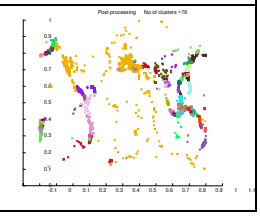
Datasets	Original Clusters	Clusters generated by SCEC	MOSAIC result
Binary Complex 8			
Binary Complex 9			
Earthquakes 1%			

Fig. 10. Experimental results of SCEC and MOSAIC on Binary Complex 8, Binary Complex 9 and Earthquakes 1% datasets

Experiment 6: The empirical study conducts a qualitative analysis of MOSAIC and SCEC for different fitness parameter settings. We perform the qualitative analysis of the benefits of MOSAIC as a post-processing clustering algorithm for SCEC clusters, in terms of quality of cluster $q(X)$, the purity of clusters, and the number of clusters. The same fitness function used in Experiment 5 is also used in MOSAIC with 4 different parameter settings: (β=1.0001, η=7), (β=1.01, η=6), (β=1.3, η=1), (β=3, η=1).

Discussion: The experimental results listed in Table 3 indicate that post-processing technique considerably improves the quality of SCEC clusters. We can draw a conclusion from the comparison between SCEC and MOSAIC that MOSAIC 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. In general, the experimental results also convince us the use of parameter β in clustering tasks; larger values for the parameter β usually results in a smaller number of clusters in the best clustering. From a quantitative point of view, Table 3 shows that the post-processing technique improves average purity by 2.8%, average cluster quality by 13.3% and decreases the average number of clusters by 51.4%.

Table 3. The qualitative and quantitative analysis of SCEC and MOSAIC

Dataset	Parameters	SCEC			MOSAIC		
		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	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
Earth-quake1%	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%	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

4 RELATED WORK

Discovering arbitrary shape clusters is very important in many domains, such as hot spot detection, region discovery and spatial data mining. Jiang [1] proposes spatial clustering techniques that employ hierarchical clustering accompanied by tree-like diagrams and claims that this is a beneficiary for visualizing cluster hierarchies at different levels of detail. Anders [15] developed an unsupervised graph-based clustering algorithm, called Hierarchical Parameter-free Graph Clustering (HPGCL) for spatial data analysis.

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 equal to $O(n^2)$ or worse limits its application to small and medium-sized data sets. Furthermore, clusters obtained by AHC are not necessarily contiguous, as illustrated in Fig. 11: a hierarchical clustering algorithm that uses average linkage[2] would merge clusters C3 and C4, although the two clusters are not neighboring. This example emphasizes the need to disallow merging of non-neighboring clusters in agglomerative clustering. Gao et al. [16] make use of Delaunay triangulation to

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

identify the neighboring distance among instances before identifying clusters using Minimum Spanning Tree. Proximity graphs are also used in divisive clustering. Amoeba [17] is a divisive clustering approach that operates on proximity graphs; after constructing a Delaunay graph for all instances, the algorithm recursively divides a cluster into sub-clusters by removing Delaunay edges whose distance exceed global mean distance.

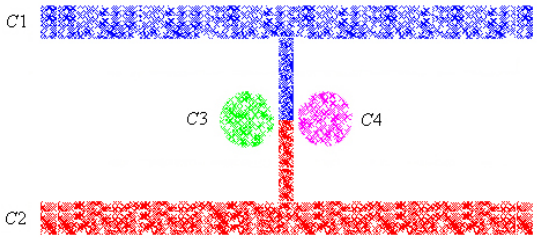


Fig. 11. Merging elongated clusters

Density-based clustering methods [13, 18, 19, and 20] have been found to be efficient for discovering dense arbitrary shaped clusters, such as the ones in the 9Diamonds dataset. The main drawbacks of density-based clustering algorithms are their need for parameters tuning, and their usually poor performance for datasets with varying density. Moreover, they do not seem to be suitable for high dimensional data. To alleviate one of those drawbacks, Duan et al. [21] propose LDBSCAN which relies on local density measures to handle varying density in datasets.

There has been significant research centering on hybrid clustering. CURE is a hybrid clustering algorithm that integrates a partitioning algorithm with an agglomerative hierarchical algorithm [22]. CURE iteratively merges the two clusters that have the closest pair of representatives, and updates mean and a set of representative points. CHAMELEON [23] provides a sophisticated two-phased clustering algorithm. In the first phase, it uses a multilevel graph partitioning algorithm to create an initial set of clusters and in the second phase it iteratively merges clusters maximizing relative inter-connectivity and relative closeness. MOSAIC also relies on two-phase clustering but it has a major advantage over CHAMELEON and CURE by being able to plug-in any fitness function and not being restricted to evaluate clusters based on inter-connectivity and closeness. Lin et al. and Zhong et al. [24 and 25] propose hybrid clustering algorithms that combine representative-based clustering and agglomerative clustering methods. However they employ different merging criteria and perform a narrow search that only considers a single pair of merge candidates. Surdeanu et al. [26] proposes a hybrid clustering approach that combines agglomerative clustering algorithm with the Expectation Maximization (EM) algorithm.

