The Fuzzy C Spherical Shells Algorithm: A New Approach

Raghu Krishnapuram, *Member, IEEE,* Olfa Nasraoui, *Student, Member, IEEE,* and Hichem Frigui, *Student, Member, IEEE*

Abstract— **The fuzzy c spherical shells (FCSS) algorithm is specially designed to search for clusters that can be described by circular arcs or, more generally, by shells of hyperspheres. In this paper, a new approach to the FCSS algorithm is presented. This algorithm is computationally and implementationally simpler than other clustering algorithms that have been suggested for this purpose. An unsupervised algorithm which automatically finds the optimum number of clusters is also proposed. This algorithm can be used when the number of clusters is not known. It uses a cluster validity measure to identify good clusters, merges all compatible clusters, and eliminates spurious clusters to achieve the final result. Experimental results on several data sets are presented.**

I. Introduction

MANY fuzzy (and hard) clustering algorithms have been suggested and used in the literature to partition data into clusters. There is a whole class of clustering algorithms in which an objective function based on a distance measure is iteratively minimized to obtain the final partition. The distance measure chosen and the objective function being optimized depend on the geometric structure of the clusters. Different distances have been invented to search for clusters of specific shapes in feature space. For example, the K means algorithm, using the Euclidian distance, looks for clusters that are hyperspherical in shape. Until recently it has been difficult to detect clusters that can be described by circular arcs or, more generally, by shells of hyperspheres. Dave's fuzzy c-shells (FCS) algorithm [1], [2] has proved to be successful in detecting such clusters, and several impressive examples involving two-dimensional data sets are given in [1] and [2]. This algorithm has also been generalized to the case of elliptical shells [3], [4]. However the FCS algorithm is somewhat implementationally complex since it requires the use of Newton's method to solve two coupled nonlinear equations for the center and radius of each cluster in each iteration. Bezdek *et al.* have suggested a modification to this algorithm to reduce the computational burden arising from Newton's method [5]. In this paper, we propose an alternative FCS algorithm (which we call the fuzzy c spherical shells algorithm) to overcome this problem. Unlike Dave's method, our method does not involve nonlinear equations. This makes our algorithm straightforward and, what is more important, computationally more attractive. In addition, we also propose an unsupervised algorithm to determine the optimum number of clusters c, when this is not known. This unsupervised algorithm uses a cluster validity (performance) measure to identify good clusters, merges compatible clusters, and eliminates spurious clusters to achieve the final result.

In Section II, we present the hard and fuzzy versions of our c spherical shells algorithm. In Section III, we introduce several cluster validity measures suitable for shell clusters, and describe our unsupervised algorithm which can be used to determine the optimum number of clusters when this is not known *a priori.* In Section IV, several examples of clustering using the proposed unsupervised algorithm are shown. Finally, Section V gives the summary and conclusions.

Manuscript received June 11, 1991; revised September 27, 1991. This work was supported by NASA through the University of Houston, Clear Lake (RICIS Project SE 42).

The authors are with the Department of Electrical and Computer Engineering, University of Missouri, Columbia, MO 65211.

IEEE Log Number 9201464.

II. The C Spherical Shells Algorithms

Let $\boldsymbol{x}_j$ be a point in feature space. We assume that each cluster resembles a hyperspherical shell. Therefore, the prototypes λ_i consist of two parameters $(\boldsymbol{c}_i, r_i)$, where $\boldsymbol{c}_i$ is the center of the hypersphere and r_i is the radius. We define the distance from $\boldsymbol{x}_j$ to a prototype $\lambda_i = (\boldsymbol{c}_i, r_i)$ as

$$d_{ij}^2 = d^2(\boldsymbol{x}_j, \lambda_i) = \left(\|\boldsymbol{x}_j - \boldsymbol{c}_i\|^2 - r_i^2\right)^2. \tag{1}$$

Note that the right-hand side of (1), when equated to zero, also gives the equation of the hypersphere. In general, the closer $\boldsymbol{x}_j$ is to the specific hypersphere, the smaller the distance will be. Based on this distance measure, we now define the hard and fuzzy c spherical shells algorithms.

A. The Hard C Spherical Shells Algorithm

We define the objective function to be minimized in this case as

$$J(L) = \sum_{i=1}^{K} \sum_{\boldsymbol{x}_j \in \lambda_i} d_{ij}^2, \tag{2}$$

where $L = (\lambda_1, \cdots, \lambda_K)$, and K is the number of clusters. In order to minimize the objective function in (2), we rewrite the distance in (1) as

$$d_{ij}^2 = \boldsymbol{p}_i^T \boldsymbol{M}_j \boldsymbol{p}_i + \mathbf{v}_j^T \boldsymbol{p}_i + b_j,$$

where

$$b_j = \left(\boldsymbol{x}_j^T \boldsymbol{x}_j\right)^2 \qquad \boldsymbol{v}_j = 2\left(\boldsymbol{x}_j^T \boldsymbol{x}_j\right)\boldsymbol{y}_j$$

$$\boldsymbol{y}_j = \begin{bmatrix} \boldsymbol{x}_j \\ 1 \end{bmatrix} \tag{3a}$$

$$\boldsymbol{M}_j = \boldsymbol{y}_j \boldsymbol{y}_j^T \quad \text{and} \quad \boldsymbol{p}_i = \begin{bmatrix} -2\boldsymbol{c}_i \\ \boldsymbol{c}_i^T \boldsymbol{c}_i - r_i^2 \end{bmatrix} \tag{3b}$$

