Estimating the Number of Clusters Using a Windowing Technique[1]

B. Boutsinas[a], D. K. Tasoulis[b], and M. N. Vrahatis[b]

[a] *Department of Business Administration, University of Patras Artificial Intelligence Research Center (UPAIRC), University of Patras, GR-26500 Rio, Patras, Greece*

[b] *Computational Intelligence Laboratory, Department of Mathematics, UPAIRC, University of Patras, GR-26110 Patras, Greece*

e-mail: vutsinas@upatras.gr, dtas@math.upatras.gr, vrahatis@math.upatras.gr

Abstract—Clustering is the process of partitioning a set of patterns into disjoint and homogeneous meaningful groups (clusters). A fundamental and unresolved issue in cluster analysis is to determine how many clusters are present in a given set of patterns. In this paper, we present the z-windows clustering algorithm, which aims to address this problem using a windowing technique. Extensive empirical tests that illustrate the efficiency and the accuracy of the proposed method are presented.

Key words: clustering algorithms, automatic cluster detection, data mining, unsupervised learning, range search.

DOI: 10.1134/S1054661806020015

1. INTRODUCTION

Clustering, that is, partitioning a set of patterns into disjoint and homogeneous meaningful groups (clusters), is a fundamental process in the practice of science. In particular, clustering is fundamental in knowledge acquisition. It is applied in various fields including data mining [19], statistical data analysis [3], compression, and vector quantization [36]. Clustering is also very important in most of the social sciences.

Recently, the task of extracting knowledge from large databases using clustering rules has become a subject of considerable interest. Clustering rules can be extracted using unsupervised learning methods. Algorithms for clustering have been widely studied in various fields including machine learning, neural networks, databases, and statistics. Clustering algorithms can be classified [3] as either hierarchical or iterative (partitional, density search, factor analytic or clumping, and graph theoretic). Complete-link, average-link, and single-link algorithms [17, 26, 41] are the most popular hierarchical clustering algorithms. k-Means [25, 35], along with its variants (e.g., [24, 38]) and hill climbing [7] are some of the most popular partitional clustering algorithms.

A fundamental issue in cluster analysis independent of the particular clustering technique is to determine the number of clusters that are present in the results of a clustering study. This fundamental issue is an unsolved problem in cluster analysis. For instance, popular iterative techniques, such as the k-means algorithm, require the user to specify the number of clusters present in the data prior to the identification of the clusters by the algorithm [25, 35].

There are a few symbolic and connectionist methods that attempt to tackle the problem of estimating the number of clusters. An obvious approach is to use hierarchical clustering. The user, the data analyst, can continue hierarchically clustering the data until a satisfactory level of clarity is achieved. There are some recent agglomerative hierarchical clustering algorithms that achieve high-quality clustering results, such as BIRCH [50], CHAMELEON [27], and CURE [22]. However, apart from a high user intervention and the typically nonlinear time complexity, this approach has serious drawbacks. For instance, the problem of determining the number of clusters is reduced to the problem of determining the level of the hierarchy at which to stop, although there are methods that try to address this problem (e.g., [29]). Additionally, dendrograms cannot be visualized for large data sets, as the diagram is too complex with too many leaf nodes and branches to allow comprehension. The main drawback, however, is that, although input data are organized into a strict hierarchy of nested subsets, there is no reason to believe that data actually follow a true hierarchical descent. The same is also true for other tree-based or lattice-based approaches [21], which suffer from the bias created by imposing a tree-structure or lattice-structure constraint on the solution of the clustering problem. The same should also be true for any method based on an imposed model of clusters. For instance, the important Bayesian Ying-Yang system [49] can automatically determine

[1] The text was submitted by the authors in English.

Received January 26, 2006

the number of clusters in the special case of a mixture of k Gaussian densities. Similarly, there are techniques for estimating the order of a Gaussian mixture model (order identification problem), as in [1, 37]. Cluster [11] is a clustering algorithm that is based on [37] and estimates the number of clusters that best fit the data. However, the special case of a mixture of Gaussian densities deviates from many practical situations. Notice that clustering methods with no structural constraint imposed on the solution require an exhaustive search for the nearest cluster, thus becoming impractical for large feature spaces.

Self-organizing maps (SOM) [28] can be categorized as a connectionist clustering technique for handling the problem. SOM can be used as a tool for mapping high-dimensional data into a two- or three-dimensional output map. The key idea of using SOM in clustering is to visually identify the clusters from that output map and thus gain some idea about the structure of the input data, due to the topology-preserving nature of SOM. By topology preserving, we mean that similar input patterns are projected into nodes that are close to each other in the output map and that nodes that are adjacent in the output map decode similar input patterns. Although the key idea is based on user intervention, a significant advantage could be gained if there was an accurate representation of the input space. However, there are serious limitations to this approach, since SOM, as has been shown theoretically [48], does not provide complete topology preservation. Moreover, the output map must be predetermined prior to the learning process. Notice that choosing a very large network may result in overfitting the training data, while choosing a very small network may not make it possible to capture the structure of the training data. Usually, the proper output map is known only after several trials. Attempting to overcome these limitations, a number of approaches based on dynamic self-organizing maps have been proposed that determine the shape and the size of the network during training. These networks "grow" as they learn and "shrink" as they forget.

The GCS algorithm [20] is based on an incremental self-organizing network with a k-dimensional simplex as an initial topology. During training, new cells are added and superfluous cells are removed according to a heuristic criterion that takes into account the relative winning frequency of a node. The GCS algorithm uses a drawing method, which works well with low-dimensional data, but there are problems in visualizing high-dimensional data, since, in this case, the mapping is not always planar.

The TRN (neural gas) algorithm [31] starts with a network of a fixed number of cells with no connections. During training, cells are adapted and connections are created between the winning cells and the closest competitor, while cells are removed according to an aging mechanism. The predetermined number of cells and the dependence on the respective locality of the input data result in visualization difficulties.

The IGG algorithm [10] starts with a small number of initial cells. During training, it generates cells from the boundary of the network according to a growth heuristic. Connections are added and removed according to an internode weight difference. However, there is a difficulty in adapting its structure in the case of a dimension mismatch between the input space and the output map.

Obviously, the aforementioned dynamic self-organizing algorithms, along with their extensions (e.g., GSOM [2], DTRN [40]), become more complex compared to their fixed structure counterparts. Moreover, they do not render an intuitive representation of the output map to visualize the final clustering results in high-dimensional spaces.

There are also approaches that adjust the number of clusters during training. The ISODATA technique [8] is based on the same key idea as the k-means algorithm; starting with a typical number of initial clusters, it iteratively merges and splits existing clusters according to "within-group variability" and "closeness" thresholds. Adaptive resonance theory (ART) algorithms [13, 14] grow cells according to a vigilance threshold that actually defines the cluster sizes (when it is small, it produces large clusters, and as it gets larger, it yields finer clusters). However, the effectiveness of these approaches is heavily dependent on the value of these thresholds, which is usually based on a tacit knowledge about the structure of the input data.

Finally, there are methods that try to tackle the problem by letting the user specify a range within which the true presumable k lies [33].