Contrast to traditional clustering, a family of supervised clustering algorithms has been proposed by Eick et al. [14, 27, 28] that allow for plug-in fitness functions. The family of supervised clustering algorithms consists of agglomerative-based, grid-based, representative-based and density-based clustering algorithms. They are mainly used in region discovery [29, 30, 31].

5 CONCLUSION

This paper proposes a novel approach that approximates arbitrary-shape clusters through unions of small convex polygons that have been obtained by running a representative-based clustering algorithm. An agglomerative clustering algorithm called MOSAIC is introduced

that greedily merges neighboring clusters maximizing an externally given fitness function. Gabriel graphs are used to determine which clusters are neighboring. We claim that using proximity graphs increases the number of merge candidates considerably over traditional agglomerative clustering algorithms that only consider "closest" clusters for merging, resulting in clusters of higher quality. MOSAIC is quite general and can be used with any representative-based clustering algorithm, any proximity graph, and any fitness function. Moreover, we claim that MOSAIC can be effectively applied to higher dimensional data.

MOSAIC also has some similarity with 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 is much more general by supporting more variety of shapes and it allows for convex polygons of different sizes. On the other hand, for a given grid structure it is easy to determine which clusters are neighboring, which is not the case for our approach.

We conducted experiments whose results suggest that using MOSAIC in conjunction with *k*-means can significantly improve cluster quality. Using Silhouettes function as a fitness function we also compared MOSAIC with DBSCAN; both algorithms obtained results of similar quality for most datasets tested. However, before using DBSCAN we had to spend significant efforts for parameter tuning which is not the case when using MOSAIC which only requires a single input parameter: the fitness function. Based on our initial experimental results, we do not believe that the Silhouettes function is the best possible fitness function to find arbitrary shape clusters. Consequently, in our current research we investigate to find more suitable fitness functions for this purpose.

The supervised clustering experiments reveal another advantage of applying MOSAIC as a post-processing algorithm to results of supervised clustering algorithms. The proposed post-processing technique considerably improves the quality of SCEC clusters. The experimental results show that MOSAIC is able to obtain complex shape clusters in both artificial and real world datasets. Specifically it can identify chain-like clusters in the Earthquake dataset and clusters residing in another cluster in the Binary Complex 9 dataset. Such kinds of complex shape clusters are hardly identified by traditional clustering algorithms.

6 ACKNOWLEDGEMENT

This research was supported in part by a grant from the Environmental Institute of Houston (EIH).

7 REFERENCES

1. Jiang, B. (2004). Spatial Clustering for Mining Knowledge in Support of Generalization Processes in GIS, *ICA Workshop on Generalisation and Multiple representation.*
2. Tan, M., Steinbach, M., & Kumar, V. (2005). (1st Ed.), *Introduction to Data Mining*. Addison Wesley.
3. Choo, J. (2007). Using Proximity Graphs to Enhance Representative-based Clustering Algorithms, *Master Thesis*, Department of Computer Science, University of Houston, TX.
4. Gabriel, K.R., & Sokal R.R. (1969). A New Statistical Approach to Geographic Variation Analysis, *Systematic Zoology*, Vol. 18 (pp. 259-278).
5. Toussaint, G. (1980). The Relative Neighborhood Graph of A Finite Planar Set, *in Int. Conf. Pattern Recognition*, Vol. 12 (pp. 261-268).
6. Kirkpatrick, D. (1980). A note on Delaunay and Optimal Triangulations, *Information Processing Letters*, Vol. 10 (pp. 127-128).