1045–9227/92$03.00 © 1992 IEEE

Therefore,

$$J(L) = \sum_{i=1}^{K} \sum_{\boldsymbol{x}_j \in \lambda_i} \left(\boldsymbol{p}_i^T \boldsymbol{M}_j \boldsymbol{p}_i + \boldsymbol{v}_j^T \boldsymbol{p}_i + b_j \right). \tag{4}$$

We may assume that the vectors $\boldsymbol{p}_i$ are independent of each other. Hence, the vectors $\boldsymbol{p}_i$ that minimize (4) must satisfy

$$\sum_{\boldsymbol{x}_j \in \lambda_i} (2\boldsymbol{M}_j \boldsymbol{p}_i + \boldsymbol{v}_j) = 0. \tag{5}$$

If we define

$$\boldsymbol{H}_i = \sum_{x_j \in \lambda_i} \boldsymbol{M}_j \quad \text{and} \quad \boldsymbol{w}_i = \sum_{\boldsymbol{x}_j \in \lambda_i} \boldsymbol{v}_j, \tag{6}$$

from (5) we obtain

$$\boldsymbol{p}_i = -\frac{1}{2} (\boldsymbol{H}_i)^{-1} \boldsymbol{w}_i. \tag{7}$$

The resulting hard c spherical shells (HCSS) algorithm is summarized below.

The Hard C Spherical Shells (HCSS) Algorithm
Fix the number of clusters K;
Set iteration counter $l = 1$ and initialize the hard K-partition (using the K means algorithm);
Repeat
Calculate $\boldsymbol{H}_i^{(l)}$ and $\boldsymbol{w}_i^{(l)}$ for each cluster using (6);
Compute $\boldsymbol{p}_i^{(l)}$ for each cluster using (7);
Classify $\boldsymbol{x}_j$ into cluster λ_i if $d_{ij}^2 \leq d_{kj}^2$ for all $k \neq i$;
Increment l;
Until $\left(\left\| \boldsymbol{p}^{(l-1)} - \boldsymbol{p}^{(l)} \right\| < \varepsilon \right)$;

B. The Fuzzy C Spherical Shells Algorithm

For the fuzzy case, we minimize the following objective function:

$$J(L, \boldsymbol{U}) = \sum_{i=1}^{K} \sum_{j=1}^{N} (\mu_{ij})^m d_{ij}^2. \tag{8}$$

In (8) N is the total number of feature vectors, and $\boldsymbol{U} = [\mu_{ij}]$ is a $K \times N$ matrix called the fuzzy K-partition matrix [6] satisfying the following conditions:

$$\mu_{ij} \in [0,1] \text{ for all } i \text{ and } j,$$
$$\sum_{i=1}^{K} \mu_{ij} = 1 \text{ for all } j,$$
$$\text{and } 0 < \sum_{\text{j=1}}^{\text{N}} \mu_{ij} < \text{N for all } i.$$

Here μ_{ij} is the grade of membership of the feature point $\boldsymbol{x}_j$ in cluster λ_i, and $m \in [1, \infty)$ is a weighting exponent called the fuzzifier. As in the hard case, it is easy to show that the vectors $\boldsymbol{p}_i$ that minimize (8) are given by (7), where

$$\boldsymbol{H}_i = \sum_{j=1}^{N} (\mu_{ij})^m \boldsymbol{M}_j \qquad \boldsymbol{w}_i = \sum_{j=1}^{N} (\mu_{ij})^m \boldsymbol{v}_j, \tag{9}$$

and $\boldsymbol{v}_j$ and $\boldsymbol{M}_j$ are given by (3). Following Bezdek's theorem for the fuzzy K means [6], it can be shown that the memberships will be updated according to

$$\mu_{ik} = \begin{cases} \dfrac{1}{\sum_{j=1}^{K} \left(\frac{d_{ik}}{d_{jk}} \right)^{\frac{2}{m-1}}} & \text{if } I_k = \Phi \\ 0 \; i \notin I_k & \text{if } I_k \neq \Phi \\ \sum_{i \in I_k} \mu_{ik} = 1 & \text{if } I_k \neq \Phi, \end{cases} \tag{10}$$

where $I_k = \{ i | 1 \leq i \leq K, d_{ik}^2 = 0 \}$. The resulting fuzzy c spherical shells (FCSS) algorithm is summarized below.

The Fuzzy C Spherical Shells (FCSS) Algorithm
Fix the number of clusters K; fix m, $1 < m < \infty$;
Set iteration counter $l = 1$;
Initialize the fuzzy K-partition $U^{(o)}$ (using the fuzzy c means algorithm);
Repeat
Calculate $\boldsymbol{H}_i^{(l)}$ and $\boldsymbol{w}_i^{(l)}$ for each cluster λ_i using (9);
Compute $\boldsymbol{p}_i^{(l)}$ for each cluster λ_i using (7);
Update $U^{(l)}$ using (10);
Increment l;
Until $\left(\left\| U^{(l-1)} - U^{(l)} \right\| < \varepsilon \right)$;

Both the hard and the fuzzy c spherical shells algorithm require the inversion of the matrix $\boldsymbol{H}_i$. This is quite trivial when the feature space is two-dimensional or three-dimensional. In the hard case, the inverse will exist if there are at least $n+1$ noncollinear points in each cluster, where n is the dimensionality of the feature space. In the fuzzy case, theoretically the inverse will always exist as long as $N > n+1$ and the feature vectors are not collinear.

