Possibilistic Shell Clustering of Template-Based Shapes

Tsaipei Wang, *Member, IEEE*

Abstract—**In this paper, we present a new type of alternating-optimization-based possibilistic c-shell algorithm for clustering-template-based shapes. A cluster prototype consists of a copy of the template after translation, scaling, rotation, and/or affine transformations. This extends the capability of shell clustering beyond a few standard geometrical shapes that have been in the literature so far. We use a number of 2-D datasets, consisting of both synthetic and real-world images, to illustrate the capability of our algorithm in detecting generic-template-based shapes in images. We also describe a progressive clustering procedure aimed to relax the requirements for a known number of clusters and good initialization, as well as new performance measures of shell-clustering algorithms.**

Index Terms—**Alternating optimization (AO), object detection, possibilistic clustering, progressive clustering, shape detection, shell clustering, template matching.**

I. Introduction

PROTOTYPE-based fuzzy and possibilistic clustering algorithms have had broad applications in many pattern recognition problems. In these algorithms, each cluster is represented by a prototype, and the clustering procedure usually involves the iterative minimization of an objective function through the adjustment of the prototype parameters and the memberships of the feature points in individual clusters. Fuzzy c-means (FCM) [1], [2] and possibilistic c-means (PCM) [3]–[6] are well-known and widely used representatives of these algorithms. The most common algorithms of this class, including FCM and PCM, are intended for detecting “compact” or “filled” clusters in the data, such as filled hyperspheres, filled hyperellipsoids [7], or filled convex polytopes [8]. Although the resulting clusters can have different shapes, they all correspond to regions of high data concentration in the feature space.

Shell clustering is a category of prototype-based clustering algorithms that use prototypes that are “shells” in the feature space. The earliest shell clustering algorithms involve the detection of lines [9] or circles [10] through fuzzy clustering. One of the difficulties of the clustering of circles is the need to numerically solve coupled nonlinear equations for updating the prototypes. This usually results in a significantly increased computational cost. An analysis of this issue is given in [11]. A computationally simpler approach by separating the update equations for the centers and radii of hyperspherical clusters is given in [12]. Later, a more efficient approach using a non-Euclidean algebraic distance measure is described in [13], although the results are not as satisfactory with highly scattered data. Shell clustering has since been studied extensively for general quadratic shells such as ellipses and hyperbola [14]–[19]. Later additions to shell-clustering algorithms include proposed methods for clustering rectangles [20] and template-based shapes [21].

The ability to efficiently detect shell-like structures of particular shapes is useful in many image processing and computer-vision applications. There have been an increasing number of applications of shell clustering in analyzing real-world images [22]–[26]. According to [17], fuzzy and possibilistic shell clustering has a number of advantages compared with the generalized Hough transform [27], [28] in detecting particular shapes; it is more computationally efficient without the large memory requirement of Hough transform, it is less sensitive to noise and zigzagged edges, and the parameter resolution is not limited by the predefined bin sizes. These properties make shell clustering a very useful option for shape and object detection.

A major limitation on the applicability of existing shell-clustering algorithms is that they are mostly specifically designed for particular shapes. The only one exception is in [21], which attempted to cluster shells of generic-template-based shapes. However, instead of the more efficient and commonly used alternating optimization (AO) approach, [21] relies on a genetic algorithm (GA) to optimize the prototypes. The reason presumably is because GA does not require update equations of the prototype parameters, as the derivation of these equations is intrinsically more difficult for generic-template-based prototypes than for prototypes of simple geometric shapes. This is similar to the use of GA for clustering shell prototypes with even nondifferentiable distance measures [29]. A performance comparison of FCM with point prototypes using only either AO or GA can be found in [30].

Our main contribution in this paper includes new AO-based clustering algorithms for the detection of generic-template-based shell clusters. This extends the applicability of efficient AO-based shell clustering beyond the detection of just a few basic geometric shapes. The prototypes are obtained through geometric transformations of the template. By relaxing the usual requirement for updating all prototype parameters simultaneously, we are able to obtain closed-form update equations, making the process very efficient. This approach is more similar to [12] and differs from those relying on numerical methods (e.g., [10]) or non-Euclidean distance measures (e.g., [13]) in order to solve their prototype update equations. In this paper, we only focus on 2-D clustering data that has object and shape detection in images as the target application.

Manuscript received July 12, 2007; revised December 1, 2007; accepted January 17, 2008. First published April 30, 2008; current version published July 29, 2009. This work was supported in part by the National Science Council of Taiwan under Grant NSC-96-2221-E-009-192.

The author is with the Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan, R.O.C. (e-mail: wangts@cs.nctu.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TFUZZ.2008.924360

The rest of this paper is organized as follows. In Section II, we briefly review the common prototype-based fuzzy and possibilistic clustering procedures. Section III contains the main definitions of template-based shell clustering and the derivation of the prototype update equations for three types of prototype transformations. Section IV describes our progressive clustering procedure, which is important to make the clustering results robust against unknown number of clusters and initialization. We present our experimental results in Section V, including synthetic datasets and real images. A measure for quantifying cluster-detection performances is also described. Section VI concludes the paper.

II. Overview of Fuzzy and Possibilistic Clustering

The following is the most commonly used objective function for various types of FCM and fuzzy c-shells clustering algorithms [1]:

$$J = \sum_{j=1}^{C}\sum_{i=1}^{N} u_{ij}^{m} d_{ij}^{2} + \sum_{j=1}^{C}\eta_j \sum_{i=1}^{N}(1-u_{ij})^{m} \tag{1}$$

where *N* is the number of data points, *C* is the number of clusters, *m* is the fuzzification factor, u_{ij} is the membership of $\boldsymbol{x}_i$ (the *i*th sample point, $1 \leq i \leq N$) in the *j*th cluster, and d_{ij} is the distance between $\boldsymbol{x}_i$ and the *j*th cluster prototype. The memberships need to satisfy the condition that

$$\forall i, \qquad \sum_{j=1}^{C} u_{ij} = 1. \tag{2}$$

The standard AO scheme involves iteratively solving the equations that correspond to the necessary conditions of local minima of (1): solving $\partial J/\partial u_{ij} = 0$ for the update equation for u_{ij} and solving $\partial J/\partial \boldsymbol{\theta}_j = 0$ for the update equation for $\boldsymbol{\theta}_j$. Here, $\boldsymbol{\theta}_j$ represents the set of parameters that define the *j*th cluster prototype included in the objective function through d_{ij}. The solution for u_{ij} is given by

$$u_{ij} = \left[\sum_{k=1}^{C}\left(\frac{d_{ij}}{d_{ik}}\right)^{2/(m-1)}\right]^{-1}. \tag{3}$$

The solutions for the prototype parameters depend on the type of prototypes and the distance measure used.

The following is the most common objective function used for various types of possibilistic c-means and c-shell clustering algorithms [3]:

$$J = \sum_{j=1}^{C}\sum_{i=1}^{N} u_{ij}^{m} d_{ij}^{2} + \sum_{j=1}^{C}\eta_j \sum_{i=1}^{N}(1-u_{ij})^{m}. \tag{4}$$

Here, the first term is the same as the objective function of FCM, but the memberships no longer need to satisfy (2). The new parameter in the second term, η_j, is termed the "bandwidth" or "zone of influence" in [3] and controls the dependence of u_{ij} on d_{ij}. This term has the effect of preventing the trivial solution of all zero memberships if we try to minimize (1) without the constraints (2). According to [3], the solution for u_{ij} is given by

$$u_{ij} = \left[1 + \left(\frac{d_{ij}^{2}}{\eta_j}\right)^{1/(m-1)}\right]^{-1}. \tag{5}$$

Since the second term in (4) does not involve the prototype parameters $\boldsymbol{\theta}_j$, both fuzzy and possibilistic versions will result in the same update equations for the prototype parameters.

Some comparisons between fuzzy and possibilistic clustering results for quadratic shell clusters are given in [17]. In general, possibilistic clustering has the advantage of being more robust against noise or outliers and, for shell clustering, can yield better clusters in noisy datasets. However, possibilistic clustering is also very sensitive to initialization. The approach in [17] is to use the fuzzy version to initialize the possibilistic version of the clustering algorithm.

The general form of prototype-based fuzzy and possibilistic clustering with AO is given as follows:

```
Initialize the prototypes
Repeat
    Update the membership values
    Update the prototype parameters
Until convergence or the maximal allowed iterations
```

We do not use the common form that starts with the initialization of the partition (memberships) mainly because the derivation of prototype parameters from memberships alone is a complex task for shell clusters of generic shapes. The general form proposed here allows us to include the prototype parameters of the previous iteration in the computation of their values for the current iteration. The detail is covered in Section III.

The previous general form requires a prespecified number of clusters (i.e., a fixed *C*), which, for many applications, is unknown initially. For compact clusters, this is usually handled by clustering using a range of different *C*. However, this approach is not reliable for shell clustering mainly because of the higher tendency of the algorithm to converge to local optima, making it more difficult to meaningfully compare the results obtained with different *C*. This issue is handled with progressive clustering in [13] and [18], which only requires an over-specified $C_{\max}$ to start with. We use a completely possibilistic version of progressive clustering discussed in Section IV that handles both initialization and the unknown number of clusters simultaneously.

III. Template-Based Shell Clustering

A. Definition of Templates

We start here by first defining the templates themselves. We present two ways of defining a template: point-based and edge-based. A point-based template consists of only a set of N_p points (referred to as "template points" later):

$$T = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_{N_p}\}. \tag{6}$$

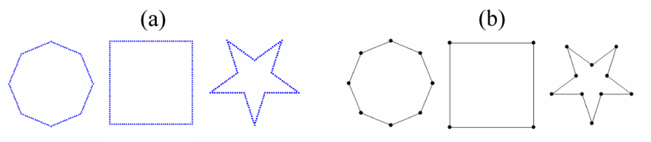

Fig. 1. Examples of templates. (a) Point-based templates. (b) Edge-based templates.

An edge-based template is defined as a set of vertices and edges that connect them:

$$T = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_{N_v}; \boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_{N_e}\}. \tag{7}$$

Here, each $\boldsymbol{v}_k$ $(1 \le k \le N_v)$ is a vertex, each $\boldsymbol{e}_k$ $(1 \le k \le N_e)$ is an edge, and N_v and N_e are the numbers of vertices and edges, respectively. Fig. 1 displays a few examples of both point-based and edge-based templates of different shapes.

The prototype of a cluster is a transformed version of the template. Therefore, the prototype parameter set $\boldsymbol{\theta}_j$ actually consists of the parameters that define the transformation. Let us state this relation as

$$P_j = H(T; \boldsymbol{\theta}_j) \tag{8}$$

with P_j being the *j*th cluster prototype. Here *H* represents the transformation that is actually applied to the template points of point-based templates or the vertices of edge-based templates. A point $\boldsymbol{p}$ on P_j is related to its corresponding point in the template $\boldsymbol{p}^*$ according to

$$\boldsymbol{p} = H(\boldsymbol{p}^*; \boldsymbol{\theta}_j) \qquad \text{and} \qquad \boldsymbol{p}^* = H^{-1}(\boldsymbol{p}; \boldsymbol{\theta}_j). \tag{9}$$

The distance between any point $\boldsymbol{q}$ and P_j is defined as the following:

1) Point-based templates:

$$\text{dist}(\boldsymbol{q}, P_j) \equiv \min_{\boldsymbol{v} \in P_j} \text{dist}(\boldsymbol{q}, \boldsymbol{v}) = \min_{\boldsymbol{v} \in T} \text{dist}(\boldsymbol{q}, H(\boldsymbol{v}; \boldsymbol{\theta}_j)). \tag{10}$$

2) Edge-based templates:

$$\text{dist}(\boldsymbol{q}, P_j) \equiv \min_{\boldsymbol{e} \in P_j} \text{dist}(\boldsymbol{q}, \boldsymbol{e}) = \min_{\boldsymbol{e} \in T} \text{dist}(\boldsymbol{q}, H(\boldsymbol{e}; \boldsymbol{\theta}_j)). \tag{11}$$

These are just the shortest distances between $\boldsymbol{q}$ and any of the transformed template points or edges of P_j, respectively. The transformation of an edge is just the edge that connects the corresponding transformed vertices. We use Euclidean distance for all the subsequent derivations.

We also define the concept of the matching point of $\boldsymbol{q}$ in P_j as the point in P_j that is closest to $\boldsymbol{q}$. For point-based templates, the matching point is always one of the transformed template points. For edge-based templates, the matching point can be a transformed vertex or a point on an edge. For both types of templates, dist($\boldsymbol{q}$,P_j) is just the distance between $\boldsymbol{q}$ and its matching point in P_j. The computation of matching points is a required but most time-consuming step in our algorithms. Since there are usually far less edges in an edge-based template than there are points in a point-based template that represents the same shape, our experiments are focused on cases with edge-based templates. Even so, all the derivations in this section are still applicable if point-based templates are used.