From our experience, the most promising approach is the DBSCAN algorithm [18]. DBSCAN is a density-based clustering algorithm that tries to recover clusters from spatial databases. Clusters are defined by means of neighborhoods of objects. The density of each accepted such neighborhood has to exceed some threshold. Although this threshold is very crucial for the algorithm, heuristics have been proposed to determine it.

In this paper, we present the z-windows algorithm, an iterative clustering technique, aiming to solve the problem of determining the number of clusters. The z-windows algorithm is based on the windowing technique of the k-windows clustering algorithm [47], used to reduce the number of patterns that need to be examined for similarity at each iteration. k-Windows utilizes the orthogonal range search technique of computational geometry. The key idea is to use a sufficiently large number of initial windows, which are properly merged during the algorithm. The windowing technique of the k-windows algorithm allows a large number of initial groups (windows) to be examined without a significant overhead in time complexity. The merge operation is guided by a certain threshold, which is set by the user.

However, the effectiveness of the z-windows algorithm largely depends on the initial windows that can be determined by the range search tree, thus taking directly into consideration the structure of the input data.

The rest of the paper is organized as follows. The k-windows algorithm is briefly described in Section 2, along with its computational complexity. The proposed z-windows algorithm is described in Section 3. In Section 4, we present extensive empirical tests that illustrate the utility of the method. The paper ends with some concluding remarks and a short discussion about further research.

2. THE *K*-WINDOWS ALGORITHM

The k-windows algorithm [47] is an improvement of the *k-means* algorithm, which is a very popular algorithm particularly suited for implementing the clustering process, because of its ability to efficiently partition large amounts of data.

k-Means consists of two main phases. At the first phase, a partition of patterns into k clusters is calculated, while during the second stage, the quality of the partition is determined. k-Means is implemented by an iterative process that starts from a random initial partition. The latter is repeatedly recalculated until its quality function reaches an optimum.

In particular, the whole process is built upon four basic steps:

(1) selection of the initial k centers;

(2) assignment of each pattern to a cluster with the nearest center;

(3) recalculation of k centers of the clusters; and

(4) computation of the quality function.

The last three steps are performed iteratively until convergence.

The k-means algorithm is computationally very expensive for large sets of patterns. It requires time proportional to the product of the number of patterns, the number of clusters, and the number of iterations. More specifically, the assignment step of each pattern to the cluster with the nearest center (mean) is computationally the most expensive. This is imposed not only by its time complexity in relative terms, but also by its basic operation, which is the calculation of the Euclidean distance.

The k-windows algorithm addresses this problem by employing a windowing technique, which allows the consideration of only a limited number of patterns in each iteration. Moreover, the basic operation in the first loop, during the assignment of patterns to clusters, is now only the arithmetic comparison between two numbers.

The key idea behind the k-windows algorithm is to use a window in order to determine a cluster. The window is defined as an orthogonal range in d-dimensional Euclidean space, where d is the number of numerical attributes. Therefore, each window is a *d-range* and has a fixed size. Every pattern that lies within a window is considered to belong to the corresponding cluster. Each window is iteratively moved in the Euclidean space by centering itself on the mean of the patterns it includes. This takes place until any further movement does not result in any increase in the number of patterns that lie within it (see solid line squares in Fig. 1). After that step, we are able to determine the means of clusters as the means of the points within the corresponding windows. However, since only a limited number of patterns is considered at each movement, the quality of a partition may not be optimal. Thus, the quality of a partition is calculated in a second phase. At first, windows are enlarged in order to contain as many patterns from the corresponding cluster as possible (see dotted-line squares in Fig. 1). The quality of a partition is determined by the number of patterns contained in any window, with respect to all the patterns.

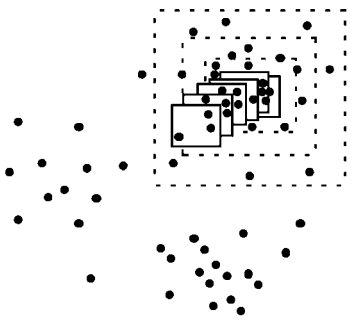

Fig. 1. Movements and enlargements of a window.

The k-windows clustering algorithm operates as follows.

Algorithm k-windows.

input k, a, v

initialize k means $i_{m1}, \ldots, i_{mk}$ along with their k d-ranges $w_{m1}, \ldots, w_{mk}$ each of area a

repeat

  for each input pattern i_l, $1 \le l \le n$

    do

      assign i_l to w_j, so that i_l lies within w_j

  for each d-range w_j

    do

      calculate its mean $i_{mj} = \frac{1}{|w_j|}\sum_{i_l \in w_j} i_l$

      and recalculate d-ranges

until no pattern has changed d-ranges

enlarge d-ranges until no significant change exists in their initial mean

compute the ratio $r = \frac{1}{n}\sum_{j=1}^{k} |i_l \in w_j|$

if $r < v$ **then** re-execute the algorithm

At first, k means are selected (possibly in a random fashion). Initial d-ranges (windows) have as centers these initial means and each one is of area a. Then the mean of the patterns that lie within each range is calculated. Each such mean defines a new d-range, which is considered a movement of the previous d-range. The last two steps are executed repeatedly until there is no d-range that includes a significant increment of patterns after a movement.

In the second phase, the quality of the partition is calculated. At first, the d-ranges are enlarged in order to include as many patterns as possible from the cluster. Then the relative frequency of patterns assigned to a d-range in the whole set of patterns is calculated. If the relative frequency is small, then, possibly, there may be a missing cluster (or clusters). In this case, the whole process is repeated.

The computational complexity of the k-windows algorithm depends on the complexity of the step that determines the patterns that lie within a d-range. This is the *orthogonal range search* problem, formally stated as:

• **Input:**

(a) $V = \{p_1, \ldots, p_n\}$ is a set of n points in $\mathbb{R}^d$, the d-dimensional Euclidean space with coordinate axes $(Ox_1, \ldots, Ox_d)$;

(b) a query d-range $\mathcal{Q} = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d]$ is specified by two points $(a_1, a_2, \ldots, a_d)$ and $(b_1, b_2, \ldots, b_d)$, with $a_j \le b_j$.

• **Output:**

report all points of V that lie within the d-range $\mathcal{Q}$.

This problem has been addressed by various computational geometry techniques [4, 9, 15, 23, 34]. The common base of these techniques is the construction of a data structure storing the patterns at the preprocessing stage. Traversing this data structure can provide fast answers to range queries. In detail, for applications of very high dimensionality, data structures like the multidimensional binary tree (MBT) [34] and Bentley and Maurer [9] seem more suitable. On the other hand, for low-dimensional data with a large number of points, the approach of Alevizos [4] appears more attractive. In the context of the k-windows algorithm, two such approaches, namely the MBT and the range tree [34], were compared in [5].

The time complexity of k-means amounts to $O(ndkt)$, where t represents the performed iterations. Comparing this with the time complexity for the k-windows algorithm, using a range tree [34] as a data structure, it is reduced to $O\left(dkqr\left(\frac{\log^{d-2} n}{d} + s\right)\right)$, where q and r stand for the number of movement and enlargement operations, respectively. The product qr is empirically shown to be proportional to t [47].