III. Determination of the Optimal Number of Clusters

Several cluster validity criteria have been presented in the literature [6]–[10]. For example, the partition coefficient, fuzzy set decomposition measure, classification entropy, and proportion exponent are discussed by Bezdek *et al.* [6]. However these performance measures are based solely on the memberships in the partition matrix $\boldsymbol{U}$, and thus do not reflect the actual geometric structure of the data set. Other cluster validity criteria, such as the fuzzy hypervolume and the average partition density, which relate more to the cluster shapes, have been presented by Gath and Geva [8]. Similar measures have been presented for shell-type clusters by Dave [9], [10]. These criteria are more appropriate for evaluating the partition of c-shell clustering algorithms, and hence can be used to assess the validity of a shell cluster. Here we present some performance measures that we found to be effective in identifying good partitions in the case of spherical shell clusters.

The hard average shell thickness of a cluster may be defined as

$$T_i = \frac{1}{N_i r_i} \sum_{x_j \in \lambda_i} (\|\boldsymbol{x}_j - \boldsymbol{c}_i\| - r_i)^2, \tag{11}$$

where N_i is the number of feature vectors assigned to cluster λ_i. In the fuzzy case, the fuzzy average shell thickness may be defined to be

$$T_i = \frac{\sum_{j=1}^{N} (\mu_{ij})^m (\|\boldsymbol{x}_j - \boldsymbol{c}_i\| - r_i)^2}{r_i \sum_{j=1}^{N} (\mu_{ij})^m}. \tag{12}$$

In (11) and (12), the thickness of each shell is measured relative to its radius. The average shell thickness is a measure of how "compact" a shell is. An overall validity measure for the partition created by the clustering algorithm can be obtained by adding the individual average shell thicknesses. We define this to be the total average shell thickness, T, as follows:

$$T = \sum_{i=1}^{K} T_i. \tag{13}$$

The measures in (11)–(13) are similar to those defined by Dave [9], [10]. Validity measures may also be defined using hypervolume and density [8], [10]. To do this, the distance vector from a feature point to a shell prototype is first defined as

$$\boldsymbol{\delta}_{ij} = (\boldsymbol{x}_j - \boldsymbol{c}_i) - r_i \frac{(\boldsymbol{x}_j - \boldsymbol{c}_i)}{\|\boldsymbol{x}_j - \boldsymbol{c}_i\|}. \tag{14}$$

The (hard) spherical shell covariance matrix of cluster λ_i is defined to be

$$\boldsymbol{F}_i = \frac{1}{N_i} \sum_{x_j \in \lambda_i} \boldsymbol{\delta}_{ij} \boldsymbol{\delta}_{ij}^T, \tag{15}$$

where N_i is the number of feature vectors assigned to cluster λ_i. The (hard) spherical shell hypervolume of a cluster may be defined as

$$V_i = \sqrt{\det(\boldsymbol{F}_i)}\,. \tag{16}$$

The hard spherical shell density of a cluster may be defined as

$$D_i = \frac{S_i}{V_i}, \tag{17}$$

where S_i is the sum of close members of shell λ_i, given by

$$S_i = \text{number of } \boldsymbol{x}_j \in \lambda_i \text{ such that } \boldsymbol{\delta}_{ij}^T \boldsymbol{F}_i^{-1} \boldsymbol{\delta}_{ij} < 1. \tag{18}$$

For the fuzzy case, the fuzzy spherical shell covariance matrix is defined by

$$\boldsymbol{F}_i = \frac{\sum_{j=1}^{N} (\mu_{ij})^m \boldsymbol{\delta}_{ij} \boldsymbol{\delta}_{ij}^T}{\sum_{j=1}^{N} (\mu_{ij})^m}. \tag{19}$$

The fuzzy spherical shell hypervolume of a cluster is still given by (16), except that $\boldsymbol{F}_i$ is computed using (19). Finally, the fuzzy spherical shell density of a cluster may be defined by (17), where the sum of close members S_i is given by

$$S_i = \sum_j \mu_{ij} \text{ such that } \boldsymbol{\delta}_{ij}^T \boldsymbol{F}_i^{-1} \boldsymbol{\delta}_{ij} < 1. \tag{20}$$

An alternative definition of S_i may also be formulated as

$$S_i = \sum_j \mu_{ij} \text{ such that } \|\boldsymbol{\delta}_{ij}\| < p\sigma_i, \tag{21}$$

where p is a suitable constant and

$$\sigma_i = \sqrt{\mathrm{Tr}(\boldsymbol{F}_i)}\,. \tag{22}$$

The total hypervolume, V, and partition density, P_D, may be defined as

$$V = \sum_{i=1}^{K} V_i \text{ and } P_D = \sum_{i=1}^{K} D_i. \tag{23}$$

Equation (23) holds for both the hard and the fuzzy case, but the V_i and D_i are computed differently for the hard and fuzzy cases. The total hypervolume and partition density are measures of how good the overall clustering is, whereas the hypervolume and spherical shell density are measures of how good a given spherical shell cluster is.

One method to determine the optimal number of clusters is to perform clustering for a range of K values, and pick the K value for which a suitable performance measure is minimized (or maximized). The performance measures that may be used include the (fuzzy) total average shell thickness, the total (fuzzy) hypervolume, and the partition density. However this method is rather tedious. Also in our experiments, we found that the K value obtained this way may not be optimum, because the corresponding partition may split a single shell cluster into more than one cluster. We now present an alternative unsupervised c spherical shells (UCSS) clustering algorithm which is computationally more efficient, because it does not perform the clustering for an entire range of K values.

Our method progressively clusters the data starting with an overspecified number, $K_{\max}$, of clusters. $K_{\max}$ is the largest number of clusters to be expected in a particular problem. Initially, the hard or fuzzy CSS algorithm is run with $K = K_{\max}$. After the algorithm converges, compatible clusters are merged. Next, points that are close enough to some identified "good" clusters are temporarily removed from the data set to reduce computations. Then spurious clusters (with very few points) are eliminated. The FCSS algorithm is invoked again with the remaining feature points. The above procedure is repeated until no more merging, removing, or elimination occurs, or until $K = 1$. In our experiments, we used the spherical shell density (given by (17), (21) and (22)) as the measure to pick "good" clusters. However, hypervolume and shell thickness may also be used with approximately the same results. (See [10] for a comparison of performance measures.) We now discuss some of the steps in the UCSS algorithm in more detail.