In addition, we also define difference measures between two prototypes. Such difference measures can be used in, for example, the test of convergence. For point-based templates, this is given by

$$\text{diff}(P_i, P_j) = \max_{\boldsymbol{v} \in P_i} \min_{\boldsymbol{v}' \in P_j} \text{dist}(\boldsymbol{v}, \boldsymbol{v}'). \tag{12}$$

The difference measure for edge-based templates is more complicated. Our previous version in [31] is given by

$$\text{diff}(P_i, P_j) = \max \left[\max_{\boldsymbol{v} \in P_i} \min_{\boldsymbol{e} \in P_j} \text{dist}(\boldsymbol{v}, \boldsymbol{e}), \max_{\boldsymbol{v} \in P_j} \min_{\boldsymbol{e} \in P_i} \text{dist}(\boldsymbol{v}, \boldsymbol{e}) \right] \tag{13}$$

with dist($\boldsymbol{v}$,$\boldsymbol{e}$) being the Euclidean distance between a vertex $\boldsymbol{v}$ and an edge $\boldsymbol{e}$. We later discovered that, while this formula works most of the time, it can fail in rare cases for some particular templates, resulting in much smaller than actual differences. Since in the subsequent algorithms, the computed prototype-prototype differences are always compared with some thresholds (tests for convergence and for prototype merging as discussed in Section IV), we continue to use (13) as a screener. If the difference given by (13) is less than the given threshold, we compose a corresponding point-based prototype of the same shape for each of the two prototypes. This is implemented by placing points along the edges such that the distances between adjacent points are not larger than the given threshold. Let $P_{j_{(p)}}$ represent the corresponding point-based prototype of an edge-based prototype P_j. A new difference measure is now computed according to

$$\text{diff}(P_i, P_j) = \max \left[\max_{\boldsymbol{v} \in P_{i(p)}} \min_{\boldsymbol{e} \in P_j} \text{dist}(\boldsymbol{v}, \boldsymbol{e}), \max_{\boldsymbol{v} \in P_{j(p)}} \min_{\boldsymbol{e} \in P_i} \text{dist}(\boldsymbol{v}, \boldsymbol{e}) \right] \tag{14}$$

and again checked against the threshold. While (14) is much more time-consuming to compute than (13) due to the increased number of points, it is only used sparingly and, therefore, does not add significantly to the computational requirement.

While there are many possible transformations to derive a cluster prototype from a template, to limit the complexity of the problem, in this paper, we present three types of transformations, termed types I, II, and III, next.

1) *Type I:* A Type I transformation is a shape-preserving transformation consisting of translation, rotation, and a single scalar scaling factor. A point $\boldsymbol{p}$ on P_j is related to its corresponding point $\boldsymbol{p}^*$ in the template according to

$$\boldsymbol{p} = H(\boldsymbol{p}^*; \boldsymbol{\theta}_j) = \boldsymbol{R}_j s_j \boldsymbol{p}^* + \boldsymbol{t}_j. \tag{15}$$

Here, s_j, $\boldsymbol{R}_j$, and $\boldsymbol{t}_j$ are the scalar scaling factor, rotation matrix, and translation vector of P_j, respectively. For 2-D data, this means that there are four adjustable parameters for each prototype. The variable $\boldsymbol{R}_j$ is the 2 × 2 rotation matrix determined by the rotation angle φ_j:

$$\boldsymbol{R}_j = \begin{pmatrix} \cos\varphi_j & -\sin\varphi_j \\ \sin\varphi_j & \cos\varphi_j \end{pmatrix}. \tag{16}$$

2) *Type II:* As an extension of type I transformations, type II transformations allow a separate scaling factor for each

dimension in the coordinate system in which the template is defined. A point $\boldsymbol{p}$ on P_j is related to its corresponding point $\boldsymbol{p}^*$ in the template according to

$$\boldsymbol{p} = H(\boldsymbol{p}^*; \boldsymbol{\theta}_j) = \boldsymbol{R}_j \boldsymbol{S}_j \boldsymbol{p}^* + \boldsymbol{t}_j. \tag{17}$$

Here, $\boldsymbol{R}_j$ and $\boldsymbol{t}_j$ are the rotation matrix and translation vector of P_j, respectively. $\boldsymbol{S}_j$, the "scaling matrix," is a diagonal matrix with its *k*th diagonal element representing the scaling factor for the *k*th dimension in the template coordinates. This allows, for example, rectangular prototypes of various aspect ratios, given a square template. For 2-D data, there are five adjustable parameters for each prototype.

3) *Type III:* Type III transformations allow more flexibility by replacing rotation and scaling transformations with a single affine transformation. A point $\boldsymbol{p}$ on P_j is related to its corresponding point $\boldsymbol{p}^*$ in the template according to

$$\boldsymbol{p} = H(\boldsymbol{p}^*; \boldsymbol{\theta}_j) = \boldsymbol{A}_j \boldsymbol{p}^* + \boldsymbol{t}_j. \tag{18}$$

Here, $\boldsymbol{A}_j$ and $\boldsymbol{t}_j$ are the affine transform matrix and translation vector of P_j, respectively. For 2-D data, there are six adjustable parameters for each cluster prototype.

In order to simplify notation, we do not distinguish between the different types of transformations in terms of the notations used before. The correct transformation type referred to later in this paper should always be clear from the context. More details and derivations of the respective prototype update equations of the three types of transformations are provided in the next three sections. The update equation for the cluster memberships u_{ij} is the same as (5) for all three types of transformations.

B. Clustering Procedure for Type I Transformations

Our first goal here is to derive the update equations of the prototype parameters. Let $\boldsymbol{p}_{ij}$ be the matching point of $\boldsymbol{x}_i$ on P_j, and let d_{ij} be the distance between $\boldsymbol{x}_i$ and P_j. For type I transformations, d_{ij} is given by

$$d_{ij}^2 = \left\|\boldsymbol{x}_i - \boldsymbol{p}_{ij}\right\|^2 = \left\|\boldsymbol{x}_i - \left(\boldsymbol{R}_j s_j \boldsymbol{p}_{ij}^* + \boldsymbol{t}_j\right)\right\|^2. \tag{19}$$

The necessary conditions for minimizing *J* with respect to the cluster parameters are obtained by setting to zero the partial derivatives of *J* with respect to φ_j, s_j, and $\boldsymbol{t}_j$, using (19) as the distance measure. The resulting equations are as follows:

$$\frac{\partial J}{\partial \boldsymbol{t}_j} = \sum_{i=1}^{N} (2u_{ij}^m)(\boldsymbol{R}_j s_j \boldsymbol{p}_{ij}^* + \boldsymbol{t}_j - \boldsymbol{x}_i) = 0 \tag{20}$$

$$\frac{\partial J}{\partial s_j} = \sum_{i=1}^{N} (2u_{ij}^m)(\boldsymbol{R}_j s_j \boldsymbol{p}_{ij}^* + \boldsymbol{t}_j - \boldsymbol{x}_i)^T (\boldsymbol{R}_j \boldsymbol{p}_{ij}^*) = 0 \tag{21}$$

and

$$\frac{\partial J}{\partial \varphi_j} = \sum_{i=1}^{N} (2u_{ij}^m) s_j (\boldsymbol{R}_j s_j \boldsymbol{p}_{ij}^* + \boldsymbol{t}_j - \mathbf{x}_i)^T \left(\frac{d\boldsymbol{R}_j}{d\varphi_j} \mathbf{p}_{ij}^*\right) = 0. \tag{22}$$

It is difficult to find analytical solutions of all three parameters that simultaneously satisfy (20)–(22). However, we can find closed-form expressions if we choose to update one parameter at a time. Specifically, this means that we solve (20) for $\boldsymbol{t}_j$, (21) for s_j, and (22) for φ_j.

It is straightforward to obtain the following solution of $\boldsymbol{t}_j$ from (20):

$$\boldsymbol{t}_j = \frac{\sum_{i=1}^N u_{ij}^m (\boldsymbol{x}_i - \boldsymbol{p}_{ij})}{\sum_{i=1}^N u_{ij}^m}. \tag{23}$$

An interesting observation is that when there is only one vertex and no edge in the template, resulting in $\boldsymbol{p}_{ij}^* = \boldsymbol{0}$, (23) reduces to the update equation for point prototypes in the standard FCM and PCM algorithms.

It is straightforward to obtain the following solution for s_j from (21):

$$\begin{aligned} s_j &= \frac{\sum_{i=1}^N u_{ij}^m (\boldsymbol{x}_i - \boldsymbol{t}_j)^T (\boldsymbol{R}_j \boldsymbol{p}_{ij}^*)}{\sum_{i=1}^N u_{ij}^m (\boldsymbol{R}_j \boldsymbol{p}_{ij}^*)^T (\boldsymbol{R}_j \boldsymbol{p}_{ij}^*)} \\ &= \frac{\sum_{i=1}^N u_{ij}^m (\boldsymbol{x}_i - \boldsymbol{t}_j)^T (\boldsymbol{R}_j \boldsymbol{p}_{ij}^*)}{\sum_{i=1}^N u_{ij}^m \left\|\boldsymbol{p}_{ij}^*\right\|^2}. \end{aligned} \tag{24}$$

The solutions for φ_j require more derivation. First, we can separate (22) into two terms:

$$\begin{aligned} &\sum_{i=1}^{N} (u_{ij}^m) s_j (\boldsymbol{R}_j \boldsymbol{p}_{ij}^*)^T \left(\frac{d\boldsymbol{R}_j}{d\varphi_j} \boldsymbol{p}_{ij}^*\right) \\ &\quad + \sum_{i=1}^{N} (u_{ij}^m)(\boldsymbol{t}_j - \boldsymbol{x}_i)^T \left(\frac{d\boldsymbol{R}_j}{d\varphi_j} \boldsymbol{p}_{ij}^*\right) = 0. \end{aligned} \tag{25}$$

Here, the common multiplication factor $2s_j$ has been discarded. Since

$$\begin{aligned} &(\boldsymbol{R}_j \boldsymbol{p}_{ij}^*)^T \left(\frac{d\boldsymbol{R}_j}{d\varphi_j} \boldsymbol{p}_{ij}^*\right) \\ &= (\boldsymbol{p}_{ij}^*)^T \begin{pmatrix} \cos\varphi_j & \sin\varphi_j \\ -\sin\varphi_j & \cos\varphi_j \end{pmatrix} \begin{pmatrix} -\sin\varphi_j & -\cos\varphi_j \\ \cos\varphi_j & -\sin\varphi_j \end{pmatrix} \boldsymbol{p}_{ij}^* \\ &= (\boldsymbol{p}_{ij}^*)^T \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \boldsymbol{p}_{ij}^* = 0 \end{aligned} \tag{26}$$

this leaves us with only

$$\sum_{i=1}^{N} (u_{ij}^m)(\boldsymbol{t}_j - \boldsymbol{x}_i)^T \left(\frac{d\boldsymbol{R}_j}{d\varphi_j} \boldsymbol{p}_{ij}^*\right) = 0. \tag{27}$$

Next, let us assume for now that

$$(\boldsymbol{t}_j - \boldsymbol{x}_i) = [\,a_{i1} \quad a_{i2}\,]^T \tag{28}$$

and

$$\boldsymbol{p}_{ij}^* = [\,b_{i1} \quad b_{i2}\,]^T. \tag{29}$$

We can now rewrite (27) as

$$\sum_{i=1}^{N}(u_{ij}^{m})\,[\,a_{i1}\ a_{i2}\,]\begin{pmatrix}-\sin\varphi_j & -\cos\varphi_j\\ \cos\varphi_j & -\sin\varphi_j\end{pmatrix}\begin{bmatrix}b_{i1}\\ b_{i2}\end{bmatrix}$$
$$=\sum_{i=1}^{N}(u_{ij}^{m})\begin{bmatrix}-a_{i1}b_{i1}\ \sin\varphi_j+a_{i2}b_{i1}\ \cos\varphi_j\\ -a_{i1}b_{i2}\ \cos\varphi_j-a_{i2}b_{i2}\ \sin\varphi_j\end{bmatrix}^{T}=0. \quad (30)$$

After reorganizing the terms, we obtain

$$\frac{\sin\varphi_j}{\cos\varphi_j}=\frac{\sum_{i=1}^{N}(u_{ij}^{m})(a_{i2}b_{i1}-a_{i1}b_{i2})}{\sum_{i=1}^{N}(u_{ij}^{m})(a_{i1}b_{i1}+a_{i2}b_{i2})}$$
$$=\frac{\sum_{i=1}^{N}(u_{ij}^{m})(\boldsymbol{t}_j-\boldsymbol{x}_i)^{T}\begin{pmatrix}0 & -1\\ 1 & 0\end{pmatrix}\boldsymbol{p}_{ij}^{*}}{\sum_{i=1}^{N}(u_{ij}^{m})(\boldsymbol{t}_j-\boldsymbol{x}_i)^{T}\boldsymbol{p}_{ij}^{*}}. \quad (31)$$

The solutions for φ_j are now given by the following equation:

$$\varphi_j=\tan^{-1}\left[\frac{\sum_{i=1}^{N}u_{ij}^{m}(\boldsymbol{x}_i-\boldsymbol{t}_j)^{T}\begin{pmatrix}0 & -1\\ 1 & 0\end{pmatrix}\boldsymbol{p}_{ij}^{*}}{\sum_{i=1}^{N}u_{ij}^{m}(\boldsymbol{x}_i-\boldsymbol{t}_j)^{T}\boldsymbol{p}_{ij}^{*}}\right]. \quad (32)$$

Equation (32) gives two values for φ_j over a range of 2π. We handle this by computing the objective function *J* using both values of φ_j, and choose the one that gives the lower *J*.