3. THE z-WINDOWS ALGORITHM

The key idea is to apply the k-windows algorithm using a sufficiently large number of initial windows. The windowing technique of the k-windows algorithm allows for a large number of initial windows to be examined, without any significant overhead in time complexity. Then, any two overlapping windows are merged. The remaining windows, after the quality of the partition criterion is met, define the final set of clusters.

At first, the initial windows are determined using the range tree (see below Step 2 of the z-windows algorithm). Internal nodes of the same level in the range tree can be used as representatives of different subsets of patterns. We use those nodes as the centers of the initial windows. We have tried various heuristic initialization procedures, which are based on the range tree, in order to choose the initial windows. The two most effective are discussed in the next section.

The initial windows cover an initial area a (Step 3) that depends on the density of the data set. Similarly to k-windows, we define across each direction i,

$$a_i = \frac{(mean\ distance\ among\ patterns\ in\ i)}{(number\ of\ windows)} \times 0.5.$$

Intuitively, we try to fill the mean space between two patterns with nonoverlapping windows (thus, we scale by 0.5).

Algorithm z-windows.

1. input a, u, th_e, th_m, th_c, th_v

2. z=DetermineInitialWindows()

3. initialize z d-ranges $w_{m1}, \ldots, w_{mz}$ each of area a along with their means $i_{m1}, \ldots, i_{mz}$

4. for each input pattern i_l, $1 \le l \le n$, **do**

assign i_l to w_j, so that i_l lies within w_j

repeat

5. for each d-range w_j **do**

calculate its mean $i_{mj} = \frac{1}{|w_j|} \sum_{i_l \in w_j} i_l$

and recalculate d-ranges

6. for each d-range w_j **do**

repeat

for each dimension d_i **do**

repeat

enlarge w_j across d_i for th_e%

until increase in number of patterns across d_i is less than th_c%

until increase in number of patterns is less than th_c% across every d_i

until no significant change ($<th_v$) of the means of d-ranges takes place

7. for each d-range w_j not marked **do**

mark w_j with label l_j

if $\exists w_i \neq w_j$ that overlaps with w_j

then mark w_i with label l_j

8. compute the ratio $r = \frac{1}{n}\sum_{j=1}^{z} |i_l \in w_j|$

9. if $r < u$ **then** re-execute the algorithm

10. for each input pattern i_l, $1 \le l \le n$, **do**

assign i_l to w_j with nearest mean i_{mj},

such as $\|i_l - i_{mj}\|^2 \le \|i_l - i_{mu}\|^2$, $1 \le j, u$

11. output clusters c_{l_1}, c_{l_2}, … such as $c_{l_i} = \{i | i \in w_j$ with label $l_i\}$

Then, iteratively, each window is moved in the Euclidean space by centering itself on the mean of the patterns included (Steps 4 and 5) until no further movement results in an increase in the number of patterns that lie within it. This criterion is determined by a variability threshold th_v that defines the least change in the center of a window that can be considered as a legitimate movement.

Then, within the loop, the enlargement phase takes place. We enlarge windows in order to contain as many patterns from the corresponding cluster as possible (Step 6). The enlargement is a gradual process, where every window is gradually enlarged, by th_e percent across every coordinate. th_e is an enlargement threshold, which is set by the user (Step 1). The enlargement phase continues until any further enlargement does not increase the number of patterns that lie within a window. This criterion is determined by a coverage threshold, th_c, that is set by the user (Step 1). The coverage threshold, th_c, defines the least increase (expressed as a ratio) in the number of patterns of a window that can be considered significant.

Movement and enlargement phases are executed, one after the other, iteratively until no significant change to the means of d-ranges, with respect to their previous state, takes place, according to the variability threshold th_v mentioned above.

After moving and enlarging windows, a search for overlapping windows is performed (Step 7). Every pair of overlapping windows is substituted by one of them. A straightforward criterion to identify windows to be merged is whether they simply overlap. Alternatively, we tested various other heuristics. The most efficient, as is shown by the experimental results, is to grade overlap according to the number of patterns that lie in the intersection. Specifically, the merge operation can be guided by a merge threshold, th_m, which can be set by the user (Step 1), so that, if

$$\frac{1}{2}\left(\frac{|i_l \in w_i \wedge i_l \in w_j|}{|w_i|} + \frac{|i_l \in w_i \wedge i_l \in w_j|}{|w_i|}\right) > th_m,$$

then w_j and w_i must be merged. Intuitively, merging is guided by how much one window is embodied in another.

Additionally, various other heuristics can be used in the merging process, as far as treating the overlapping windows is concerned. We tested several such heuristics. For instance, merging the two overlapping windows to form a larger one leads to a small number of very large clusters. Alternatively, the trivial heuristic to delete one of the two overlapping windows is more effective compared to the previous one. The proposed heuristic is a combination of the above: both windows are retained, but they are considered to belong to the same cluster, and, hence, they are assigned the same label.

The last phase concerns examining the quality of the partition. As in the case of k-windows, the quality of a partition is determined by the number of patterns contained in each window, with respect to all patterns (Steps 8 and 9). If the desired quality, set by the user through the parameter u, is not achieved, the whole algorithm is re-executed. We are currently investigating various approaches that can be used to guide such re-executions. For instance, a new re-execution can be started with the same initial means but with d-ranges of a larger area $a' > a$. Another approach is to start with different initial means located at a maximum distance from the previous ones.

However, as the number of initial window grows, the possibility that there would not be any missing cluster decreases. In all of the tests we performed, there was no need for re-executing the algorithm. In conclusion, the value of u, according to our experience, does not play any role, and we use it only for the sake of completeness.

Finally, we define the output clusters. First, each input pattern is assigned to a group determined by the window with the nearest mean to the pattern (Step 10). Then, the algorithm outputs the final clusters that consist of patterns that lie within windows with the same labels (Step 11). After determining the clusters, we can easily detect clusters of noise patterns, since the typical density of patterns inside the clusters must be considerably higher than that outside the clusters. The same property is also used by the DBSCAN algorithm.