7. Okabe, A., Boots, B., & Sugihara, K. (1992). *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley, New York .
8. Bhattacharya, B., Poulsen, R., & Toussaint, G. (1981). Application of Proximity Graphs to Editing Nearest Neighbor Decision Rule, *Int. Sym. on Information Theory*.
9. Asano, T., Ibaraki, T., Imai, H., & Nishizeki, T. (1990). Algorithms, *Lecture Notes in Computer Science, Springer-Verlag* (pp. 70-71), New York Berlin Heidelberg.
10. Rousseeuw, P.J. (1987). Silhouettes: A Graphical Aid to The Interpretation and Validation of Cluster Analysis, *Int. J. Computational and Applied Mathematics*, Vol. 20. (pp. 53-65).
11. Data Mining and Machine Learning Group website, University of Houston, Texas (2007), http://www.tlc2.uh.edu/dmmlg/Datasets
12. UCI Machine Learning Repository (2007), http://www.ics.uci.edu/~mlearn/MLRepository.html
13. Ester, M., Kriegel, H.P., Sander, J., & Xu, X. (1996). Density-Based Spatial Clustering of Applications with Noise, *Int. Conf. Knowledge Discovery and Data Mining*.
14. C. Eick, N. Zeidat, & Z. Zhao (2004). Supervised Clustering --- Algorithms and Benefits, *Int. Conf. Tools with AI (ICTAI)*.
15. Anders, K.H. (2003). A Hierarchical Graph-Clustering Approach to Find Groups of Objects. Technical Paper, *ICA Commission on Map Generalization, 5th Workshop on Progress in Automated Map Generalization*.
16. Gao, D., Peuquet, D., & Gahegan, M. (2002). Opening the Black Box: Interactive Hierarchical Clustering for Multivariate Spatial Patterns, *10th ACM Int. Symposium on Advances in geographic information systems* (pp. 131-136).
17. Estivill-Castro, V. & Lee, I. (2000). Amoeba: Hierarchical Clustering Based On Spatial Proximity Using Delaunaty Diagram., *9th International Symposium on Spatial Data Handling*, (pp. 7a.26-7a.41).
18. Sander, J., Ester, M., Kriegel, & H.P. Xu, X. (1998). Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and its Applications, *Int. Conf. Data Mining and Knowledge Discovery* (pp. 169-194).
19. Kriegel, H.P., & Pfeifle, M. (2005). Density-Based Clustering of Uncertain Data. *Int. Conf. Knowledge Discovery in Data Mining* (pp. 672-677).
20. Hinneburg, A., & Keim, D. (1998). An Efficient Approach to Clustering in Large Multimedia Databases with Noise, *Int. Conf. Knowledge Discovery in Data Mining*.
21. Duan, L., Xu, L., Guo, F., Lee, J., & Yan, B. (2007). A Local-Density Based Spatial Clustering Algorithm with Noise, *Information Systems* Vol. 32, (pp. 978-986).
22. Guha, S., Rastogi, R., & Shim, K. (1998). CURE: An Efficient Clustering Algorithm for Large Databases, *Int. Conf. ACM SIGMOD on Management of data* (pp. 73-84).
23. Karypis, G., Han, E.H., & Kumar, V. (1999). CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling, *IEEE Computer* Vol. 32 (pp. 68-75).
24. Lin, C., & Chen, M. (2002). A Robust and Efficient Clustering Algorithm based on Cohesion Self-Merging, *Int. Conf. 8th ACM SIGKDD on Knowledge Discovery and Data Mining* (pp. 582-587).
25. Zhong, S., & Ghosh, J. (2003). A Unified Framework for Model-based Clustering. *Int. J. Machine Learning Research*, Vol.4 (pp. 1001-1037).
26. Surdeanu, M., Turmo, J. & Ageno, A. (2005). A Hybrid Unsupervised Approach for Document Clustering, *Int. Conf. 11h ACM SIGKDD on Knowledge Discovery in Data Mining* (pp. 685-690).

27. Eick, C.F., Vaezian, B., Jiang, D., & Wang, J. (2006). Discovery of Interesting Regions in Spatial Datasets Using Supervised Clustering, *10th European Conf. on Principles and Practice of Knowledge Discovery in Databases*, Vol. 4213 (pp. 127-138).
28. Jiang, D., Eick, C.F., & Chen, C.S. (2007). On Supervised Density Estimation Techniques and Their Application to Clustering, *15th ACM Int. Symposium on Advances in Geographic Information Systems*.
29. Ding, W., Eick, C.F., Wang, J., & Yuan, X. (2006). A Framework for Regional Association Rule Mining in Spatial Datasets, *6th IEEE Int. Conf. on Data Mining* (pp. 851-856).
30. Ding, W., Eick, C.F., Yuan, X., Wang, J., & Nicot, J.-P. (2007). On Regional Association Rule Scoping, *Int. Workshop on Spatial and Spatio-Temporal Data Mining* (pp.595-600).
31. Ding, W., Jiamthapthaksin, R., Parmar, R., Jiang, D., Stepinski, T., & Eick, C.F. (2008). Towards Region Discovery in Spatial Datasets, *Pacific-Asia Conf. on Knowledge Discovery and Data Mining*.