In the merging step, two clusters, λ_i and λ_j, are considered compatible if

$$\|\boldsymbol{c}_i - \boldsymbol{c}_j\| < \varepsilon_1 \quad \text{and} \quad |r_i - r_j| < \varepsilon_2. \tag{24}$$

After the merging has been performed, a cluster is identified as a "good" cluster if its cluster partition density $D_i > T_D$,

where T_D is some threshold, and the number of points in the cluster $N_i > N_{\min}$, where $N_{\min}$ is dependent on the number of feature points being clustered and the number of clusters K. A good choice for $N_{\min}$ was found to be

$$N_{\min} = \frac{N}{K+1}. \tag{25}$$

The above value for $N_{\min}$ was used in all our experiments. Also, the following rule of thumb may be used to remove points that are close to good clusters temporarily:

$$\text{remove feature point } \boldsymbol{x}_k \text{ if } |\|\boldsymbol{x}_k - \boldsymbol{c}_i\| - r_i| < \eta\sigma_i, \tag{26}$$

where σ_i is given by (22) and η is some multiplicative constant.

When the fuzzy or hard CSS algorithm is used for clustering, the algorithm sometimes finds a few small spurious clusters. These spurious clusters frequently arise as a consequence of noise points. Such tiny clusters are not compatible with any of the rest of the clusters; hence merging cannot correct this problem. To eliminate such spurious clusters which contain too few points, we just discard the prototypes for the small clusters, and decrement K by the number of discarded clusters. This eventually forces the points belonging to the spurious clusters to be reassigned to the closest "good" clusters. The procedure of merging, removing, and eliminating is repeated until the K value can no longer be decremented. The unsupervised algorithm that finds the optimum number of clusters taking into account the above mentioned steps is summarized below.

The Unsupervised C Spherical Shells (UCSS) Algorithm
Set $K = K\max$; fix $m, 1 < m < \infty$;
$KRemoved := 0$; $MergeFlag := EliminmateFlag :=$
$RemoveFlag :=$ TRUE;
While $K > 1$ **and** (*MergeFlag* = TRUE **or** *EliminmateFlag* = TRUE **or**
Remove Flag = TRUE) **do**
MergeFlag := *EliminmateFlag* := *RemoveFlag*:=FALSE;
Perform the c spherical shells algorithm with the number of clusters = K;
Store the final K prototypes;
Merge compatible prototypes among the K prototypes using (24), update K, and set *MergeFlag* = TRUE if merging has occured;
Identify good clusters using (17) and set *RemoveFlag* = TRUE if there are any good clusters;
Remove points that lie close to the newly identified good clusters using (26);
Store the good clusters' prototypes in the prototype list, and update K and *KRemoved* accordingly;
Eliminate tiny clusters, decrement K accordingly, and set *EliminmateFlag* = TRUE if any elimination has occured;
Save the remaining clusters' prototypes;
End While
$K := K + KRemoved$;
Replace all the removed feature points back into the data set.
Append the list of remaining clusters' prototypes from the last iteration in the while loop to the list of removed clusters' prototypes;
Merge compatible prototypes in the prototype list and update K;
Update U using the prototypes in the prototype list and (10);
Perform the CSS algorithm with the new K;
Do
Eliminate tiny clusters and decrement K accordingly;
Perform the CSS algorithm with the new K, using the remaining prototypes to obtain initial partition;
Until No More Elimination Takes Place

The above UCSS algorithm is applicable to both the hard and the fuzzy case.

IV. Experimental Results

Although the algorithms presented in the previous sections are applicable to feature spaces of any dimension, here we present only results of two-dimensional and three-dimensional data sets. We found that the HCSS algorithm is much faster than the FCSS algorithm, but performs well only when the data set is "clean," i.e., when the clusters are well separated and there is little noise. This is because the HCSS algorithm has a higher tendency to get stuck in local minima, and sometimes it terminates abruptly because of the occurrence of singular matrices (see remark at the end of Section II). Therefore, the HCSS algorithm is not very robust, and we do not present the results of the HCSS algorithm in this paper.

Some of the two-dimensional data sets were artificially generated, and had between 89 and 132 feature points. Uniformly distributed noise with an interval of 2 was added to the feature point locations so that they do not always lie exactly on the ideal circles. In addition, noise points were added at random locations to some of the data sets. The rest of the two-dimensional data sets consist of real data (object edges) and had between 591 and 951 points. The (three-dimensional) range image examples were artificially generated, and had between 13 500 and 23 905 points, but they were sampled at a sampling rate of 3 in both directions, yielding between 1494 and 2656 actual feature points. In all the examples shown in this paper, the FCSS algorithm was applied with fuzzifier $m = 2$. The value of ε_1 and ε_2 used was 2 (see (24)). This means that two clusters are considered compatible only if their centers and radii are within two pixels. The value of p in (21) was 2 and the value for η in (26) was 3. The value of $K_{\max}$ was 14 in all cases. The number of points that was used as the upper bound to determine spurious clusters varied between 5 and 90, depending on whether the data set was synthetic, real, or three-dimensional. The threshold, T_D, on the cluster partition density also varied between 200 and 800 depending on whether the data set was synthetic, real, or three-dimensional. However, it is possible to pick these thresholds automatically, and we are currently looking into this issue.