The z-windows algorithm is very fast compared to other related algorithms, since it achieves sublinear time complexity with respect to the total number of patterns. The assignment of patterns to a d-range, using a d-dimensional range tree, needs $O(s + \log^{d-2} n)$ time, where s is the number of patterns that lie within the d-range. Notice that the area of d-ranges is small enough, so that $s \ll n$. Therefore, the first loop in Step 4, where the patterns are assigned to d-ranges, has time complexity $O(z(s + \log^{d-2} n))$. The second loop in Step 5, where the means of d-ranges are calculated, needs $O(sdz)$. The third loop in Step 6, where, during the enlargement of d-ranges, assignments of patterns to a

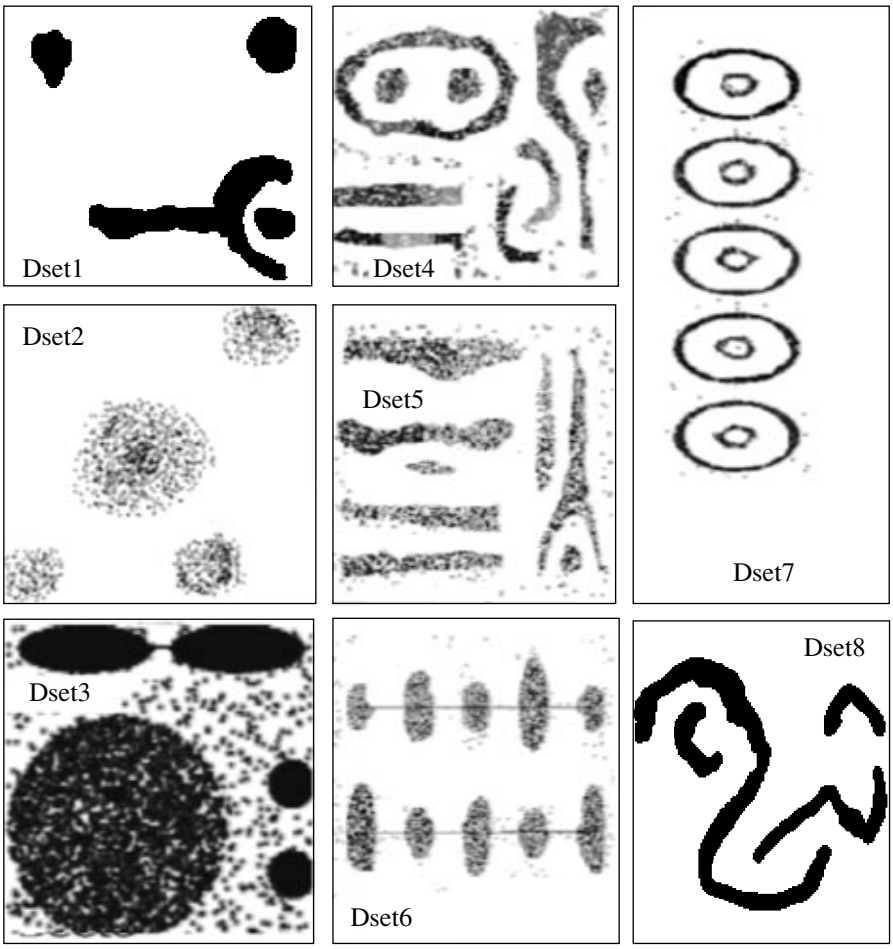

Fig. 2. The eight synthetic sample databases.

d-range take place, has time complexity $O(z(s+\log^{d-2}n))$. Finally, the quality function is also calculated in $O(sdz)$. Thus, the proposed algorithm has time complexity $O\left(dzqr\left(\frac{\log^{d-2}n}{d}+s\right)\right)$, where q is the number of movements and r is the number of repetitions caused by missing clusters (Step 9). Notice that, due to deletions of d-ranges in Step 7, it gradually decreases. Finally, the basic operation regarding the time complexity of the z-windows algorithm is the arithmetic comparison between two numbers without any distance computation, which is cheap in computational terms.

4. EMPIRICAL TESTS

From a theoretical standpoint, the proposed z-windows algorithm has a lower time complexity with respect to other clustering algorithms. Moreover, it achieves high-quality clustering results. Since we could not compare our method to the great number of k-means variations and centroid/medoid based methods, due to space limitations, we chose to simply resort to visual inspection. We applied the z-windows algorithm to various two-dimensional synthetic sample databases that have already been used as test data sets to evaluate BIRCH, CHAMELEON, CLARANS, CURE, and DBSCAN (e.g., in [22, 27, 39]). We have also applied the z-windows algorithm to high-dimensional databases.

To evaluate the z-windows algorithm, we implemented a system in the C++ language under the Linux operating system with the “gcc ver. 3.3.2" compiler. Using this system, we applied z-windows to eight two-dimensional synthetic sample databases and six high-dimensional databases. The sample databases (DSet1–8) are depicted in Fig. 2 and include clusters with both normal and irregular shape. We also applied k-means for different values of k to those datasets. The DSet9–12 are high-dimensional databases (5D, 12D, 20D, and 50D, respectively) which are generated as mixtures of Gaussian densities by using a multidimensional Gaussian random number generator which was built using the "rnmvn” routine from the IMSL library of Fortran90.

We also applied the cluster algorithm [11] to DSet9–12 datasets. The high-dimensional databases, DSet13–14, represent two images that are stored and displayed

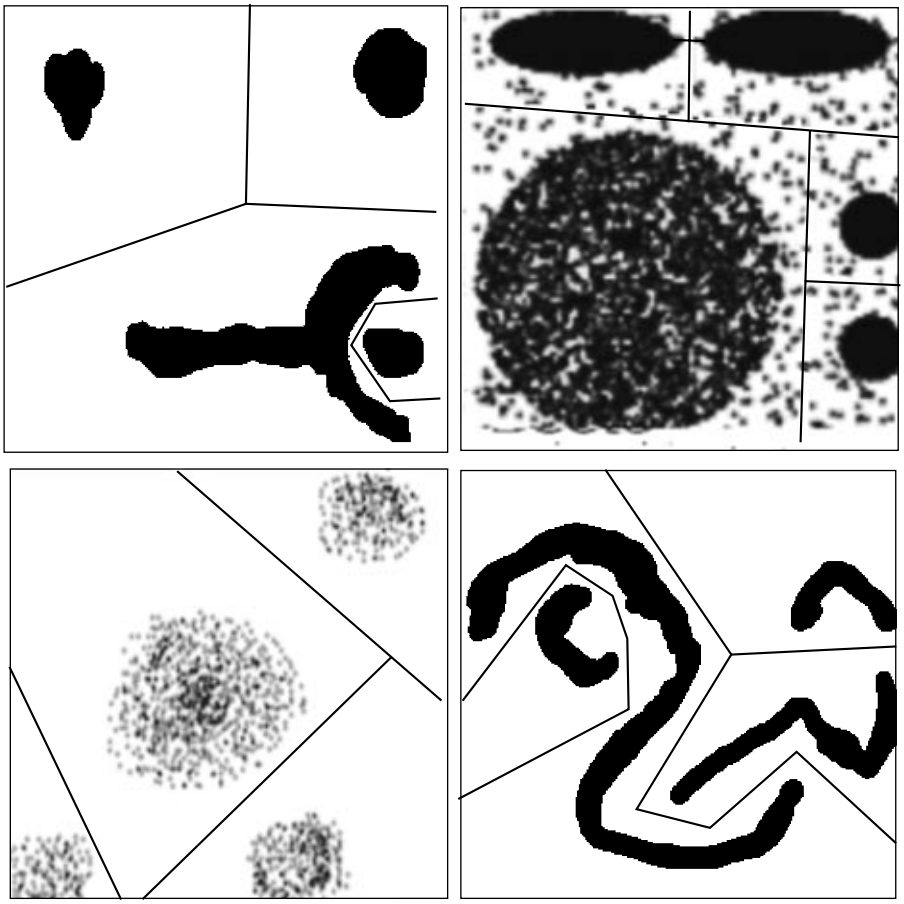

Fig. 3. Clusters discovered by z-windows algorithm from two-dimensional sample databases.