The resulting clustering algorithm has the same form as the general form in Section II, with the part within the loop replaced by the following statements:

```
Find all $\boldsymbol{p}_{ij}$ and corresponding $\boldsymbol{p}_{ij}^{*}$
Compute $d_{ij}$ using (19)
Update $u_{ij}$ using (5)
Update $\boldsymbol{t}_j$ using (23)
Update $s_j$ using (24)
Update $\varphi_j$ using (32)
```

We use the name PCT-I to collectively represent these statements for later reference.

Within each iteration of AO, the task of updating the prototype parameters is divided into three separate steps for updating $\boldsymbol{t}_j$, s_j, and φ_j. In addition to making closed-form update equations possible, another advantage of this decomposition is the added flexibility to, say, "disable" a type of transformation. For example, if we are looking for circles, we can simply skip the step for updating φ_j because there is no need to rotate the prototypes.

Another issue regarding this algorithm is that, by definition, the points $\boldsymbol{p}_{ij}$ themselves are also dependent on the prototype parameters. However, we are unable to express this dependence in a differentiable function. Our solution is to keep the points $\boldsymbol{p}_{ij}$ unchanged while updating the prototype parameters and recalculate $\boldsymbol{p}_{ij}$ in a separate step within each iteration of AO. Since $\boldsymbol{p}_{ij}$ is selected to minimize d_{ij}, this step is consistent with our goal of minimizing *J*.

C. Clustering Procedure for Type II Transformations

Update equations for type II transformations are mostly similar to those for type I transformations, and therefore, we will focus on what are actually different in this subsection. The distance measure here is given by

$$d_{ij}^{2}=\left\|\boldsymbol{x}_i-\boldsymbol{p}_{ij}\right\|^{2}=\left\|\boldsymbol{x}_i-\left(\boldsymbol{R}_j\boldsymbol{S}_j\boldsymbol{p}_{ij}^{*}+\boldsymbol{t}_j\right)\right\|^{2}. \quad (33)$$

This is very similar to (19), with the only difference being the scalar scaling factor s_j being replaced by the scaling matrix $\boldsymbol{S}_j$. As a matter of fact, since the update equations [(23) and (32)] for $\boldsymbol{t}_j$ and φ_j for type I transformations do not involve scaling factors, they can be used for updating $\boldsymbol{t}_j$ and φ_j for type II transformations without change.

To update $\boldsymbol{S}_j$, we actually derive the equation for updating each of its diagonal elements. Let s_{jk} be the *k*th diagonal element of $\boldsymbol{S}_j$. The necessary condition for minimizing *J* relative to s_{jk} becomes

$$\frac{\partial J}{\partial s_{jk}}=\sum_{i=1}^{N}u_{ij}^{m}\frac{\partial}{\partial s_{jk}}\left\|\boldsymbol{R}_j\boldsymbol{S}_j\boldsymbol{p}_{ij}^{*}+\boldsymbol{t}_j-\boldsymbol{x}_i\right\|^{2}$$
$$=\sum_{i=1}^{N}2u_{ij}^{m}\left[\boldsymbol{R}_j\boldsymbol{S}_j\boldsymbol{p}_{ij}^{*}+\boldsymbol{t}_j-\boldsymbol{x}_i\right]^{T}\left[\boldsymbol{R}_j\frac{\partial(\boldsymbol{S}_j\boldsymbol{p}_{ij}^{*})}{\partial s_{jk}}\right]=0. \quad (34)$$

Let us define $\boldsymbol{p}_{ij(k)}^{*}$ as a vector of the same length as $\boldsymbol{p}_{ij}^{*}$, where its *k*th element is equal to the *k*th element of $\boldsymbol{p}_{ij}^{*}$ (denoted as p_{ijk}^{*}) and all the other elements being zero. Then, we have

$$\frac{\partial(\boldsymbol{S}_j\boldsymbol{p}_{ij}^{*})}{\partial s_{jk}}=\boldsymbol{p}_{ij(k)}^{*}=[0\cdots 0\quad p_{ijk}^{*}\quad 0\ \cdots 0]^{T}. \quad (35)$$

After substituting (35) into (34) and some rearrangement, we end up with

$$\sum_{i=1}^{N}u_{ij}^{m}(\boldsymbol{x}_i-\boldsymbol{t}_j)^{T}\boldsymbol{R}_j\boldsymbol{p}_{ij(k)}^{*}=\sum_{i=1}^{N}u_{ij}^{m}(\boldsymbol{R}_j\boldsymbol{S}_j\boldsymbol{p}_{ij}^{*})^{T}\boldsymbol{R}_j\boldsymbol{p}_{ij(k)}^{*}$$
$$=\sum_{i=1}^{N}u_{ij}^{m}(\boldsymbol{p}_{ij}^{*})^{T}\boldsymbol{S}_j\boldsymbol{R}_j^{T}\boldsymbol{R}_j\boldsymbol{p}_{ij(k)}^{*}$$
$$=\sum_{i=1}^{N}u_{ij}^{m}(\boldsymbol{p}_{ij}^{*})^{T}\boldsymbol{S}_j\boldsymbol{p}_{ij(k)}^{*}=s_{jk}\sum_{i=1}^{N}u_{ij}^{m}(p_{ijk}^{*})^{2} \quad (36)$$

which gives the following update equation for s_{jk}:

$$s_{jk}=\frac{\sum_{i=1}^{N}u_{ij}^{m}(\boldsymbol{x}_i-\boldsymbol{t}_j)^{T}\boldsymbol{R}_j\boldsymbol{p}_{ij(k)}^{*}}{\sum_{i=1}^{N}u_{ij}^{m}(p_{ijk}^{*})^{2}}. \quad (37)$$

The resulting clustering algorithm has the same form as the general form in Section II, with the part within the loop replaced by the following statements:

```
Find all $\boldsymbol{p}_{ij}$ and corresponding $\boldsymbol{p}^*_{ij}$
Compute $d_{ij}$ using (33)
Update $u_{ij}$ using (5)
Update $\boldsymbol{t}_j$ using (23)
Update $s_{jk}$ using (37)
Update $\varphi_j$ using (32)
```

We use the name PCT-II to collectively represent these statements for later reference.

D. Clustering Procedure for Type III Transformations

Again our focus in this section is on what is different from the previous two types of transformations. The distance measure now is given by

$$d_{ij}^2 = \left\|\boldsymbol{x}_i - \boldsymbol{p}_{ij}\right\|^2 = \left\|\boldsymbol{x}_i - \left(\boldsymbol{A}_j\boldsymbol{p}^*_{ij} + \boldsymbol{t}_j\right)\right\|^2. \qquad (38)$$

The update equation (23) for $\boldsymbol{t}_j$ continues to be applicable here.

We derive the following update equation for $\boldsymbol{A}_j$ by setting the partial derivative of J to $\boldsymbol{A}_j$ to zero:

$$\begin{aligned}\frac{\partial J}{\partial \boldsymbol{A}_j} &= \sum_{i=1}^{N} u_{ij}^m \frac{\partial}{\partial \boldsymbol{A}_j}\left[(\boldsymbol{A}_j\boldsymbol{p}^*_{ij} + \boldsymbol{t}_j - \boldsymbol{x}_i)^T(\boldsymbol{A}_j\boldsymbol{p}^*_{ij} + \boldsymbol{t}_j - \boldsymbol{x}_i)\right]\\ &= \sum_{i=1}^{N} u_{ij}^m \frac{\partial}{\partial \boldsymbol{A}_j}\\ &\quad\times\begin{bmatrix}(\boldsymbol{A}_j\boldsymbol{p}^*_{ij})^T(\boldsymbol{A}_j\boldsymbol{p}^*_{ij})\\ +2(\boldsymbol{t}_j - \boldsymbol{x}_i)^T(\boldsymbol{A}_j\boldsymbol{p}^*_{ij}) + (\boldsymbol{t}_j - \boldsymbol{x}_i)^T(\boldsymbol{t}_j - \boldsymbol{x}_i)\end{bmatrix}\\ &= \sum_{i=1}^{N} u_{ij}^m\left[2\boldsymbol{A}_j\boldsymbol{p}^*_{ij}(\boldsymbol{p}^*_{ij})^T + 2(\boldsymbol{t}_j - \boldsymbol{x}_i)(\boldsymbol{p}^*_{ij})^T\right] = 0.\end{aligned} \qquad (39)$$

Rearrangement of the last step in (39) yields the following update equation for $\boldsymbol{A}_j$:

$$\boldsymbol{A}_j = \left[\sum_{i=1}^{N} u_{ij}^m(\boldsymbol{t}_j - \boldsymbol{x}_i)(\boldsymbol{p}^*_{ij})^T\right]\left[\sum_{i=1}^{N} u_{ij}^m\boldsymbol{p}^*_{ij}(\boldsymbol{p}^*_{ij})^T\right]^{-1}. \qquad (40)$$

The resulting clustering algorithm has the same form as the general form in Section II, with the part within the loop replaced by the following statements:

```
Find all $\boldsymbol{p}_{ij}$ and corresponding $\boldsymbol{p}^*_{ij}$
Compute $d_{ij}$ using (38)
Update $u_{ij}$ using (5)
Update $\boldsymbol{t}_j$ using (23)
Update $\boldsymbol{A}_j$ using (40)
```

We use the name PCT-III to collectively represent these statements for later reference.

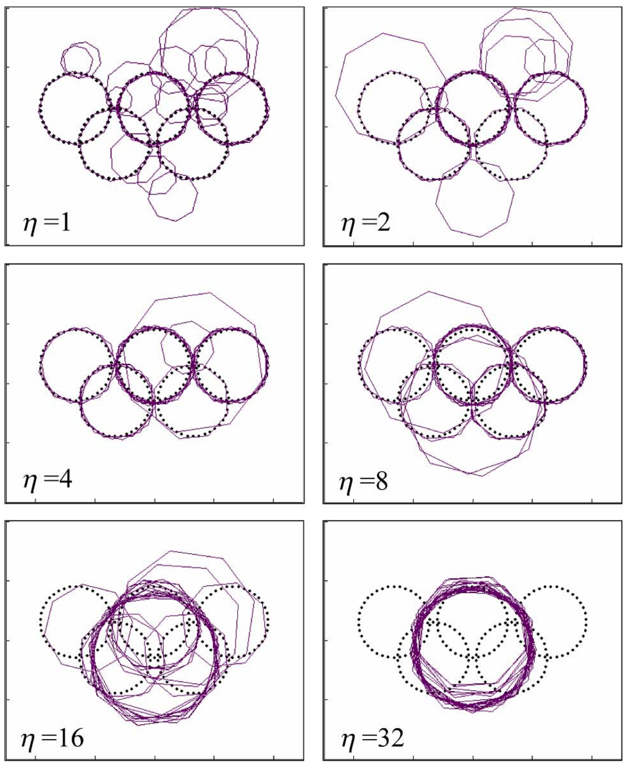

Fig. 2. Demonstration of the convergent prototypes of the same dataset using different η values.

E. Issues of Convergence

One interesting question in possibilistic shell clustering is this: When a prototype converges, how likely and how closely does it converge to an actual cluster? For shell clusters of template-based shapes, the answer can vary widely for different shapes. This is because different shapes have different probabilities that incorrect combinations of data points from one or multiple actual clusters can lead to local minimums of the cost function. While it is impossible to analyze this issue for all possible shapes, there is a common factor, the "zone of influence" η, that significantly affects this effect. This is discussed in more detail here.