We first show the results on the synthetic two-dimensional data sets. In our illustrations depicting final results, different symbols are used to represent feature points belonging to different clusters, after each point is assigned to the cluster with highest membership. The first example, in Fig. 1(a), shows the result of the fuzzy UCSS clustering of three semicircles contaminated by noise. This example illustrates that the algorithm is successful even when only parts of circles are present. The second example (Fig. 1(b)) consists of two
concentric circles contaminated by a few noise points. This is an example where conventional clustering methods fail miserably. The fuzzy UCSS algorithm finds the correct number of clusters $K = 2$. As seen in Fig. 1(b), the twoconcentric circles are correctly classified, and each of the noise points

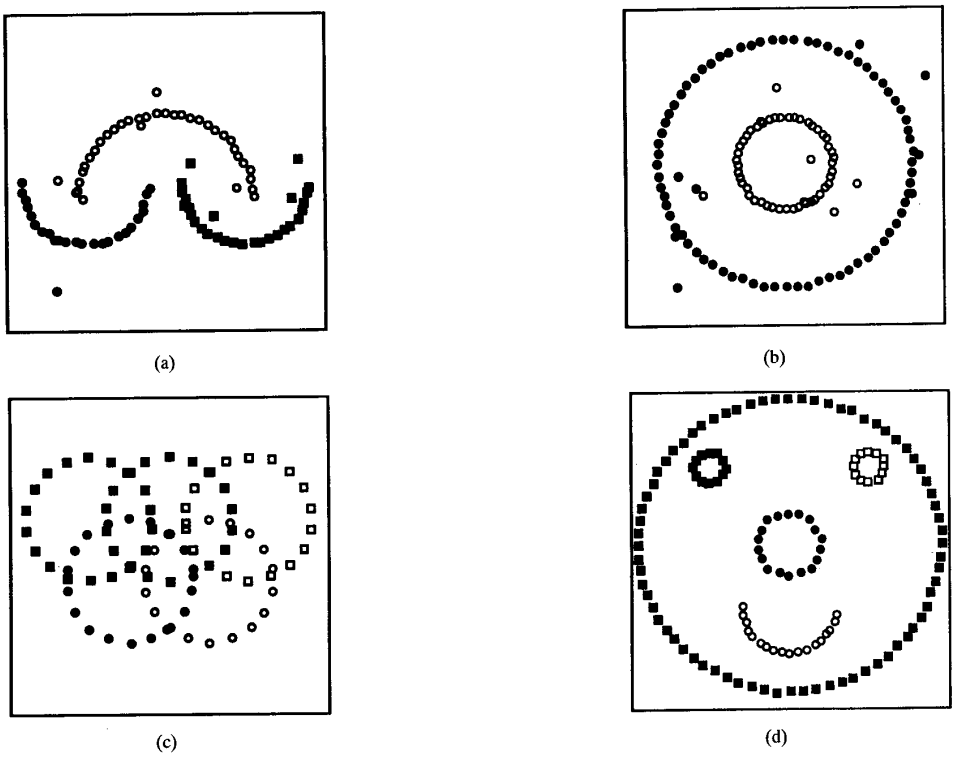


Fig. 1. Results of the unsupervised FCS algorithm (a) for three semicircles, (b) for two concentric circles, (c) for five sparsely sampled circles, and (d) for "Smiley."

TABLE I
TRUE AND ESTIMATED PARAMETERS FOR THE CIRCLES IN FIG. 1(a)

	Generating Parameters		Fitting Parameters	
	Center	Radius	Center	Radius
Cluster 1	(100, 120)	55	(99.2, 119.7)	55.1
Cluster 2	(50, 110)	40	(49.6, 110.1)	39.7
Cluster 3	(150, 110)	40	(149.5, 111.7)	38.5

TABLE II
TRUE AND ESTIMATED PARAMETERS FOR THE CIRCLES IN FIG. 1(b)

	Generating Parameters		Fitting Parameters	
	Center	Radius	Center	Radius
Cluster 1	(100, 100)	30	(99.4, 99.7)	31.0
Cluster 2	(100, 100)	80	(100.2, 99.4)	80.5

TABLE III
TRUE AND ESTIMATED PARAMETERS FOR THE CIRCLES IN FIG. 1(c)

	Generating Parameters		Fitting Parameters	
	Center	Radius	Center	Radius
Cluster 1	(125, 120)	40	(124.6, 119.8)	40.0
Cluster 2	(75, 120)	40	(74.3, 119.3)	40.0
Cluster 3	(50, 80)	40	(49.3, 79.5)	39.9
Cluster 4	(100, 80)	40	(99.9, 79.3)	40.2
Cluster 5	(150, 80)	40	(149.6, 79.5)	40.1

TABLE IV
TRUE AND ESTIMATED PARAMETERS FOR THE CIRCLES IN FIG. 1(d)

	Generating Parameters		Fitting Parameters	
	Center	Radius	Center	Radius
Cluster 1	(100, 140)	30	(99.4, 140.1)	29.7
Cluster 2	(100, 100)	20	(99.6, 99.6)	20.1
Cluster 3	(100, 100)	95	(99.6, 99.5)	94.9
Cluster 4	(50, 50)	10	(49.5, 49.3)	10.0
Cluster 5	(150, 50)	10	(149.6, 49.5)	9.9

is assigned to the closest cluster. The third example consists of five sparsely sampled overlapping circles. The fuzzy UCSS clustering with the correct number of clusters is shown in Fig. 1(c). This is a very difficult case, because the circles are very sparse, truly entangled, and the initial partition obtained using the fuzzy c means algorithm is quite wrong. Fig. 1(d) shows the fuzzy UCSS clustering of a "face" ("Smiley"). This is a challenging example because the clusters have wide-ranging radii, and the outer cluster completely encloses all the remaining clusters. Thus, a truly global search for clusters is required. The final result is $K = 5$, as shown in Fig. 1(d).