in the RGB space. In both images, the color of each pixel follows red/green/blue (RGB) color specification (three 0 to 255 ASCII numbers indicating red, green, and blue). Thus, each pixel on the grid is represented by a three-dimensional vector, corresponding to its RGB values. The first one is a color retinal image that was obtained using a retinal camera with a field of view of 45 and a 760×570 resolution in 24-bit RGB. Such retinal images can be used in detecting retinal exudates. Retinal exudates are typically manifested as spatially random yellow/white patches of varying sizes and shapes, and they are a characteristic feature of retinal diseases such as diabetic maculopathy [32]. The second image, with a 255×192 resolution, is a test image that is used for color image segmentation. Color image segmentation can be seen as an example of feature space analysis, which is a widely used tool for solving low-level image understanding tasks [16]. Given an image, feature vectors are extracted from local neighborhoods and mapped into the space spanned by their components. Significant features in the image then correspond to high-density regions in this space. Feature space analysis is the procedure of recovering the centers of the high-density regions, i.e., the representations of the significant image features. Color image segmentation, that is partitioning the image into homogeneous regions, is a challenging task [16].

Clusters discovered by z-windows in the synthetic two-dimensional sample databases are shown in Figs. 3 and 4. Note that z-windows achieve high-quality clustering results even in cases where the results of other clustering algorithms vary quite dramatically. For instance, some algorithms are unsuitable for nonspherical or nonconvex clusters (e.g., BIRCH and CLARANS). Additionally, if there are large differences in the sizes of clusters (e.g., in DSet3), partitional clustering algorithms based on the square-error measure may split large clusters to minimize it. Also, if the clusters are close to one another (e.g., in DSet4 or DSet5), agglomerative hierarchical clustering algorithms may merge portions belonging to neighboring clusters. Finally, if there is a dense string of points connecting two clusters (e.g., in DSet6), both single-link hierarchical clustering algorithms and density-based clustering algorithms (e.g., DBSCAN) may merge the two clusters.

It is obvious from Figs. 3 and 4 that z-windows copes with all those cases.

Clusters discovered by k-means in the DSet1 and DSet8 synthetic two-dimensional sample databases are shown in Fig. 5, for values of k found by applying

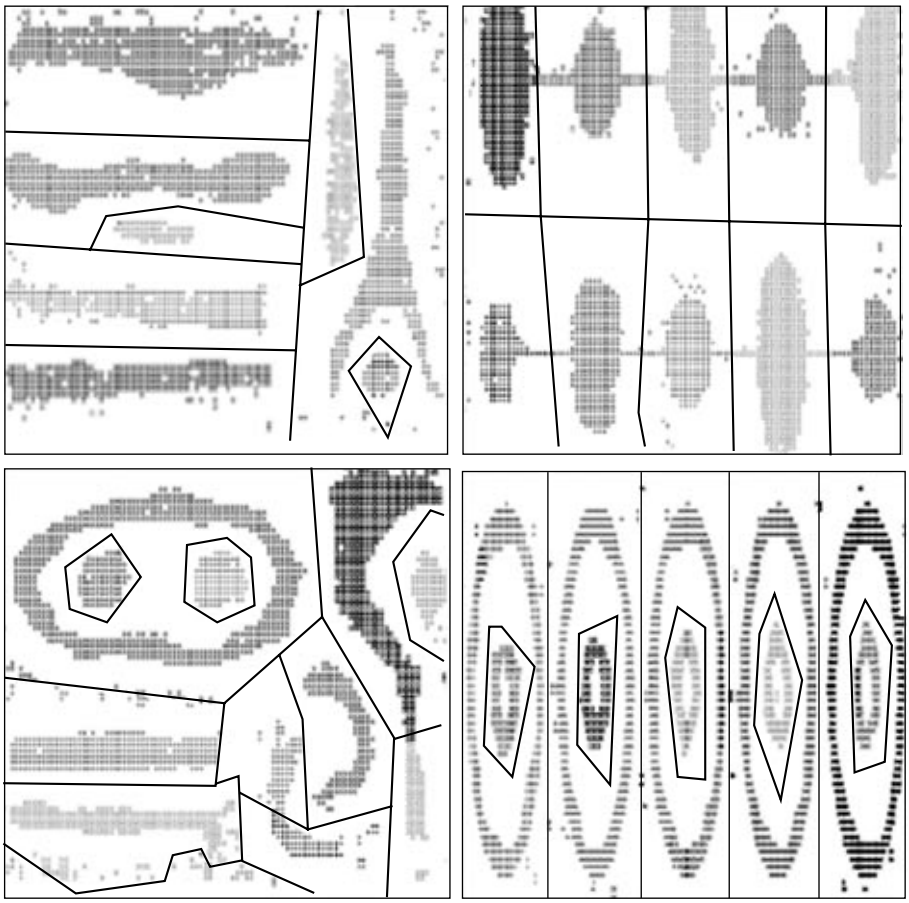

Fig. 4. Clusters discovered by *z*-windows algorithm from two-dimensional sample databases.

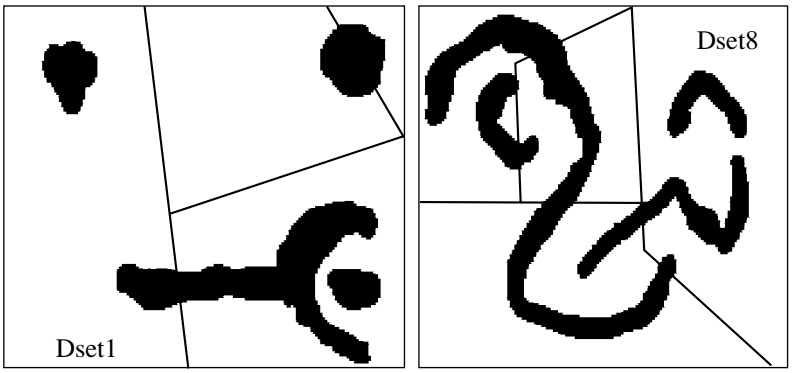

Fig. 5. Clusters discovered by *k*-means algorithm from datasets DSet1 and DSet8.

z-windows on the same databases. Notice that some self-organizing algorithms have the same difficulty with *k*-means in datasets with a wide spread distribution. They seem to have become stuck at a minimum that is closer to the mass center [30].

Clusters discovered by *k*-means and *z*-windows are taken from our implementation of the algorithms. Notice that, as far as the *k*-means algorithm is concerned, we used a primitive initialization method that consists in pre-executing *k*-means in a sample of the data in order to refine the initial points. If this primitive initialization method was not used, the partition accuracy of the *k*-means algorithm would be worse.

Empirical tests aim also at examining the sensitivity of *z*-windows to some of its critical parameters. There are six user-defined parameters (a, u, th_e, th_m, th_c, and th_v). If clusters in the dataset are convex, none of them seems to contribute to the quality of the clustering.