One important property of η, as previously pointed out in [32], is that it plays the role of "resolution" of the feature space. Larger η values correspond to lower resolutions and smoother energy surfaces because the contribution from each data point now spans a larger range. This again leads to generally smaller numbers of local optima. However, if η is too large, these local optima may not correspond to actual clusters because the contribution from data points that belong to different actual clusters are blended together. This is illustrated in Fig. 2 where the same dataset (the noiseless five-circle dataset also shown in Fig. 5) is clustered using different fixed values of η and 25 randomly initialized prototypes. We can see that for $\eta = 32$, all the prototypes tend to converge to the same "global optima," which is not our intended result. On the other hand, when $\eta = 1$, many prototypes converge to different local optima, implying a relatively complex energy surface. A good compromise here seems to be $\eta = 4$, where 23 out of 25 prototypes converge to actual clusters. However, this observation still does not tell us how to choose the best value of η in general. Instead, we describe in the next section a coarse-to-fine strategy that continuously adjusts the value of η during the clustering process.

All the clustering algorithms described in this section have Gauss–Seidel type iterations. This means that, while different parameters are updated separately, the computation of the new value of each parameter always utilizes the most recent values of the other parameters. This is the case for the FCM, PCM, and many other clustering algorithms derived from them. As pointed out in [33], because of faster convergence in practice (at least for nonparallel implementations), Gauss–Seidel type iterations are often preferred over Jacobi-type iterations, where the update of all the parameters occurs simultaneously in each iteration. Another reason of this choice is that it ensures that the cost function decreases monotonically after each update.

IV. Progressive Clustering Procedure

The strong dependence of clustering results on initialization—including the number of clusters and initial prototype parameters—is a very challenging issue for any type of shell clustering. To deal with this challenge, we implement a progressive clustering procedure that enables the detection of an uncertain number of clusters and the estimation of their parameters. The basic idea of progressive clustering is to progressively remove prototypes that appear to represent good clusters as well as the data points belonging to these clusters. The partial removal of data reduces the complexity of the remaining problem and increases the likelihood of finding good clusters in the remaining data. The use of progressive clustering for clustering hyperspherical and hyperquadratic shells has been described in [13] and [18]. Another GA-based detection algorithm of hyperquadratic shells [34] employs a similar idea.

Our method incorporates a few ideas from the progressive clustering algorithm for spherical and quadratic shells described in [18], including the detection and deletion of spurious clusters, the merging of similar prototypes, the use of surface density to determine the "goodness" of individual prototypes, and a refinement stage at the end of the main clustering loop. There are also a few elements of our algorithm that are different. The main problem with the procedure in [18] is that it employs the fuzzy version of the clustering algorithm to initialize the possibilistic clustering procedure. While this approach works fine with quadratic shells, the results with more general shapes have been much less satisfactory. We find that such an initialization method often produces prototypes that overlap with parts of one or more actual clusters, such that they contain too many data points to be considered spurious and too few data points to be considered good. Mostly such prototypes remain after the progressive clustering procedure and become part of the incorrect final result.

For the reason stated before, we employ a completely possibilistic approach that allows randomly initialized prototypes to search for their local optima individually. Instead of focusing on determining "the correct number of clusters," our approach is to find "as many good clusters as possible in a reasonable amount of time." The additional steps are the coarse-to-fine searching process through the adjustment of η values and the reinitialization of prototypes. The pseudocode listing next summarizes our progressive clustering procedure, followed by more detailed explanations.

```
Randomly initialize $C_0$ prototypes
/* The main loop */
REPEAT
  FOR each prototype
    Run one iteration of PCT-I, -II, or -III based on the type
        of prototype transformation
END-FOR
Merge similar prototypes
Discard spurious or bad convergent prototypes
Detect good convergent prototypes, extract them to a
      separate list, and remove the data points that are within
      a small distance from these good prototypes
Adjust $\eta$ for each prototype
Replace discarded, merged, and extracted prototypes with
      new randomly initialized prototypes
UNTIL there are few data points left or the maximum
      allowed number of iterations is reached
/* The refinement loop */
Replace the removed data points into the dataset
Add the extracted good prototypes to the remaining
      prototypes at the end of the main loop
FOR a predefined number (typically 5) of iterations
  FOR each prototype
    Run one iteration of PCT-I, -II, or -III based on the type
        of prototype transformation
    Adjust $\eta$ for each prototype
  END-FOR
END-FOR
/* The final selection loop */
WHILE there is at least one prototype with a density of at
      least a given threshold, $\rho_{\text{select}}$
  Extract the prototype with the highest density to a separate
      list, and remove the data points that are within a
      small distance from this prototype
  Recalculate the densities of the remaining prototypes
      based on the remaining data points
END-WHILE
```

There are three separate loops in the aforementioned pseudocode: the main, the refinement loop, and the final selection loop. The prototypes that are deemed likely to represent actual clusters are extracted during the main loop. The purposes of the two subsequent loops are given as follows.

Within the main loop, the update of a prototype is terminated when it is determined to have converged. However, if the convergence is slow, it is possible that the prototype is still slightly off its optimal location. The refinement loop allows the prototype update to proceed for several additional iterations without checking for convergence in an attempt to move the prototypes

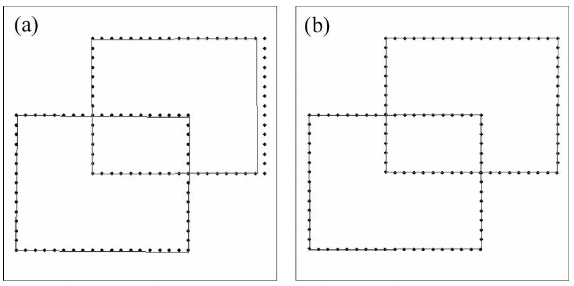

Fig. 3. Example of the effect of the refinement loop. (a) and (b) Prototypes before and after the refinement loop, respectively.

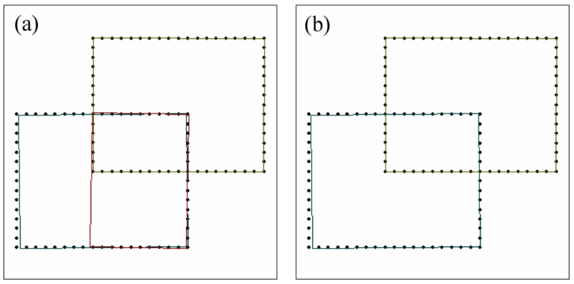

Fig. 4. Example of the effect of the final selection loop. (a) and (b) Prototypes before and after the final selection loop, respectively.

closer to their optimal locations. This is illustrated in Fig. 3, where plots (a) and (b) display two rectangular prototypes before and after the refinement loop, respectively. We are able to see that the prototypes match to the data points better after the refinement loop.

The final selection loop is designed to handle the cases when multiple extracted prototypes partially overlap with on another, meaning that they share some common data points. In this case, we consider the prototype with the highest density as the one most likely to be correct. Removing this prototype and its associated data points lowers the density of the other prototypes. Since we only retain prototypes with densities of at least ρ_{select} (currently 0.6) in this loop, some prototypes that are not correct clusters can be discarded here. Fig. 4 displays an example of this effect. Here, one incorrect prototype (dark red) in Fig. 4(a) that is due to a local optimum is discarded through this selection process, and the final selected prototypes are shown in Fig. 4(b).

More details of the various components in the pseudocode are explained next.

Prototype density: This is similar to the surface density used in [18] as a single-cluster validity measure in shell clustering. This density is defined as

$$\rho_j = \frac{1}{L_j} \sum_{d_{ij} \leq \delta_w} u_{ij} \tag{41}$$

where L_j is the total edge length of the *j*th prototype. L_j replaces the effective arc length used in [18], where its purpose is to allow the density calculation and extraction of, say, a partial circle. This is because we are not interested in detecting objects that do not contain at least most of the template. The value used for the distance threshold δ_w is 1.0 and 2.5 for synthetic and image datasets, respectively.

Prototype initialization: Each prototype is initialized with its parameters randomly selected from within a predefined allowed parameter range (which will be explained later).

Prototype convergence test: This is done separately for each prototype. A sequence of prototypes is considered to have converged if the difference between the same prototype at consecutive iterations is within a predefined threshold δ_c (which is currently set to 0.25).

Prototype merging: When two prototypes whose difference according to (14) is within a predefined threshold δ_m (which is currently set to 1.0), the prototype with lower density is considered merged into the other prototype and is therefore marked for deletion.

Good prototype extraction: Prototypes with density above a threshold ρ_{good} is considered a "good" one and is moved to a separate list. Data points within δ_w of this prototype are also removed. The value of ρ_{good} is set to a high value (currently 0.8) in the beginning and reduced by a small amount (currently 0.1%) after each iteration. This gradually relaxes the requirement of good prototypes and allows less-than-perfect clusters to be detected at later stages of the process. An example of such clusters in computer vision applications is objects that are somewhat obscured.

Discarding Spurious and bad prototypes: Prototypes with densities below a given threshold $\rho_{\min}$ (currently 0.3) at convergence are marked for deletion. Also marked for deletion are prototypes whose parameters fall outside of some predefined bounds, which are given based on prior knowledge of the data. In addition, during the main loop, "stuck" prototypes (defined as convergent prototypes whose densities are between $\rho_{\min}$ and ρ_{good}) are deleted with a probability of r_{stuck} (which is currently 0.2).

Adjustment of η: The parameter η_j is updated in a coarse-to-fine manner. In the main loop, its value is set to a relatively large value $\eta_{(0)}$ [$\eta_{(0)}^{1/2} \approx$ (total range of data)/10 has worked well for our datasets] for a newly initialized prototype to make sure that it is able to move toward nearby data efficiently. It is reduced by a multiplicative factor r_η [$0 < r_\eta < 1$, currently 0.7] according to

$$\eta_{j(t+1)} = \max[\eta_{\min}, r_\eta \eta_{j(t)}], \tag{42}$$

so that the prototype can gradually converge. Here *t* is the iteration counter, and $\eta_{\min}$, currently set to be equal to δ_w^2, is a lower bound of η. The values of η are all reset to $4\eta_{\min}$ at the beginning of the refinement loop because we do not want η values to be so large that they cause the prototypes to move too much here. In addition, during the main loop, the η values for "stuck" prototypes that are not selected for deletion are increased by a factor of 4 to increase the likelihood of them moving out of the local minimums of the cost function.

Prototype replacement: At the end of each iteration of the main loop, prototypes marked for deletion are replaced with new randomly initialized prototypes. This keeps the total number of prototypes constant. One advantage of this step is that we do not need to over-specify the number of initial prototypes as in [13] and [18], which requires at least an upper bound on the total number of clusters present in the data.

Termination criterion: The main loop terminates after the ratio between the amounts of remaining and total data is less than a threshold r_T. This threshold is set to a prespecified value $r_{T(0)}$ (which is currently 5%) in the beginning and after each good prototype extraction, and is multiplied by a factor slightly larger than 1 (which is currently 1.02) after each iteration. This effectively ensures that the main loop will eventually terminate even when no prototype is extracted at all.

A number of thresholds and other parameters are used in this progressive clustering procedure. To estimate the sensitivity of the clustering results on the parameter values, we include in Section V a series of experimental results for this purpose. We find that the clustering results do not exhibit strong sensitivity to most of the parameters. Among the parameters, choices related to η (r_η and $\eta_{(0)}$) are found to have stronger influence on the clustering performance, even though our method of selecting their values work in most of our experiments. This is discussed in more detail in Section V.

Currently, the main purpose of prototype parameter range constraints (as for the removal of "bad" prototypes) is only to remove prototypes that have become too large or too elongated because the procedure is trapped in local optima that include data points from multiple actual clusters or too small because it is trapped by a few individual data points. As a result, we actually use only constraints that correspond to scaling. For an affine transformation, we obtain its two corresponding scaling factors according to its effect on a unit circle. We employ constraints on 1) the mean scaling factor (geometric mean of the scaling factors in two directions for types II and III transformations) and 2) the aspect ratio (for types II and III transformations only). In order to make sure that the constraints are not overly limiting, we keep the ratio between the upper and lower bounds of the mean scaling factor to at least 3. The typical allowed range of the aspect ratio is between 1 and 3 or 4. Our experiences have indicated that prototype parameter range constraints that are too tight can actually worsen the efficiency of the progressive clustering procedure by causing too many reinitializations of bad clusters.

V. Experimental Results

In this section, we present our experimental results using various datasets. Section V-A includes synthetic datasets consisting of data points that form various shapes. We include synthetic datasets with and without noise points and minor scatters. Section V-B includes results using edge pixels that are extracted from real images. Section V-C describes how we measure the efficiency of our overall algorithm. Section V-D describes experiments designed to measure the sensitivity of the clustering performance on parameter choices. For all the results presented, unless noted otherwise, the progressive clustering procedure described in Section IV is employed. For each dataset, we only look for clusters of one particular shape, i.e., with all the prototypes derived from the same template. Edge-based templates are used throughout the experiments.