Tables I through IV show the centers and radii used to generate the circles of the two-dimensional data examples in

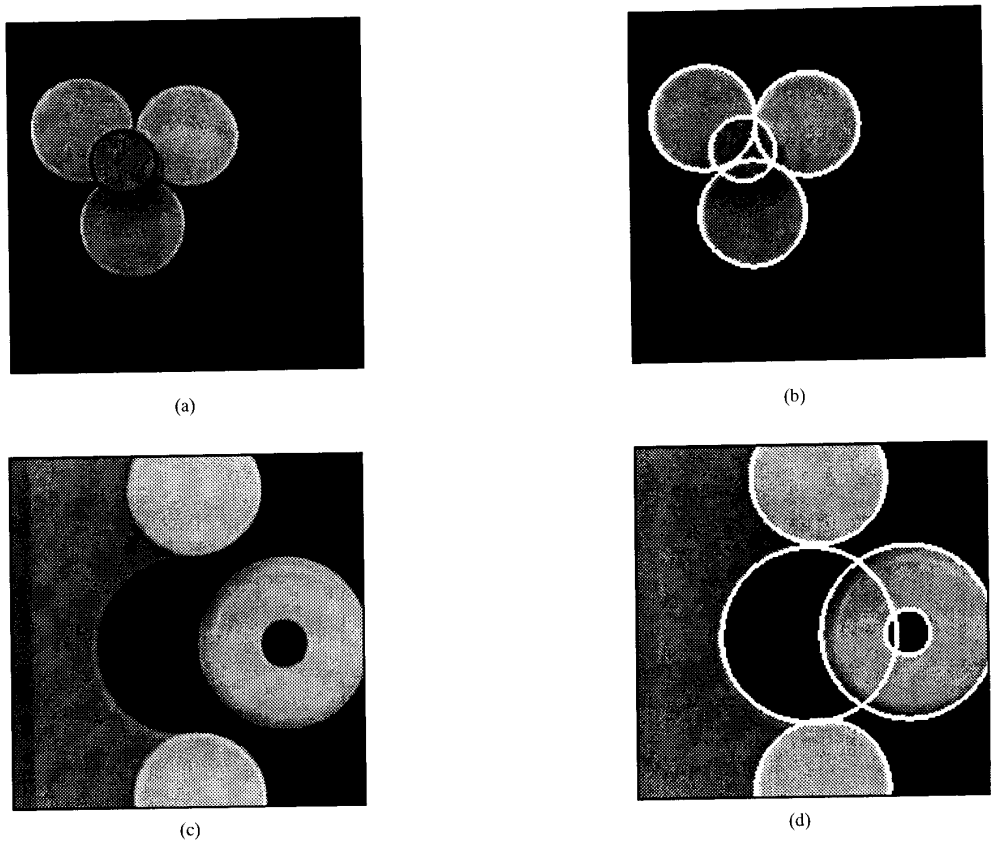


Fig. 2. (a) An image of a collection of cylinders; (b) result of the fuzzy UCSS algorithm; (c) an image of a collection of toys; and (d) result of the fuzzy UCSS algorithm.

Fig. 1, and the corresponding centers and radii as estimated by the fuzzy UCSS algorithm. It can be seen that the algorithm not only clusters the data, but also finds the parameters of the shell clusters with good accuracy.

Figs. 2 and 3 show two-dimensional examples involving real 200×200 images. The fuzzy UCSS algorithm was applied to the thresholded edge images. (The edges were obtained by applying a Sobel operator.) The prototype circles estimated by the algorithm are shown superimposed on the original image. The circles are drawn three pixels thick for emphasis. Fig. 2(a) shows a top-down view of a collection of cylinders. The fuzzy UCSS algorithm correctly clusters this data as shown in Fig. 2(b). Parts (c) and (d) of Fig. 2 show a collection of toys with (partial) circular edges that are clustered correctly. Fig. 3(a) shows three overlapping pulleys. The fuzzy UCSS algorithm clusters all the circles in the pulleys correctly, and finds the right number of clusters ($K = 6$). The result is shown in Fig. 3(b). Parts (c) and (d) of Fig. 3 show pulleys which are clustered correctly with $K = 8$.

Two three-dimensional examples involving synthetic range images are shown in Fig. 4. In these examples, points belonging to different clusters are shown in different gray levels. The first example, in Fig. 4(a), consists of three spherical urns, each inside another. The fuzzy UCSS algorithm clusters the shells correctly and finds the correct number of clusters, $K = 3$, as seen in Fig. 4(b). The second example, in Fig. 4(c), consists of three assorted spheres inside a partial spherical shell. The result of the fuzzy UCSS clustering is shown in Fig. 4(d).

Tables V and VI show the centers and radii that were used to synthetically generate the spherical clusters in the three-dimensional data examples shown in Fig. 4. The centers and radii estimated by the unsupervised algorithm are also shown. As can be seen, the estimates are very accurate.

The CPU time required on a SUN 4 workstation to run the fuzzy UCSS algorithm ranged from 4 s to over 50 s, depending on the complexity of the data set. The plain FCSS algorithm (when K is known) typically takes only a few seconds.