Table 1. Range of parameter values for identical results for data set DSet2

Threshold	From	To
Variability, th_v	0.2	1.0
Coverage, th_c	0.1	0.8
Enlargement, th_e	0.0	1.0

However, even when there exist clusters with irregular shape:

(a) initial area a (Step 1) is automatically calculated across each dimension i as a_i = (*mean distance among patterns in i*)/(*number of windows*) × 0.5;

(b) in all the tests we made, the algorithm was never re-executed. Thus, u is needed only to exceptional and not practical cases;

(c) variability (th_v), coverage (th_c), and enlargement (th_e) thresholds do not seem to significantly contribute to the quality of the clustering. In all the tests presented in the paper, they were set to a fixed value. Of course, if they were set to large values, e.g., 10, then the quality of the clustering would be different. Table 1 shows the range of values of these thresholds that lead to results identical to those in Fig. 3 for data set DSet2;

(d) the number of identified clusters depends only on the merge threshold (th_m) that actually determines the clarity of the clustering. We have investigated the effect of this parameter on the quality of the clustering, applying the proposed z-windows algorithm to the same test database with the merge threshold set to different values. In Fig. 6, the number of discovered clusters is depicted as a function of the merge threshold for the dataset DSet2. It is obvious that the merge threshold determines the number of discovered clusters.

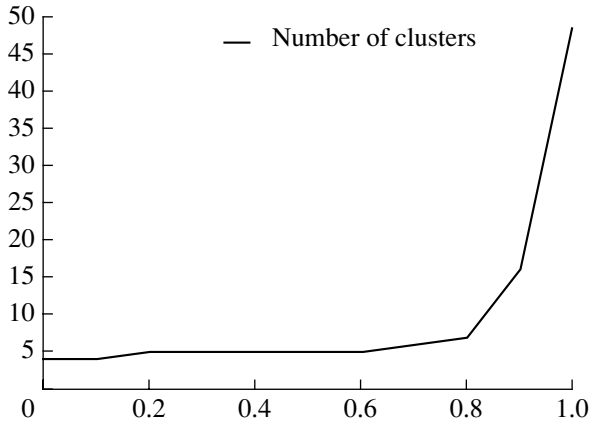

Fig. 6. Number of discovered clusters in DSet2 as a function of the merge threshold.

Table 2. The DSet9–12 high-dimensional databases

	Dimensions	Size	Number of noise patterns
DSet9	5	50000	5000
DSet10	12	2600	900
DSet11	20	2600	900
DSet12	50	2600	900

We have also compared the z-windows algorithm to algorithms that can automatically determine the number of clusters in the special case of a mixture of Gaussian densities. The characteristics of DSet9–12 high-dimensional databases are shown in Table 1. Notice that the noise was generated using a covariance matrix with large values. The projection of each dataset in two dimensions looks like what is shown in Fig. 7. There are four clusters in every high-dimensional database, which have been discovered by both z-windows and cluster algorithms. However, the cluster algorithm requires the "diag" parameter to be properly set in order to discover those clusters. Moreover, in any case, a fifth cluster was discovered by the cluster algorithm due to the presence of noise patterns.

Finally, clusters discovered by z-windows from DSet13–14 are shown in Figs. 8 and 9. Each pixel was colored exactly as the mean of the cluster containing the pixel. The means of the clusters were randomly colored. Notice that, although there are 150 different three-dimensional vectors in the first image and 256 in the second one, only five clusters were discovered in

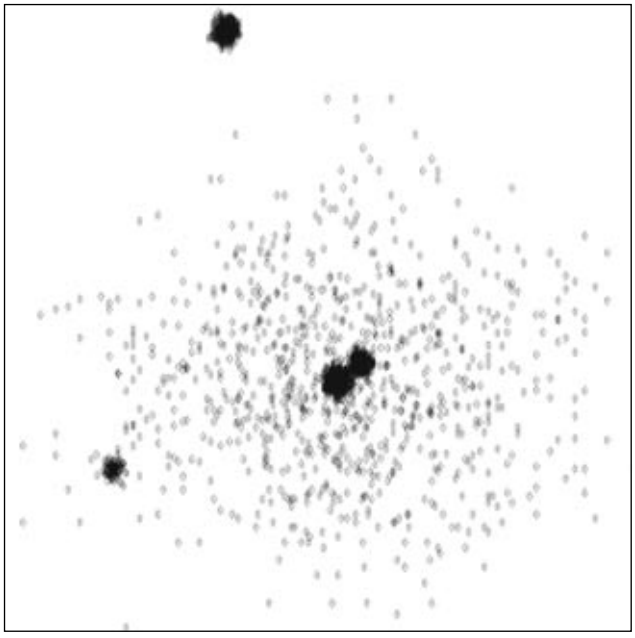

Fig. 7. The projection of DSet9–12 datasets in two dimensions.

Fig. 8. Segmented image by z-windows algorithm.

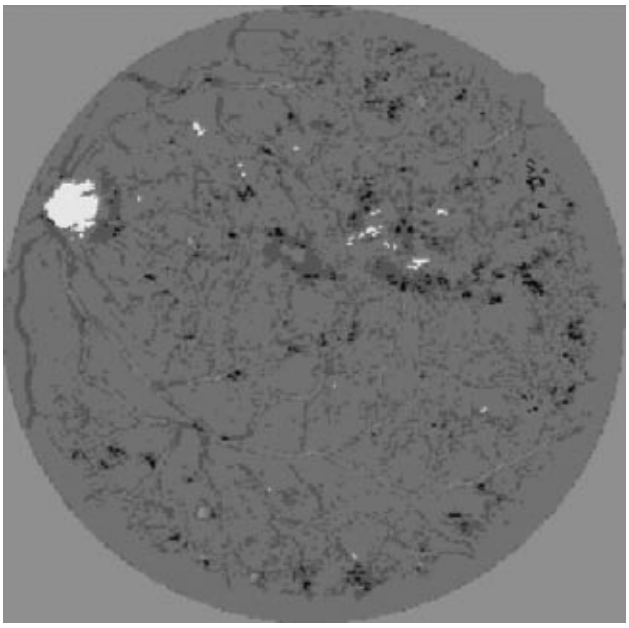

Fig. 9. Colored image by z-windows algorithm.

the first case and twenty-five clusters in the second case. In the first image, candidate retinal exudates are shown in white. In the second image, there is a very high degree of correspondence between the obtained image and the reference image.

We have also investigated the effect of the number of initial windows. We tested various initialization processes. One such process, for instance, is based on traversing the range tree. It starts from the first level and iteratively assigns a window to each different node of each level. The initialization process ends if the number of the windows assigned to a specific level is the same as the number of the windows assigned to the previous level. The most efficient initialization process we tested assigns a window for each node of the middle interior level of the range tree. Notice that all the tests presented so far were made adopting the latter initialization process.

It is obvious that a greater number of initial windows considerably improves the quality of the clustering at the expense of a worse time performance. Figure 10 shows the time performance as a function of the number of initial windows for data set DSet2.