A. *Results for Synthetic Data*

Figs. 5–7 display the clustering results of synthetic datasets obtained by using algorithms based on type I, type II, and type III transformations, respectively. Each of these figures demonstrates the detection of several different shapes. For each shape, we show clustering results for both noiseless data with no scatter, and for data that contain noise and minor scatter. Detected prototypes are plotted with the data as solid lines. The datasets shown in Figs. 5–7 are not identical because we have selected in each case only the datasets with shapes that are within the capability of the prototype transformation being used. For example, we do not use data that contain elliptic shapes for type I transformation when the prototypes are derived from a circular template, which yields only circular prototypes.

For the synthetic datasets in Figs. 5–7, each square corresponds to a size of 50 $\times$ 50, while the distances between adjacent data points in noiseless datasets are approximately equal to 1. The numbers of nonnoise data points in these datasets range approximately from 140 to 300. For the datasets with noise, we always place 50 noise points randomly within the 50 $\times$ 50 area, and each data point is displaced by an amount around ± 0.5 from its location in the corresponding noiseless dataset. The results indicate that the algorithm is capable of detecting the desired shapes in the presence of minor scatter and noise and of detecting the correct number of clusters. We also demonstrate that the same template (such as a square) can be used for the detection of several different shapes through different types of prototype transformations.

Figs. 5 and 6 include examples that involve open shapes (the "U" and "S" shapes, respectively). This demonstrates that our algorithm does not require the templates to have closed shapes. However, we want to note that there are some open shapes that are not suitable for our algorithm. An example of such a shape is the cross "+." The problem here results from the difference between the cost function used to derive the prototype parameters and the validity measure used to determine the goodness of prototypes. This is illustrated with the examples in Fig. 8, where we have two "cross" prototypes with different sizes. Both are matched perfectly with the same set of data points, resulting in the same values of the cost function *J*. However, the larger prototype has lower validity [the surface density in (41)] due to the factor L_j, and may be rejected based on the low density. During the progressive clustering procedure, it is quite possible that prototypes that do converge and match well to the data points are rejected due to low density. Our current method of updating prototype parameters will not reduce the scaling factor for such cases (to increase validity) because this does not further lower *J*. The overall consequence is that our clustering procedure will have difficulty and much worse efficiency in identifying such clusters. Other than the cross shape, shapes that have this problem include line segments, "X," "V," "T," and any shape that can "contain a smaller version of itself" after some translation and rotation. In addition to the "U" and "S" shapes in Figs. 5 and 6, other open shapes that can be clustered with our current algorithms include "C," "W," "Z," etc.

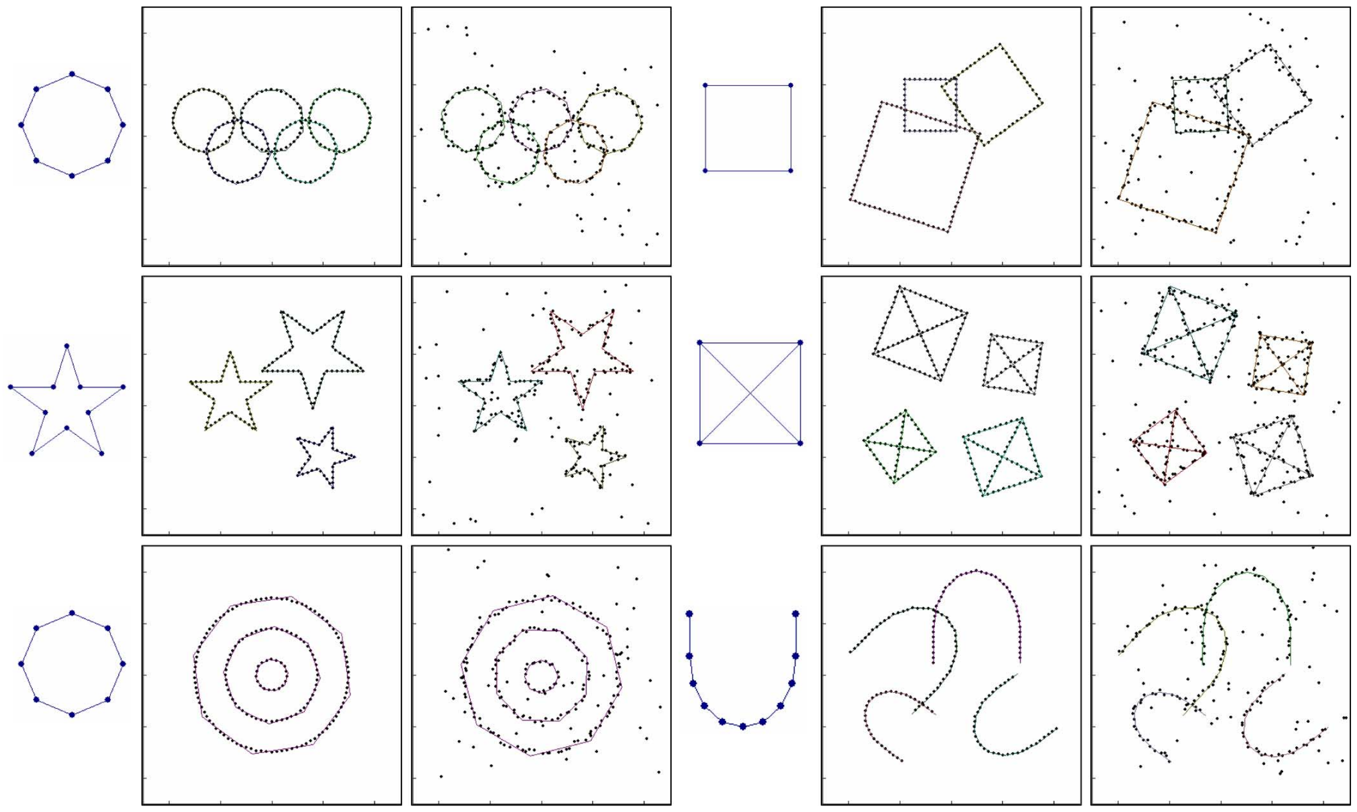
Fig. 5. Clustering results of synthetic datasets using type I prototype transformations. For each shape, there are three plots. From left to right: the template, the clustering result with a noiseless and scatterless dataset, and the clustering result with a similar dataset with noise and minor scatter.

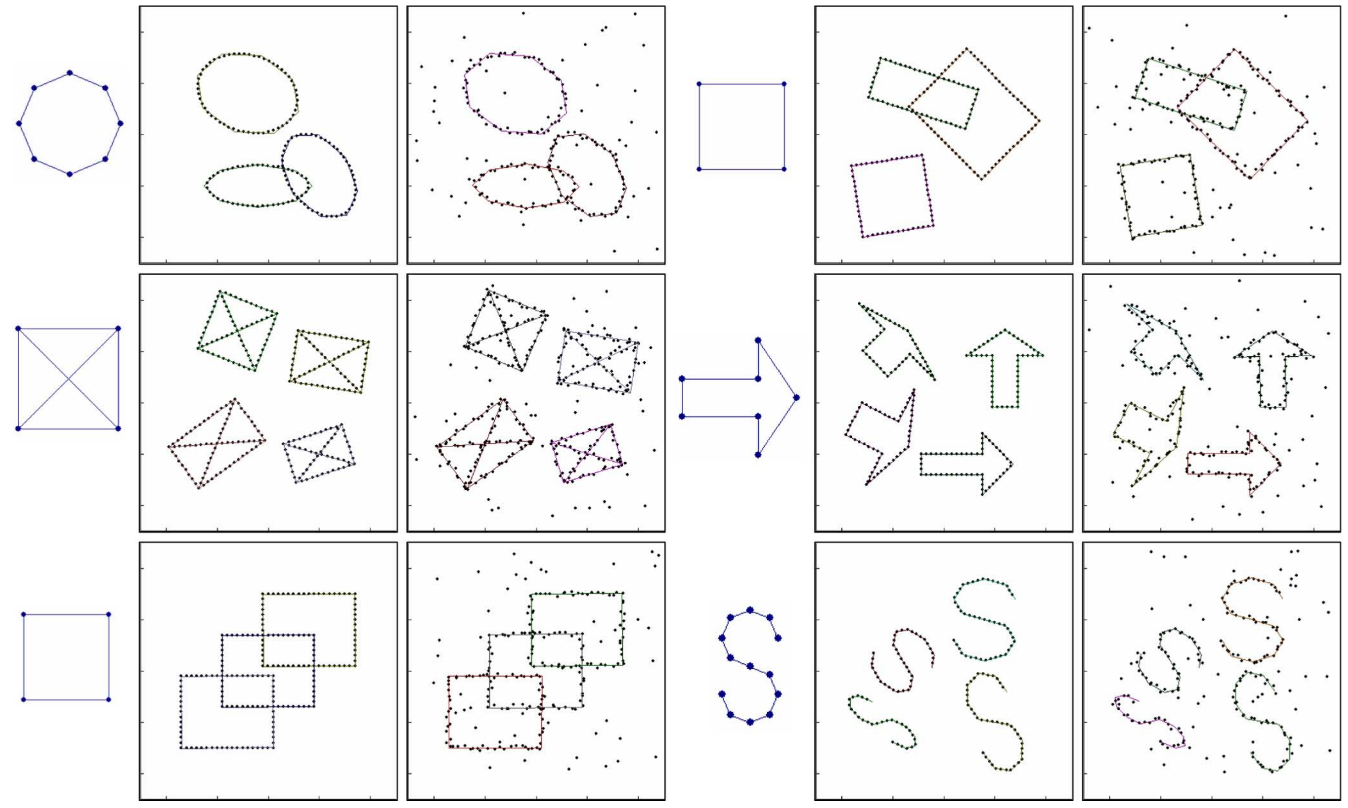
Fig. 6. Clustering results of synthetic datasets using type II prototype transformations. For each shape, there are three plots. From left to right: the template, the clustering result with a noiseless and scatterless dataset, and the clustering result with a similar dataset with noise and minor scatter.

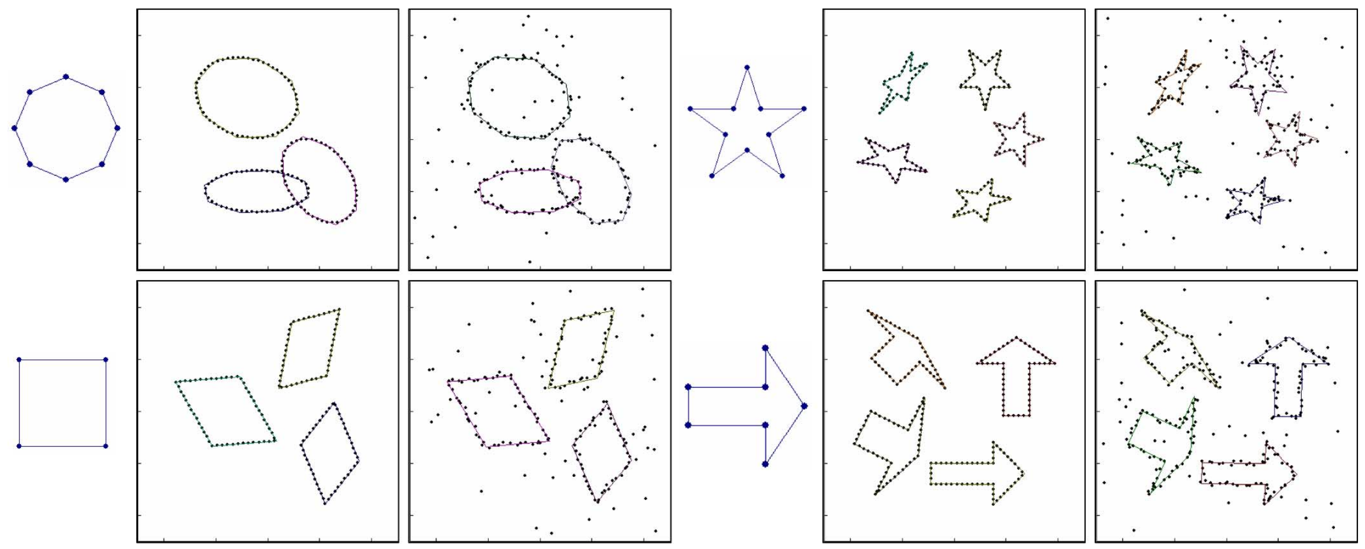
Fig. 7. Clustering results of synthetic datasets using type III prototype transformations. For each shape there are three plots. From left to right: the template, the clustering result with a noiseless and scatterless dataset, and the clustering result with a similar dataset with noise and minor scatter.