V. Conclusions

In this paper, we introduced a new approach to the fuzzy c spherical shells algorithm, which seeks clusters that can be approximated by hyperspherical shells. Our version of the algorithm does not involve solving coupled nonlinear equations, and hence is implementationally more attractive than other clustering algorithms that have been suggested in the literature for this purpose. We also presented an unsupervised c spherical shells algorithm which automatically

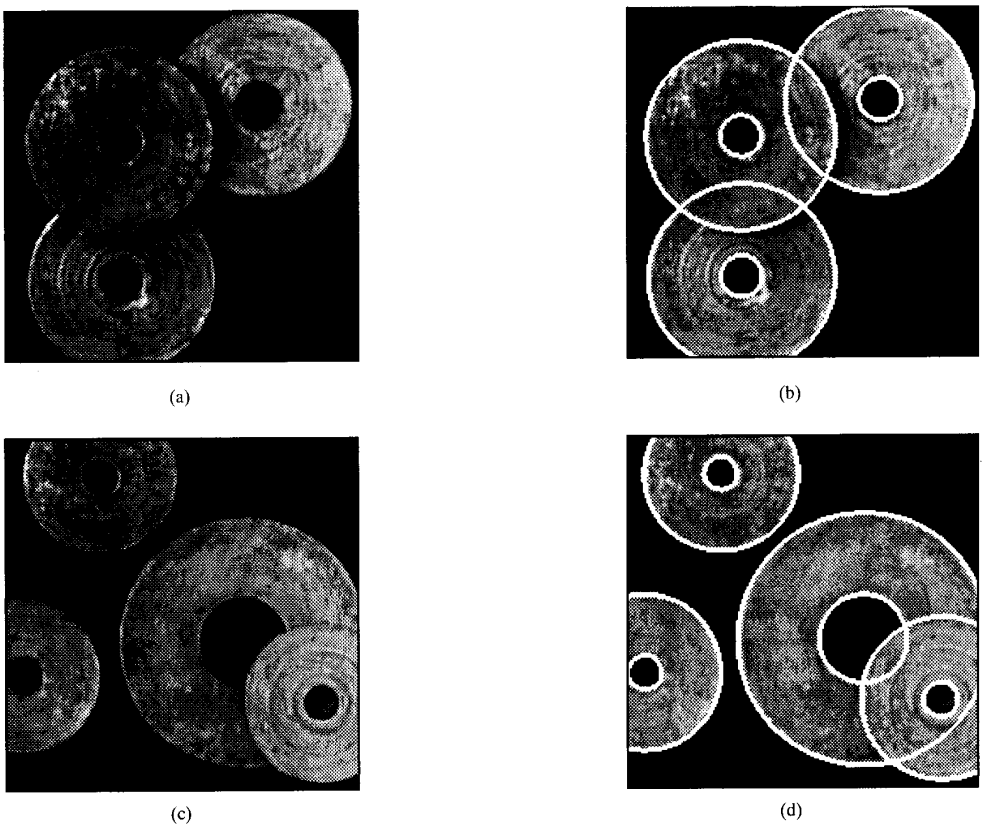


Fig. 3. (a) An image of three overlapping pulleys; (b) result of the fuzzy UCSS algorithm; (c) an image of four pulleys; and (d) result of the fuzzy UCSS algorithm.

TABLE V
TRUE AND ESTIMATED PARAMETERS FOR THE SPHERES IN FIG. 4(a)

	Generating Parameters		Fitting Parameters	
	Center	Radius	Center	Radius
Cluster 1	(120, 120, 100)	40	(120.1, 120.8, 99.9)	39.9
Cluster 2	(120, 120, 100)	100	(120.1, 121.1, 100.2)	99.8
Cluster 3	(120, 120, 100)	70	(120.0, 120.6, 99,8)	69.9

TABLE VI
TRUE AND ESTIMATED PARAMETERS FOR THE SPHERES IN FIG. 4(c)

	Generating Parameters		Fitting Parameters	
	Center	Radius	Center	Radius
Cluster 1	(100, 100, 100)	100	(100.0, 100.5, 99.9)	99.6
Cluster 2	(80, 160, 140)	25	(80.1, 160.0, 141.1)	26.1
Cluster 3	(105, 130, 140)	15	(105.0, 130.0, 140.7)	15.7
Cluster 4	(120, 160, 140)	20	(120.1, 160.1, 140.7)	20.8

finds the optimum number of hyperspherical clusters when this information is not known. The unsupervised algorithm is based on merging compatible clusters and eliminating spurious clusters. We have suggested several validity measures that are attractive for the purpose of identifying good spherical shell clusters. Experimental results on a variety of synthetic and real data sets demonstrate that the algorithms are effective. More recently, we have formulated a more general version of the hard and fuzzy CSS algorithms which is applicable to all second-degree curves [11].