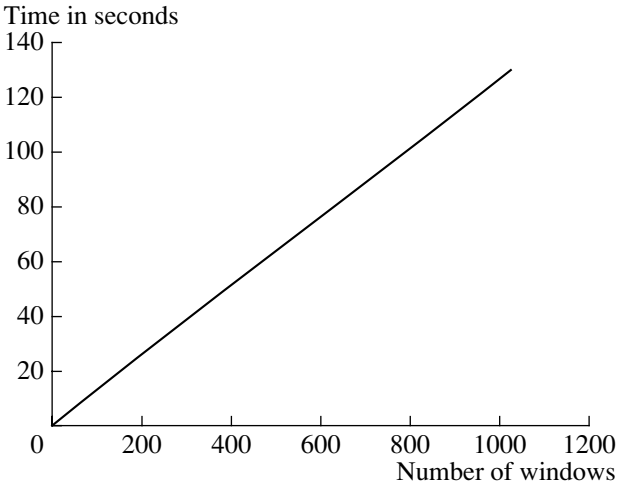

Fig. 10. Performance as a function of the number of initial windows for the data set DSet2.

As a conclusion, from all of the tests we performed, it seems that both the merge threshold and the number of initial windows are the critical factors to the quality of the clustering.

5. CONCLUSIONS AND FUTURE WORK

The proposed z-windows algorithm is an iterative clustering technique that tackles the problem of determining how many clusters are present in the results of a given clustering study. The z-windows algorithm is based on the k-windows clustering algorithm, which, by employing a windowing technique, reduces time complexity of the k-means algorithm. The key idea is to use a sufficiently large number of initial windows, which are properly merged during the algorithm.

The k-windows algorithm has already been extended to be applicable to categorical data [46]. Moreover, parallel and distributed versions of the k-windows algorithm have been presented [6, 43, 44]. A simple idea is to assign a different processor for each window. Notice that, under such an assignment scheme, k-windows can efficiently be parallelized, in contrast to k-means that would require a large communication overhead. This is because a processor dedicated to a cluster in k-means must be synchronized with all the others before the assignment of a pattern. There is no such need in k-windows, where the decision of assigning a pattern to a d-range is taken by each processor independently. Since the d-ranges are enlarged in a

parallel setting, there may be overlaps between them which can be treated by the merging process described above. Finally, the algorithm has been extended [45] to be applicable in databases that undergo update operations such us deletions and insertions. All the above extensions of the k-windows algorithm could also be applied to the proposed z-windows algorithm.

Moreover, a lot of heuristics can be used to improve the z-windows algorithm. However, the version proposed in this paper is effective in most cases. For instance, we are investigating the combination of the z-windows algorithm with collaborative filtering. If cluster identification is based on windows and on previously known and recognized categories, e.g., using category-based filtering [42], the estimate of the number of clusters may be improved.

REFERENCES

1. H. Akaike, "A New Look at the Statistical Model Identification," *IEEE Trans. Automat. Contr. AC-19, 1974*, pp. 716–723.
2. D. Alahakoon, S. K. Halgamuge, and B. Srinivasan, "Dynamic Self-Organizing Maps with Controlled Growth for Knowledge Discovery," IEEE Trans. Neural Networks **11** (3), 601–614 (2000).
3. M. S. Aldenderfer and R. K. Blashfield, "Cluster Analysis," in *Ser. Quantitative Applications in the Social Sciences* (SAGE Publications, London, 1984).
4. P. Alevizos, "An Algorithm for Orthogonal Range Search in $d \geq 3$ Dimensions," *Proc. of the 14th European Workshop on Computational Geometry, Barcelona, 1998.*
5. P. Alevizos, B. Boutsinas, D. K. Tasoulis, and M. N. Vrahatis, "Improving the Orthogonal Range Search k-Windows Clustering Algorithm," *Proc. of the 14th IEEE Int. Conf. on Tools with Artificial Intelligence, Washington, D.C., 2002*, pp. 239–245.
6. P. Alevizos, D. K. Tasoulis, and M. N. Vrahatis, "Parallelizing the Unsupervised k-Windows Clustering Algorithm," Lecture Notes Comp. Sci. **3019**, 225–232 (2004).
7. M. Anderberg, *Cluster Analysis for Applications* (Academic Press, New York, 1973).
8. G. H. Ball and D. J. Hall, "A Clustering Technique for Summarizing Multivariate Data," Behav. Sci. **12**, 153–155 (1967).
9. J. L. Bentley and H. A. Maurer, "Efficient Worst-Case Data Structures for Range Searching," Acta Inform. **13**, 1551–1568 (1980).
10. J. Blackmore and R. Miikkulainen, "Visualizing High-Dimensional Structure with the Incremental Grid Growing Neural Network," *Proc. of the 12th Int. Conf. on Machine Learning, 1995*, pp. 153–155.
11. C. A. Bouman and M. Shapiro, "A Multiscale Random Field Model for Bayesian Image Segmentation," IEEE Trans. Image Processing **3** (2), 162–177 (1994).
12. B. Boutsinas and T. Gnardellis, "On Distributing the Clustering Process," Pattern Recognit. Lett. **23** (8), 999–1008 (2002).
13. G. A. Carpenter and S. Grossberg, "A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine," Comp. Vis. Graph. Image Processing **37**, 54–115 (1987).
14. G. A. Carpenter and S. Grossberg, "A ART-2: Self-Organization of Stable Category Recognition Codes for Analog Input Patterns," Appl. Opt. **26** (23), 4919–4930 (1987).
15. B. Chazelle, "Filtering Search: A New Approach to Query-Answering," SIAM J. Comput. **15** (3), 703–724 (1986).
16. D. Comaniciu and P. Meer, "Robust Analysis of Feature Spaces: Color Image Segmentation," *IEEE Conf. Computer Vision and Pattern Recognition (CVPR'97), San Juan, Puerto Rico, 1997*, pp. 0–755.
17. R. C. Dubes and A. K. Jain, "Clustering Methodologies in Exploratory Data Analysis," Adv. Comput. **19**, 113–228 (1980).
18. M. Ester, H. P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," *Proc. of 2nd Int. Conf. on Knowledge Discovery and Data Mining, 1996*, pp. 226–231.
19. U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, *Advances in Knowledge Discovery and Data Mining* (MIT Press, 1996).
20. B. Fritzke, "Growing Cell Structure: A Self-Organizing Network for Supervised and Unsupervised Learning," Neural Networks **7** (9), 1441–1460 (1994).
21. A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression* (Kluwer Academic Publishers, Boston, MA, 1992).
22. S. Guha, R. Rastogi, and K. Shim, "CURE: An Efficient Algorithm for Clustering Large Databases," *Proc. of ACM-SIGMOD 1998 Int. Conf. on Management of Data, Seattle, 1998*, pp. 73–84.
23. B. Chazelle and L. J. Guibas, "Fractional Cascading: II. Applications," Algorithmica **1**, 163–191 (1986).
24. Z. Huang, "Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values," Data Mining Knowledge Discovery **2**, 283–304 (1998).
25. A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data* (Prentice-Hall, Englewood Cliffs, NJ, 1988).
26. S. Johnson, "Hierarchical Clustering Schemes," *Phychometrika* (1967), pp. 241–254.
27. G. Karyapis, E. H. Han, and V. Kumar, "CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling," IEEE Computer Special Issue on Data Analysis and Mining **32** (8), 68–75 (1999).
28. T. Kohonen, *Self-Organizing Map* (Springer, Heidelberg, 1995).
29. R. Kothari and D. Pitts, "On Finding the Number of Clusters," Pattern Recognit. Lett. **20**, 405–416 (1999).
30. S. Lin and J. Si, "Weight Value Convergence of the SOM Algorithm for Discrete Input," Neural Comput. **10** (4), 807–814 (1998).
31. T. Martinetz, S. Berkovich, and K. Schulten, "Neural-Gas Network for Vector Quantization and Its Application to Time-Series Prediction," IEEE Trans. Neural Networks **4** (4), 558–569 (1993).
32. A. Osareh, M. Mirmehdi, B. Thomas, and R. Markham, "Automatic Recognition of Exudative Maculopathy Using Fuzzy c-Means Clustering and Neural Networks," *Proc. Medical Image Understanding and Analysis Conference* (BMVA Press, 2001), pp. 49–52.