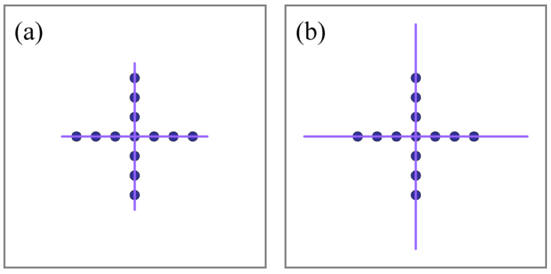

Fig. 8. Example of the same dataset matched perfectly to two prototypes with different scaling.

B. Results for Image Data

This section contains examples that demonstrate the detection of generic shapes in real images using our algorithms. All these datasets contain some scatter and noise/clutter, as would be expected from real-world images. Figs. 9–11 display examples of clustering with the algorithm based on type I, type II, and type III transformations, respectively. Again, in each figure, we include only datasets with shapes that are within the capability of the particular type of transformations. Each row of these figures, from left to right, shows the template, the original image, the extracted edge points used as the data points for clustering, and the original image overlaid with the final extracted prototypes, respectively. The images are all 160 × 120 in size. The steps of edge point extraction include smoothing, gradient computation (on either the intensity, saturation, or a single RGB component), thresholding, and thinning. We do not explain the detail of edge point extraction for each image here as this is intrinsically application dependent and is not the focus of this paper.

C. Correctness and Time Complexity

In this section, we focus on an important issue in a lot of practical problems: the tradeoff between correctness and time complexity. Shell clustering has been known for its tendency to be trapped in suboptimal solutions, and this is exactly the reason for the various robust shell clustering procedures described in Section IV and in the literature [13], [18], [34], and [35]. Even with progressive clustering or the use of other near global optimization techniques such as GA, the final result is not guaranteed to be globally optimal. However, there has been little analysis in the literature regarding this aspect of shell clustering. We provide a framework that should be useful for analyzing and comparing the performances of various shell-clustering algorithms.

Let us consider the synthetic data first. Since we know the actual locations of all the shell clusters (called the "target clusters" later) used for generating the data, we just compute their distances to the detected cluster prototypes using (13) and (14). We use the following formula to calculate a "grade of detection" g_d for each target cluster:

$$g_d(P_T) = \begin{cases} 1, & d \le \delta_w \\ \dfrac{(3 - d/\delta_w)}{2}, & \delta_w < d \le 3\delta_w \\ 0, & \text{otherwise.} \end{cases} \tag{43}$$

Here, P_T represents the target cluster prototype, and d is its distance to the closest prototype generated by the clustering algorithm:

$$d = \min_j \operatorname{diff}(P_T, P_j). \tag{44}$$

Let C^* be the number of target clusters in a given dataset. When $C^* > 1$, the overall g_d is just the average g_d of all the target clusters. This measure for evaluating clustering results is more intuitive and meaningful for shell clustering than the objective function, which is the measure used in [30] for point-prototype clusters.

One performance measure we propose here is the amount of computation (which is denoted as k_c below) required to reach a given g_d. We define k_c as the number of d_{ij} computations

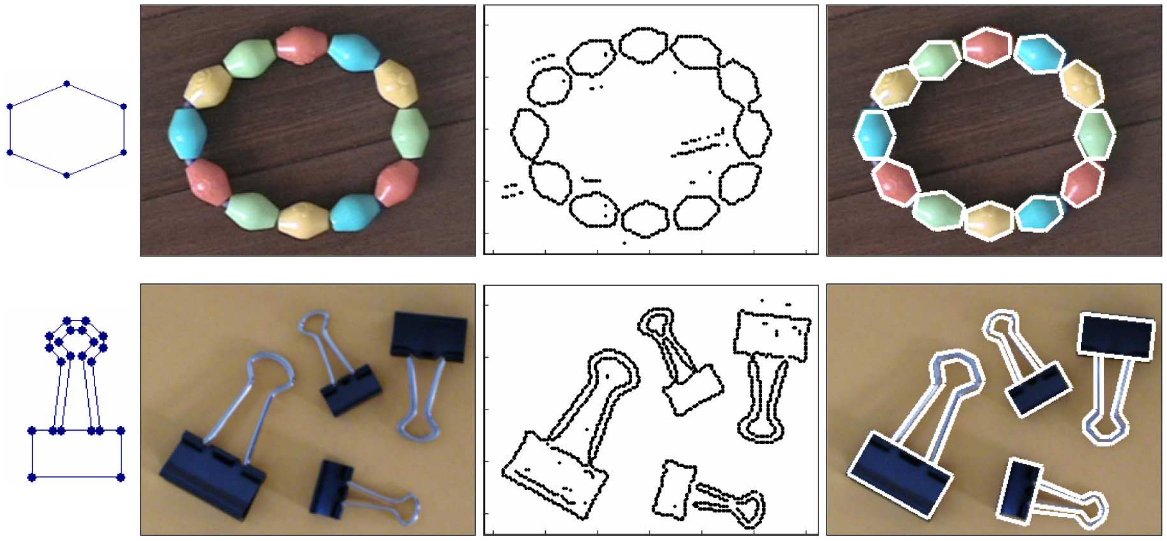
Fig. 9. Clustering results of image datasets using type I prototype transformations. From left to right, each row contains the template, the original image, the edge points to be clustered, and the original image overlaid with the extracted prototypes.

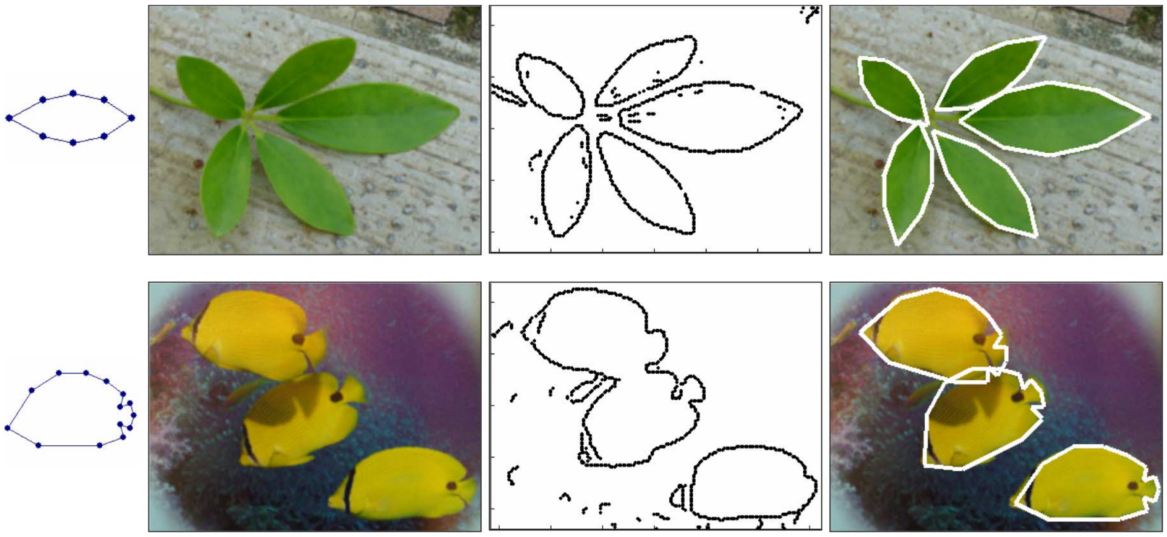
Fig. 10. Clustering results of image datasets using type II prototype transformations. From left to right, each row contains the template, the original image, the edge points to be clustered, and the original image overlaid with the extracted prototypes.

(including matching point determination) per data point. This is because this step is the most computationally expensive of the whole algorithm and consumes the majority (above 80%) of the total execution time. This unit of computation provides a mechanism for comparing and optimizing algorithms (except for the part on d_{ij} computation) that are independent of the computer system used.

For each given set of data and parameters, we always do 20 runs with the only difference being the seed of the random number generator. For each run, we can obtain a curve by using the cumulative amount of computation after each main-loop iteration as the horizontal axis and the g_d value based on the extracted good clusters up to that iteration as the vertical axis. The resulting curves of the 20 runs are then averaged. Where a horizontal line (corresponding to a reference g_d) first intersects this average curve is used as the expected amount of computation required to reach that grade of detection [which is denoted as $k_c(g_d)$]. This process is illustrated in Fig. 12.

Analyzing g_d as a function of k_c allows us to observe the dynamic evolution of g_d as the algorithm proceeds. However, since it is possible that in some of the runs, $k_c(g_d^*)$ may not exist because the final g_d is less than g_d^*, we cannot reliably compute the standard deviation of $k_c(g_d^*)$ over multiple runs for stability analysis. To facilitate the analysis of performance stability over multiple runs, we also use a second performance measure Δg_d, defined as the ratio $g_{d(\text{end})}/k_{c(\text{end})}$. Here, the subscript "(end)" indicates that we are using the values of k_c and g_d when the progressive clustering procedure terminates. This ratio represents the mean grade of detection achieved per unit of computation. We want to note that these performance measures are not tied to the particular algorithms in Sections III and IV. Comparisons of the performance measures across datasets may indicate their

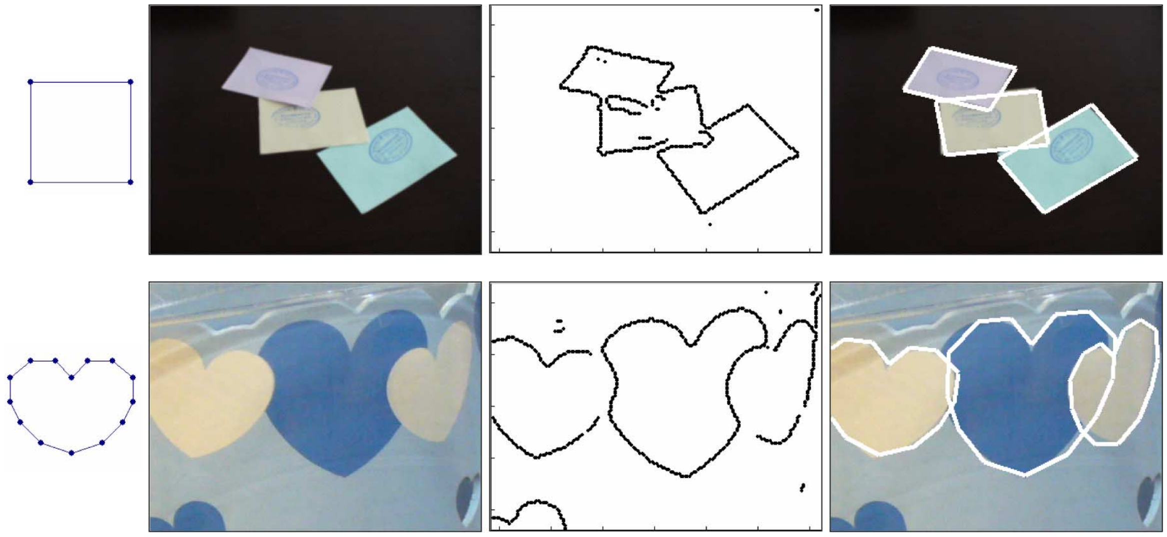

Fig. 11. Clustering results of image datasets using type III prototype transformations. From left to right, each row contains the template, the original image, the edge points to be clustered, and the original image overlaid with the extracted prototypes.

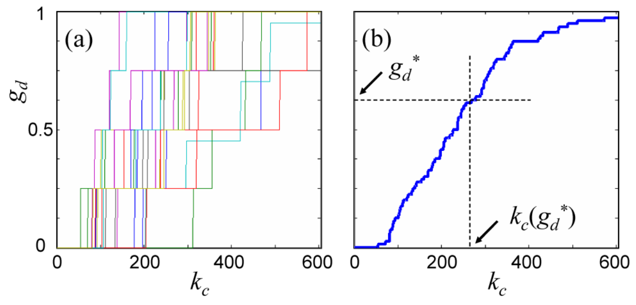

Fig. 12. Computation of $k_c(g_d)$. (a) Individual g_d versus k_c plots of the 20 runs. (b) Average g_d versus k_c plot. The dashed lines illustrate how $k_c(g_d^*)$, g_d^* being a reference g_d, is determined.

relative levels of difficulty for the same algorithm. For the same dataset, these performance measures are useful for parameter optimization or even the choice among different algorithms for the same purpose.

Table I lists the performance for all the synthetic data experiments shown in Figs. 5–7. The results for $g_{d(\mathrm{end})}$, time, and Δg_d are their mean values and standard deviations over 20 runs. The program is implemented in Matlab 7.0 and run on a computer with a 3.0-GHz Pentium IV processor and 1 GB of memory.