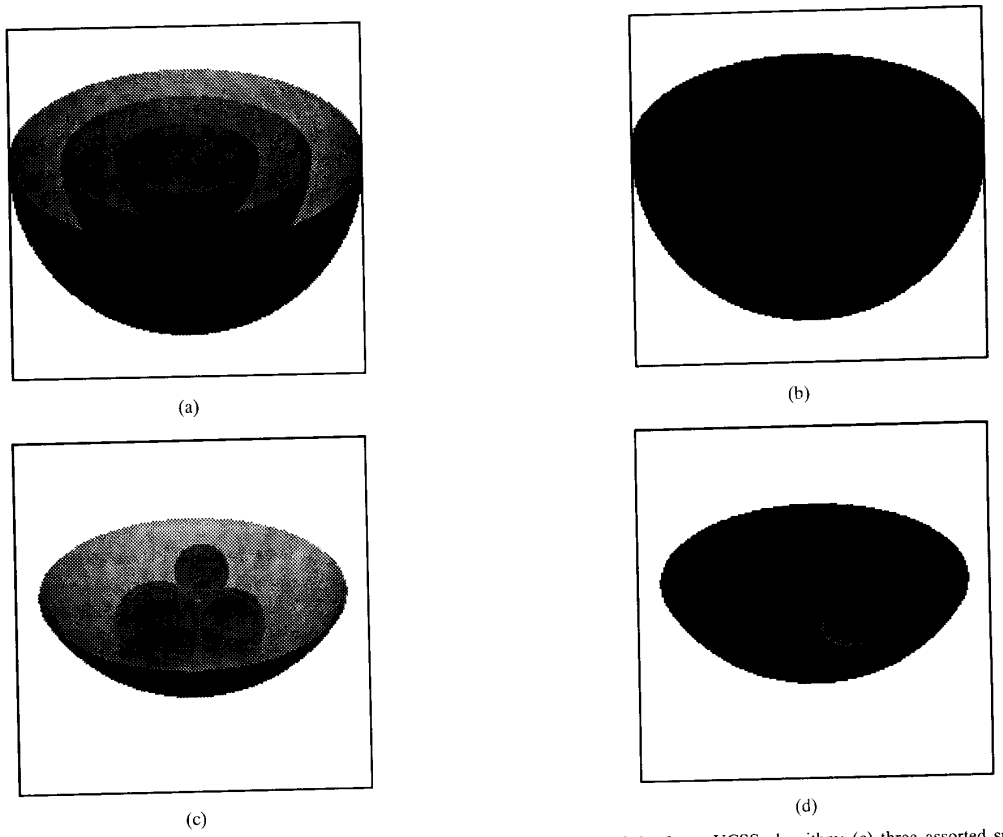


Fig. 4. Three-dimensional examples: (a) three concentric hemispherical shells; (b) result of the fuzzy UCSS algorithm; (c) three assorted spherical shells inside a partial spherical shell; and (d) result of the fuzzy UCSS algorithm.

REFERENCES

[1] R. N. Dave and S. K. Bhamidipati, "Application of the fuzzy-shell clustering algorithm to recognize circular shapes in digital images," in *Proc. Int. Fuzzy Systems Association Congress* (Seattle), 1989, pp. 238–241.

[2] R. N. Dave, "Fuzzy-shell clustering and applications to circle detection in digital images," *Int. J. General Systems,* vol. 16, pp. 343–355, 1990.

[3] R. N. Dave and K. J. Patel, "Fuzzy ellipsodial-shell clustering algorithm and detection of ellipsoidal shapes," in *Proc. SPIE Conf. Intelligent Robots and Computer Vision IX: Algorithms and Techniques* (Boston), Nov. 1990, pp. 320–333.

[4] R. N. Dave, "Adaptive C-shells clustering," in *Proc. North American Fuzzy Information Processing Society Workshop* (Columbia, MO), 1991, pp. 195–199.

[5] J. C. Bezdek and R. J. Hathaway, "Accelerating convergence of the fuzzy c-shells clustering algorithms," in *Proc. Int. Fuzzy Systems Association Congress* (Brussels), July 1991 (volume on *Mathematics*), pp. 12–15.

[6] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms.* New York: Plenum Press, 1981.

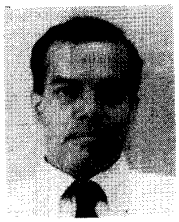
[7] R. Dubes and A. K. Jain, *Algorithms for Clustering Data.* Englewood Cliffs, NJ: Prentice-Hall, 1988.

[8] I. Gath and A. Geva, "Unsupervised optimal fuzzy clustering," *IEEE Trans. Pattern Anal. Mach. Intell.,* vol. 11, pp. 773–781, 1989.

[9] R. N. Dave and K. J. Patel, "Progressive fuzzy clustering algorithms for characteristic shape recognition," in *Proc. North American Fuzzy Information Processing Society Workshop* (Toronto), 1990, pp. 121–124.

[10] R. N. Dave, "New measures for evaluating fuzzy partitions induced through c-shells clustering," in *Proc. SPIE Conf. Intelligent Robots and Computer Vision X: Algorithms and Techniques,* (Boston), Nov. 1991, pp. 406–414.

[11] R. Krishnapuram, H. Frigui, and O. Nasraoui, "New fuzzy shell clustering algorithms for boundary detection and pattern recognition," in *Proc. SPIE Conf. Intelligent Robots and Computer Vision X: Algorithms and Techniques* (Boston), Nov. 1991, pp. 458–465.



Raghu Krishnapuram (S'84–M'87) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Bombay, in 1978. He was with the Research and Development Department (audio entertainment equipment), Bush India Ltd., Bombay, and later with Bharat Electronics Ltd. (sonar equipment), Bangalore, India. He obtained the M.S. degree in electrical engineering from Louisiana State University, Baton Rouge, in 1985 and the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, in 1987.

He is currently an Associate Professor at the University of Missouri, Columbia. His research encompasses all aspects of computer vision, and his current interests include applications of fuzzy set theory, morphology, and neural networks.

Dr. Krishnapuram is a member of SPIE and NAFIPS.

Olfa Nasraoui (S'91) was born in Tunis, Tunisia, in 1968. She received the B.S. degree in electrical and computer engineering from the University of Missouri, Columbia, in 1990. She is currently pursuing the M.S. degree in electrical engineering there, where she also holds a position as research assistant in the Computer Vision Laboratory. Her research interests include pattern recognition, computer vision, neural networks, and applications of fuzzy set theory.

Hichem Frigui (S'91) was born in Monastir, Tunisia, in 1968. He graduated with a B.S. degree in electrical and computer engineering in 1990 from the University of Missouri, Columbia. He is currently pursuing the M.S. degree in the Department of Electrical Engineering there, where he is also a research and teaching assistant. His research interests include pattern recognition, computer vision, fuzzy set theory, and artificial intelligence.