33. D. Pelleg and A. Moore, "X-Means: Extending *k*-Means with Efficient Estimation of the Number of Clusters," *Proc. of the 17th Int. Conf. on Machine Learning, 2000*, pp. 727–734.
34. F. Preparata and M. Shamos, *Computational Geometry* (Springer Verlag, 1985).
35. J. B. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations," *Proc. of the 5th Berkeley Symposium on Mathematics Statistics and Probability, 1967*, pp. 281–297.
36. V. Ramasubramanian and K. Paliwal, "Fast *k*-Dimensional Tree Algorithms for Nearest-Neighbor Search with Application to Vector Quantization Encoding," IEEE Trans. Signal Processing **40** (3), 518–531 (1992).
37. J. Rissanen, "A Universal Prior for Integers and Estimation by Minimum Description Length," Ann. Stat. **11** (2), 417–431 (1983).
38. E. H. Ruspini, "A New Approach to Clustering," Inf. Control **15**, 22–32 (1969).
39. J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications," Data Mining Knowledge Discovery **2** (2), 169–194 (1998).
40. J. Si, S. Lin, and M. A. Vuong, "Dynamic Topology Representing Networks," Neural Networks **13**, 617–627 (2000).
41. R. Sokal and C. D. Michener, "A Statistical Method for Evaluating Systematic Relationships," Univ. Kansas Sci. Bull. **38**, 1409–1438 (1958).
42. M. Sollenborn and P. Funk, "Category-Based Filtering and User Stereotype Cases to Reduce the Latency Problem in Recommender Systems," *Proc. of the 6th Europ. Conf. on Case Based Reasoning, ECCBR2002*, Springer Verlag Lecture Notes Series (2002), pp. 395–405.
43. D. K. Tasoulis, P. Alevizos, B. Boutsinas, and M. N. Vrahatis, "Parallel Unsupervised *k*-Windows: an Efficient Parallel Clustering Algorithm," Lecture Notes Comp. Sci. **2763**, 336–344 (2003).
44. D. K. Tasoulis and M. N. Vrahatis, "Unsupervised Distributed Clustering," *IASTED Int. Conf. on Parallel and Distributed Computing and Networks, 2004*, pp. 347–351.
45. D. K. Tasoulis and M. N. Vrahatis, "Unsupervised Clustering on Dynamic Databases," Pattern Recognit. Lett. **26**, 2116–2127 (2005).
46. D. K. Tasoulis and M. N. Vrahatis, "Generalizing the *k*-Windows Clustering Algorithm in Metric Spaces," Math. Comput. Modeling (2006) (in press).
47. M. N. Vrahatis, B. Boutsinas, P. Alevizos, and G. Pavlides, "The New *k*-Windows Algorithm for Improving the *k*-Means Clustering Algorithm," J. Complexity **18**, 375–391 (2002).
48. T. Villmann, R. Der, M. Hermann, and M. Martinetz, "Topology Preservation in Self-Organizing Feature Maps: Exact Definition and Measurement," IEEE Trans. Neural Networks **8**, 256–266 (1997).
49. L. Xu, "Bayesian Ying-Yang Machine, Clustering and Number of Clusters," Pattern Recognit. Lett. **18**, 1167–1178 (1997).
50. T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An Efficient Data Clustering Method for Very Large Databases," *Proc. of ACM SIGMOD Int. Conf. on Management of Data, 1996*, pp. 103–114.

Basilis Boutsinas. Received his diploma in Computer Engineering and Informatics in 1991 from the University of Patras, Greece. He also conducted studies in Electronics Engineering at the Technical Education Institute of Piraeus, Greece, and Pedagogics at the Pedagogical Academy of Lamia, Greece. He received his PhD on Knowledge Representation from the University of Patras in 1997. He has been an assistant professor in the Department of Business Administration at the University of Patras since 2001. His primary research interests include data mining, business intelligence, knowledge representation techniques, nonmonotonic reasoning, and parallel AI.

Dimitris K. Tasoulis received his diploma in Mathematics from the University of Patras, Greece, in 2000. He attained his MSc degree in 2004 from the postgraduate course "Mathematics of Computers and Decision Making" from which he was awarded a postgraduate fellowship. Currently, he is a PhD candidate in the same course. His research activities focus on data mining, clustering, neural networks, parallel algorithms, and evolutionary computation. He is coauthor of more than ten publications.

Michael N. Vrahatis is with the Department of Mathematics at the University of Patras, Greece. He received the diploma and PhD degree in Mathematics from the University of Patras in 1978 and 1982, respectively. He was a visiting research fellow at the Department of Mathematics, Cornell University (1987–1988) and a visiting professor to the INFN (Istituto Nazionale di Fisica Nucleare), Bologna, Italy (1992, 1994, and 1998); the Department of Computer Science, Katholieke Universiteit Leuven, Belgium (1999); the Department of Ocean Engineering, Design Laboratory, MIT, Cambridge, MA, USA (2000); and the Collaborative Research Center "Computational Intelligence" (SFB 531) at the Department of Computer Science, University of Dortmund, Germany (2001). He was a visiting researcher at CERN (European Organization of Nuclear Research), Geneva, Switzerland (1992) and at INRIA (Institut National de Recherche en Informatique et en Automatique), France (1998, 2003, and 2004). He is the author of more than 250 publications (more than 110 of which are published in international journals) in his research areas, including computational mathematics, optimization, neural networks, evolutionary algorithms, and artificial intelligence. His research publications have received more than 600 citations. He has been a principal investigator of several research grants from the European Union, the Hellenic Ministry of Education and Religious Affairs, and the Hellenic Ministry of Industry, Energy, and Technology. He is among the founders of the "University of Patras Artificial Intelligence Research Center" (UPAIRC), established in 1997, where currently he serves as director. He is the founder of the Computational Intelligence Laboratory (CI Lab), established in 2004 at the Department of Mathematics of University of Patras, where currently he serves as director.