For the noiseless datasets, we can see that we can achieve over 80% of correct cluster detection (i.e., $g_{d(\mathrm{end})} \geq 0.8$) for most of the datasets except for ellipses and rectangles. This is partly due to the fact that these datasets involve significant overlap between target clusters. Closer examination also reveals some shape-dependent difficulties. For example, a rectangular prototype that has only three sides correctly matched to the data may have a high enough density to be extracted as a good cluster, especially at later stages of the main loop when the density threshold for good clusters is lower. Another observation is that when the same shape can be modeled by two transformations (for example, ellipses can be derived from a circular template through either type II or type III transformations), the one with more degrees of freedom (type III here) usually performs worse. This can be explained by the fact that more degrees of freedom usually lead to more complicated energy surfaces and, therefore, a higher probability for a prototype to converge to local optima, which is an effect that is directly related to the cost function and does not result from the progressive clustering procedure. For the noisy datasets, it always takes longer time than the corresponding noiseless datasets. The degradation of $g_{d(\mathrm{end})}$ due to noise is quite dataset-dependent.

Table II lists the performance measures using the image datasets in Figs. 9–11. We can see that good clustering results can be achieved in the order of 10 s for a variety of shapes.

Next, we present an example of the ability of our algorithm in handling cluttered data, i.e., data that also contain shapes different from the template. In Fig. 13(a) is a synthetic dataset that contains three different shapes: two each of squares, hearts, and stars, as well as some noise. We run our clustering algorithm on this dataset three times, each time using just the template of one shape. We use only type I transformations in this experiment. The detected clusters are shown in Fig. 13(b)–(d). It is clear that we are able to extract only the clusters of the correct shape each time. For further analysis, we also test on three additional (uncluttered) datasets, each consisting of only the points in Fig. 13(a) that correspond to one of the three shapes. The performance comparison (averaged over 20 runs in each case) with identical parameter settings is given in Table III. We can see that the clutter increases the execution time by approximately a factor of 2 (approximately proportional to N) and somewhat decreases the probability of detection with the $g_{d(\mathrm{end})}$ value dropping from 0.9 to 0.63 for "hearts," which is the most difficult of the three shapes. On the other hand, we have no false detection (identification of shapes where they do not exist) throughout this experiment. This is a clear evidence of the ability of our algorithm to distinguish among different shapes.

TABLE I
CLUSTERING PERFORMANCE MEASURE FOR SYNTHETIC DATASETS

Type	Shape	C^*	C_0	N	Noiseless data				Noisy data with scatter			
					$g_{d(end)}$	time (s)	$k_c(g_d)$ g_d=0.5	Δg_d (x10000)	$g_{d(end)}$	time (s)	$k_c(g_d)$ g_d=0.5	Δg_d (x10000)
I	Circle	5	2	244	1.00±0.00	0.6±0.1	56	86±14	1.00±0.00	0.9±0.2	65	72±18
	Square	3	3	176	0.95±0.12	0.6±0.2	70	82±22	0.94±0.16	1.2±0.2	96	54±18
	Star	3	2	183	1.00±0.00	0.6±0.2	59	94±25	0.99±0.03	1.1±0.2	84	63±19
	Square+X	4	4	306	1.00±0.00	1.1±0.3	149	48±17	0.95±0.22	1.8±0.5	208	32±13
	Concentric circles	3	3	169	0.80±0.41	1.2±0.7	294	45±43	0.83±0.35	1.7±0.5	223	35±24
	U-shape	4	6	153	0.85±0.15	2.0±0.7	250	27±13	0.68±0.18	3.4±0.6	398	12± 5
II	Ellipse	3	6	147	0.88±0.22	1.3±0.7	146	40±19	0.82±0.28	2.5±0.5	177	23±11
	Rectangle	3	10	184	0.73±0.21	1.3±0.5	186	29±10	0.71±0.25	3.3±0.7	204	17± 9
	Rectangle+X	4	8	314	1.00±0.00	1.5±0.4	215	33±10	0.94±0.14	2.8±0.5	268	24± 7
	Arrow	4	6	220	0.97±0.08	1.9±0.6	227	27± 9	0.93±0.14	3.3±0.7	312	18± 6
	Aligned rectangles	3	8	192	0.79±0.21	1.4±0.6	172	34±13	0.66±0.30	2.9±0.5	222	19±10
	S-shape	4	6	152	1.00±0.00	1.6±0.5	197	33±12	0.83±0.17	3.6±0.7	348	16± 6
III	Ellipse	3	6	147	0.63±0.31	1.5±0.6	212	27±16	0.53±0.32	2.4±0.4	340	15±11
	Star	5	10	202	0.94±0.08	2.7±0.3	368	16± 2	0.58±0.26	5.9±1.1	870	6± 3
	Parallelogram	3	8	143	0.98±0.07	1.2±0.6	189	38±16	0.86±0.19	2.7±0.6	352	18± 9
	Arrow	4	6	220	0.90±0.21	2.5±1.0	367	19±10	0.77±0.22	4.1±0.8	562	10± 6

Time and performance measures are the average results of 20 runs.
The values of N are for noiseless data sets. The corresponding noisy data sets always have 50 more data points.

TABLE II
CLUSTERING PERFORMANCE MEASURE FOR IMAGE DATASETS

Type	Shape	C^*	C_0	N	$g_{d(end)}$	time (s)	$k_c(g_d)$ g_d=0.5	Δg_d (x10000)
I	ring of beads	12	12	748	0.94±0.05	10.0±1.6	711	8±2
	paper clips	4	8	1093	0.85±0.19	13.8±4.2	670	11±6
II	leaves	5	8	740	0.70±0.28	7.7±0.9	919	7±3
	fish	3	6	644	0.67±0.16	5.5±1.0	495	12±5
III	paper card	3	4	403	0.88±0.17	1.9±0.5	249	27±11
	heart	3	6	520	0.57±0.26	5.9±1.4	704	9±5

Time and performance measures are the average results of 20 runs.

TABLE III
CLUSTERING PERFORMANCE FOR MULTISHAPE DATA

Shape (data)	Shape (template)	C^*	C_0	N	$g_{d(end)}$	time (s)	Δg_d (x10000)
All	Square	2	6	641	0.90±0.21	4.7±0.5	10± 3
All	Heart	2	6	641	0.63±0.32	9.7±1.0	6± 3
All	Star	2	6	641	0.85±0.24	6.5±1.0	9± 4
Square	Square	2	6	266	1.00±0.00	2.5±0.6	27±10
Heart	Heart	2	6	218	0.90±0.26	4.3±1.0	15± 8
Star	Star	2	6	257	1.00±0.00	2.0±0.2	38± 7

Time and performance measures are the average results of 20 runs.

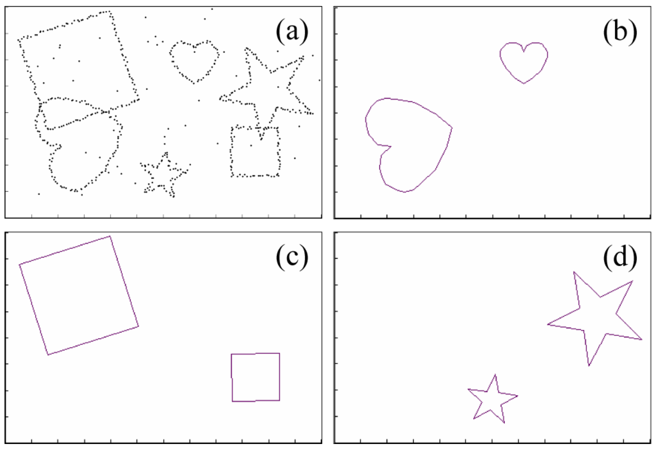

Fig. 13. Clustering results for datasets containing multiple shapes. (a) Data points. (b)–(d) Extracted clusters using heart, square, and star templates, respectively.

We also repeat the experiments using the same datasets and type II transformations, and the results are not as robust as when using type I transformations. With the multishape dataset, the $g_{d(\mathrm{end})}$ values drop to 0.32, 0.28, and 0.84 for squares, hearts, and stars, respectively. In addition, there is an average of 1.0 false detection per run (20 total for the 20 runs) when clustering squares but zero false detection when clustering hearts and stars. While the worse performance with type II transformations is expected, as more degrees of freedom lead to more possibilities of wrong matches to the data, the significance of this effect varies a lot for different shapes: reduced true detection and increased false detection for squares/rectangles, reduced true detection but no effect on false detection for hearts, and virtually no effect for stars. We believe that this can also lead to an interesting research question regarding what characteristics of the target shape can affect the robustness of clustering results.

It may appear from the earlier observation that the type II version of our algorithm generally performs worse compared with the type I version. We want to note that this phenomenon only occurs to cases when the shapes to be detected can be derived from the template through type I transformations, which is the case here. For example, with a square template, the type I version can only be used for detecting squares, while the type II version is required for detecting rectangles of various aspect ratios. Similar arguments apply to the observation from Table I where the type III version performs worse than the type II version when detecting ellipses. Overall, for best performance, we should always select the simplest (i.e., with the least

TABLE IV
DEPENDENCE OF CLUSTERING PERFORMANCE ON PARAMETER CHOICES

Parameter	Value	$g_{d(end)}$			Δg_d (x10000)			Parameter	Value	$g_{d(end)}$			Δg_d (x10000)		
		Set A	Set B	Set C	Set A	Set B	Set C			Set A	Set B	Set C	Set A	Set B	Set C
ρ_{select}	0.5	1.00	0.97	0.94	46	24	15	δ_c	0.15	0.98	0.97	0.94	42	25	13
	0.6	1.00	0.97	0.94	48	27	16		0.25	1.00	0.97	0.94	48	27	16
	0.7	1.00	0.96	0.94	49	28	17		0.5	1.00	0.98	0.88	50	34	18
	0.8	1.00	0.96	0.95	49	28	18	C_0	2	0.90	0.86	0.63	57	42	25
ρ_{min}	0.2	1.00	0.98	0.94	46	27	16		4	1.00	0.97	0.90	48	28	24
	0.3	1.00	0.97	0.94	48	27	16		6	1.00	0.97	0.98	39	27	23
	0.4	1.00	0.97	0.95	48	28	16		8	1.00	0.99	0.93	32	25	17
	0.5	1.00	0.99	0.91	46	30	16		10	1.00	0.99	0.94	28	22	16
ρ_{good}	0.7	1.00	0.78	0.83	48	24	15		12	1.00	1.00	0.96	26	22	14
	0.8	1.00	0.97	0.94	48	27	16	$\eta_{(0)}$	8	1.00	0.81	1.00	49	14	29
	0.9	1.00	1.00	1.00	48	27	16		16	1.00	0.95	0.98	51	23	24
r_{stuck}	0	0.65	0.80	0.93	33	20	15		25	1.00	0.97	0.94	48	27	16
	0.1	1.00	0.96	0.94	47	25	16		36	0.79	0.94	0.78	25	19	8
	0.2	1.00	0.97	0.94	48	27	16		50	0.05	0.33	0.27	2	3	1
	0.4	1.00	0.98	0.95	43	28	16	r_η	0.6	1.00	0.99	1.00	49	27	20
$r_{T(0)}$	0.025	1.00	0.97	0.94	48	26	16		0.7	1.00	0.97	0.94	48	27	16
	0.05	1.00	0.97	0.94	48	27	16		0.8	1.00	0.92	0.76	36	22	7
	0.1	1.00	0.95	0.94	48	26	16		0.9	0.70	0.81	0.06	22	11	0

Parameters and performance values in shaded cells indicate the baselines.

degrees of freedom) of the three versions that is applicable to the shape being detected. However, the versions with higher degrees of freedom are applicable to a much broader variety of shapes.

D. Effects of Parameter Choices

Our clustering procedure involves a number of parameter choices. It is always desirable that the clustering results are not highly sensitive to these parameter choices. This section describes our experiments aimed at investigating how the clustering results depend on the various parameters. These experiments are preformed on three synthetic datasets in Table I and Figs. 5–7: set A, the "squares+X" for type I; set B, the "arrows" for type II; set C, the "stars" for type III.

We start with the parameter values given in Section IV and used for the experiments in Section V-A as our baseline. For simplicity, in each set of experiments here, we vary only one parameter value from the baseline at a time. Table IV summarizes the experiments and resulting clustering performances. We do not include δ_w and δ_m (they are always the same in our experiments) here because their values should be set to the expected amount of scatter in the data and, therefore, are application-dependent.

Other than the last three parameters in Table IV [C_0, $\eta_{(0)}$, and r_η], we can see that the clustering performance is very stable over the tested parameter ranges. In most cases, the variations are less than 10%. For C_0, there appears to be a trend of generally higher $g_{d(\text{end})}$ but lower Δg_d when C_0 is increased, although the difference is most significant between $C_0 = 2$ and $C_0 = 4$. This observation can be explained with the following arguments: By keeping more prototypes at the same time (larger C_0), the probability of having some of them converging to the target clusters is higher, resulting in higher $g_{d(\text{end})}$. On the other hand, it is also more likely that multiple prototypes can converge to the same target cluster. This translates into a "waste" of computation and, therefore, lower Δg_d because the program is detecting the same thing several times.

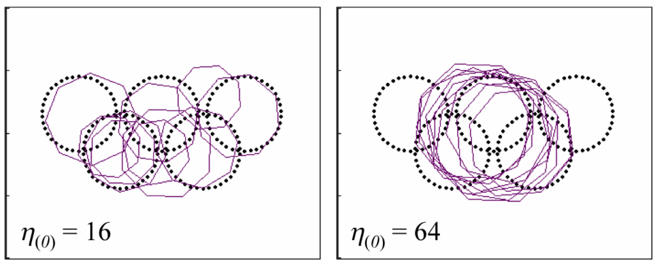

Fig. 14. Demonstration of the convergent prototypes of the same dataset using different η values.

One other interesting observation from Table IV is that the performance is lower when r_{stuck} is zero. This observation supports the benefit of reinitializing prototypes that appear to be trapped at local optima in our progressive clustering procedure.

As mentioned in Section IV and also evident in Table IV, the two parameters [$\eta_{(0)}$ and r_η] related to the "zone of influence" (η) of possibilistic clustering have the most influence to the clustering performance. The most serious degradation appears to occur when $\eta_{(0)}$ is too large and the drop-off is actually quite steep. Additional examination of the intermediate clustering results indicates that the cause is similar to the effect of large η, illustrated in Fig. 2. In Fig. 14, we plot the simulation results that are similar to those in Fig. 2 with ten randomly initialized prototypes. Here, we use no progressive clustering, except for the baseline $r_\eta = 0.7$ for adjusting η values, and only five iterations of the main loop. For the plot with $\eta_{(0)} = 64$, we can see that the prototypes are converging to the "global optimum" even though η is already down to about 16 at the fifth iteration. On the other hand, the plot with $\eta_{(0)} = 16$ seems to show good results with most of the prototypes already quite close to the actual clusters. The performance degradation due to larger r_η can be explained with a similar argument that the prototypes are allowed to look at most of the data points for too long (due to the slower decrement of η) and, therefore, are more likely to move toward the global optimum or a local optimum that spans several actual clusters.

The degradation of clustering performance for smaller $\eta_{(0)}$ and r_η is not nearly as serious and is actually evident only for one (set B) of the three datasets tested here. This observation seems to indicate that it is safer to select smaller $\eta_{(0)}$ and r_η when we are not sure of the optimal values. The only drawback, which is somewhat lower performance and slower convergence for some datasets, is preferred over the performance breakdown by using $\eta_{(0)}$ and r_η values that are too large.

VI. Conclusion

This paper describes the algorithms that facilitate possibilistic c-shell clustering with generic-shape template-based prototypes. We present three different types of transformations for obtaining cluster prototypes from a given template with different degrees of complexity and flexibility. All the algorithms employ the efficient AO scheme that has been common in many existing c-means-based clustering algorithms. The separation of prototype update equations provides two advantages: First, this makes possible the derivation of closed-form update equations for various prototype parameters, hence saving us from the complexity and uncertainty in computational time associated with iterative numerical methods. Second, we have more flexibility in taking advantage of known properties of the templates. We also describe an entirely possibilistic progressive clustering procedure that allows the detection of clusters without prior knowledge of the number of clusters or good initialization of prototypes.

We have presented clustering results of a large variety of shapes using both synthetic and real-world image datasets. Our algorithm has proved to have high probabilities of correctly detecting the desired shapes/objects of various degrees of complexity within seconds. As a result, we believe that this approach has great potential for use in this area. Overall, we believe that further study and understanding of this technique will help expand the application of shell clustering to more image analysis problems.

We have also identified a number of research questions and topics that we will continue to pursue. First, for practical applications, it is important to devise more efficient methods for the task of locating the matching points to improve the overall speed. We are also interested in investigating possible modifications to the algorithms to enable the clustering of shapes such as "X," "V," and "T," as discussed in Section V-A. Another possible future topic is to investigate shell clustering approaches that first group the data points into short line segments, which are then used as the unit of data in the clustering algorithm. This allows the use of not only the location but the direction information of the data points as well, and we believe it is interesting to see whether this can lead to better performance in high-noise scenarios. In addition, we would also like to try using our algorithm for initializing more flexible shape detection techniques, such as deformable-template-based algorithms [36]. Another useful enhancement will be to allow the simultaneous clustering of several different shapes.

Acknowledgment

The author thanks Prof. J. Keller for helpful discussion.

References

[1] J. C. Bezdek, *Pattern Recognition With Fuzzy Objective Function Algorithms*. New York: Plenum, 1981.

[2] I. Gath and A. B. Geva, "Unsupervised optimal fuzzy clustering," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 11, no. 7, pp. 773–781, Jul. 1989.

[3] R. Krishnapurum and J. M. Keller, "A possibilistic approach to clustering," *IEEE Trans. Fuzzy Syst.*, vol. 1, no. 2, pp. 98–110, May 1993.

[4] R. Krishnapurum and J. M. Keller, "The possibilistic c-means algorithm: Insights and recommendations," *IEEE Trans. Fuzzy Syst.*, vol. 4, no. 3, pp. 385–393, Aug. 1996.

[5] M. S. Yang and K. L. Wu, "Unsupervised possibilistic clustering," *Pattern Recog.*, vol. 39, pp. 5–21, 2006.

[6] N. R. Pal, K. Pal, J. M. Keller, and J. C. Bezdek, "A possibilistic fuzzy c-means," *IEEE Trans. Fuzzy Syst.*, vol. 13, no. 4, pp. 517–530, Aug. 2005.

[7] E. E. Gustafson and W. C. Kessel, "Fuzzy clustering with a fuzzy covariance matrix," in *Proc. IEEE CDC*, 1979, pp. 761–766.

[8] I. H. Suh, J.-H. Kim, and F. C.-H. Rhee, "Convex-set-based fuzzy clustering," *IEEE Trans. Fuzzy Syst.*, vol. 7, no. 3, pp. 271–285, Jun. 1999.

[9] R. N. Dave, "Use of the adaptive fuzzy clustering algorithm to detect lines in digital images," *Proc. SPIE*, vol. 1192, pp. 600–611, 1989.

[10] R. N. Dave, "Fuzzy shell-clustering and application to circle detection in digital images," *Int. J. Gen. Syst.*, vol. 16, pp. 343–355, 1990.

[11] J. C. Bezdek and R. J. Hathaway, "Numerical convergence and interpretation of the fuzzy c-shells clustering algorithm," *IEEE Trans. Neural Netw.*, vol. 3, no. 5, pp. 787–793, Sep. 1992.

[12] Y. H. Man and I. Gath, "Detection and separation of ring-shaped clusters using fuzzy clustering," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 8, pp. 855–861, Aug. 1994.

[13] R. Krishnapurum, O. Nasraoui, and H. Frigui, "The fuzzy c spherical shells algorithm: A new approach," *IEEE Trans. Neural Netw.*, vol. 3, no. 5, pp. 663–671, Sep. 1992.

[14] R. N. Dave and K. Bhaswan, "Adaptive fuzzy C-shells clustering and detection of ellipses," *IEEE Trans. Neural Netw.*, vol. 3, no. 5, pp. 643–662, Sep. 1992.

[15] H. Frigui and R. Krishnapurum, "A comparison of fuzzy shell clustering methods for the detection of ellipses," *IEEE Trans. Fuzzy Syst.*, vol. 4, no. 2, pp. 193–199, May 1996.

[16] R. Krishnapurum, H. Frigui, and O. Nasraoui, "The fuzzy c quadratic shell clustering algorithm and the detection of second-degree curves," *Pattern Recog. Lett.*, vol. 14, pp. 545–554, 1993.

[17] R. Krishnapurum, H. Frigui, and O. Nasraoui, "Fuzzy and possibilistic shell clustering algorithms and their application to boundary detection and surface approximation—Part I," *IEEE Trans. Fuzzy Syst.*, vol. 3, no. 1, pp. 29–43, Feb. 1995.

[18] R. Krishnapurum, H. Frigui, and O. Nasraoui, "Fuzzy and possibilistic shell clustering algorithms and their application to boundary detection and surface approximation—Part II," *IEEE Trans. Fuzzy Syst.*, vol. 3, no. 1, pp. 44–60, Feb. 1995.

[19] I. Gath and D. Hoory, "Fuzzy clustering of elliptic ring-shaped clusters," *Pattern Recog. Lett.*, vol. 16, pp. 727–741, 1995.

[20] F. Hoeppner, "Fuzzy shell clustering algorithms in image processing: Fuzzy c-rectangular and 2-rectangular shells," *IEEE Trans. Fuzzy Syst.*, vol. 5, no. 4, pp. 599–613, Nov. 1997.

[21] X.-B. Gao, W.-X. Xie, J.-Z. Liu, and J. Li, "Template based fuzzy c-shells clustering algorithm and its fast implementation," in *Proc. IEEE Int. Conf. Signal Process.*, 1996, pp. 1269–1272.
[22] I. Gath and D. Hoory, "Detection of elliptic shells using fuzzy clustering: application to MRI images," in *Proc. Int. Conf. Pattern Recog.*, 1994, vol. 2, pp. 251–255.
[23] M. Barni and R. Gualtieri, "A new possibilistic clustering algorithm for line detection in real world imagery," *Pattern Recog.*, vol. 32, pp. 1897–1909, 1999.
[24] M. Barni, A. Mecocci, and L. Perugini, "Craters detection via possibilistic shell clustering," *Proc. IEEE Int. Conf. Image Process.*, vol. 2, pp. 720–723.
[25] A. Verikas, K. Malmqvist, and L. Bergman, "Detecting and measuring rings in banknote images," *Eng. Appl. Artif. Intell.*, vol. 18, pp. 363–371, 2005.
[26] S. Delbò, P. Gamba, and D. Roccato, "A fuzzy shell clustering approach to recognize hyperbolic signatures in subsurface radar images," *IEEE Trans. Geosci. Remote Sens.*, vol. 38, no. 3, pp. 1447–1451, May 2000.
[27] D. H. Ballard, "Generalizing the Hough transform to detect arbitrary shapes," *Pattern Recog.*, vol. 13, pp. 111–122, 1981.
[28] J. Illingworth and J. Kittler, "The adaptive Hough transform," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-9, no. 5, pp. 690–698, Sep. 1987.
[29] J. C. Bezdek, R. J. Hathaway, and N. R. Pal, "Norm-induced shell-prototypes (NISP) clustering," *Neural, Parallel Sci. Comput.*, vol. 3, pp. 431–449, 1995.
[30] L. O. Hall, I. B. Ozyurt, and J. C. Bezdek, "Clustering with a genetically optimized approach," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 103–112, Jul. 1999.
[31] T. Wang, "Possibilistic c-template clustering and its application in object detection in images," *Springer Lecture Notes Comput. Sci.*, vol. 4139, pp. 383–392, 2006.
[32] R. N. Dave and R. Krishnapuram, "Robust clustering methods: A unified view," *IEEE Trans. Fuzzy Syst.*, vol. 5, no. 2, pp. 270–293, May 1997.
[33] D. P. Bertsekas and J. N. Tsitsiklis, "Some aspects of parallel and distributed iterative algorithms—A survey," *Automance*, vol. 27, pp. 3–21, 1991.
[34] H. Kawanishi and M. Hagiwara, "A shape detection method using improved genetic algorithm," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, 1995, vol. 1, pp. 235–240.
[35] X. L. Yang, Q. Song, A. Z. Cao, S. Liu, and C. Y. Guo, "Robust c-shells based deterministic annealing clustering algorithm," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, 2004, vol. 3, pp. 1413–1417.
[36] A. K. Jain, Y. Zhong, and M. P. Dubuisson-Jolly, "Deformable template models: A review," *Signal Process.*, vol. 71, pp. 109–129, 1998.

Tsaipei Wang (M'05) received the B.S. degree in physics from the National Tsing Hua University, Hsinchu, Taiwan, in 1989, the M.S. degree in computer science from the University of Missouri–Columbia, in 2002, the Ph.D. degree in physics from the University of Oregon, Eugene, in 1999 and the second Ph.D. degree in computer engineering and computer science from the University of Missouri–Columbia, in 2005.

He is currently an Assistant Professor with the Department of Computer Science, National Chiao Tung University, Hsinchu. He was a Postdoctoral Fellow with the University of Missouri–Columbia. He was previously involved in research topics including frequency-selective optical memory, coherent transient phenomena, signal/image processing for landmine/tripwire detection, and automated vision development assessment with video photoscreening. His current research interests include fuzzy systems, image processing, medical image analysis, and visualization.