11-2007

Incremental procedures for partitioning highly intermixed multi-class datasets into hyper-spherical and hyper-ellipsoidal clusters

Qinglu Kong
University of Nebraska at Omaha

Qiuming Zhu
University of Nebraska at Omaha, qzhu@unomaha.edu

Incremental procedures for partitioning highly intermixed multi-class datasets into hyper-spherical and hyper-ellipsoidal clusters

Qinglu Kong, Qiuming Zhu *

Department of Computer Science, University of Nebraska at Omaha, Omaha, NE 68182-0050, USA

Abstract

Two procedures for partitioning large collections of highly intermixed datasets of different classes into a number of hyper-spherical or hyper-ellipsoidal clusters are presented. The incremental procedures are to generate a minimum numbers of hyper-spherical or hyper-ellipsoidal clusters with each cluster containing a maximum number of data points of the same class. The procedures extend the move-to-front algorithms originally designed for construction of minimum sized enclosing balls or ellipsoids for dataset of a single class. The resulting clusters of the dataset can be used for data modeling, outlier detection, discrimination analysis, and knowledge discovery.

Keywords: Data models; Data clustering; Mini-max partition; Geometrical approximation; Knowledge discovery

1. Introduction

Cluster analysis has been an important technique for discovering the embedded patterns of data groups and identifying underlying distributions in the interested datasets [10,13]. An exact clustering process over a large dataset can take a prohibitive amount of time due to the computational complexity of the process. Efforts have been taken on finding methods for efficient and effective clustering on large datasets. Active themes of research include the applications of genetic programming methods, the spatial- and high-dimensional geometrical shape models, and the mixture of numerical and categorical primitives [23]. In this research, we pay attention to clustering analysis to identify the coherent patterns embedded in data sets in terms of certain geometric primitives in which the data distribution characteristics are modeled. As the size of datasets grow larger and the structures of dataset turn to be highly intermixed [6], the high-dimension geometrical representations emerge to be a proper way to abstract, model, and approximately describe the characteristics of the complex datasets.

The problem of using spherical and ellipsoidal primitives to describe dataset distributions in high dimensions has been a topic of study by Calafiore [4] and Welzl [24], and recently by Gartner [8,9]. Most of their

* Corresponding author. Tel.: +1 402 554 3685; fax: +1 402 554 3400.
E-mail address: qzhu@mail.unomaha.edu (Q. Zhu).

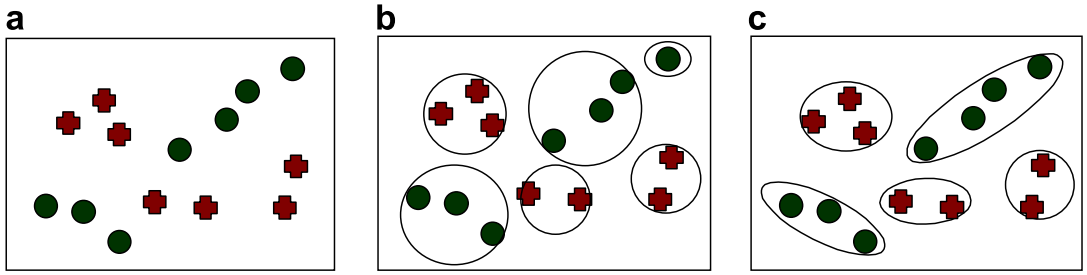

Fig. 1. (a) Intermixed dataset of two classes, (b) hyper-spheric partition of the dataset, and (c) hyper-ellipsoidal partition of the dataset.

work have been done on approximating the dataset of a uniform (single) class by a minimum sized enclosing hyper-ball [9,24] and hyper-ellipsoid [8,18], or fitting the data points by spherical and ellipsoidal surfaces [3]. It was mentioned in [3] that for the problem arising in pattern recognition it is often required to separating two or more sets of data points of different classes. However, no detail of the construction procedures for such applications was discussed.

The problem dealt with in this paper regards to a partition of n-dimensional datasets of highly intermixed data points of multiple classes. We model the dataset by a minimal number of non-intersecting hyper-spheres or hyper-ellipsoids, where each of the partitions contains a maximum number of data points of the same class. Such a model is intended to provide an approximation to the underlying distribution of the dataset. As an example of such approximation (Fig. 1) shows an intermixed dataset of two classes that cannot be partitioned by a single hyper-sphere or hyper-ellipsoid for each class without intersecting each other. However, using a set of hyper-spheres or hyper-ellipsoids it is possible to partition the dataset into non-intersecting clusters.

In the following of this paper, Section 2 reviews some existing geometrical methods for data clustering. Section 3 introduces the concept of the "move-to-front" algorithm for constructing the smallest enclosing ball/ellipsoid on a set of uniform data points. In Section 4, we describe two "move-to-front" based hyper-ball and hyper-ellipsoid generation procedures, named MFMHBC and MFMHEC, respectively, for clustering data points of multiple classes. Section 5 presents experimental results of applying the MFMHBC and MFMHEC procedures to datasets of various underlying distribution characteristics. Section 6 contains conclusion remarks.

2. Overview of geometrical data clustering

Given a dataset of n points (vectors) of p_j, $j = 1, \ldots, n$, a clustering process constructs k partitions C_i, $i = 1, \ldots, k$, $k \leqslant n$, of the point set such that the resulting clusters satisfy the criteria: (1) each partition contains data points of the same class, and (2) a certain measurement (criterion) defined on the partitions $E = \sum_{i=1}^{k} \sum_{p \in C_i} f(p, C_i)$ is minimized or maximized.

A clustering process usually starts with an initial partitioning. The process then applies an iterative allocation/reallocation technique to improve the partitions by moving data points among the clusters. Clustering with the squared-error criterion $\sum_{i=1}^{k} \sum_{p \in C_i} \|p - m_i\|^2$, i.e. $f(p,\ C_i) = \|p - m_i\|^2$, where m_i is the center of the partition C_i, is the most common in practice. With this criterion, the total Euclidean distance of data points to their partition center is smaller than those between the partitions. That is, the process tries to make the k clusters as compact and separated as possible. The squared-error minimization criterion based partition process leads naturally to geometric entities in the forms of circles in two dimension [21], spheres (balls) in three dimension [9], and hyper-spheres (hyper-sphere) in multi-dimension. The criterion works well when clusters are compact clouds that are well separated from one to another. However, the method is often inadequate to model datasets of arbitrarily or irregularly shaped clusters. Moreover, to achieve global optimality, this clustering process requires an exhaustive enumeration of all possible partitions [10]. In practice, most applications adopt some heuristic methods for the efficiency consideration that leads to sub-optimal solutions.

To overcome the limitations of the spherical cluster modeling, several non-spherical clustering techniques have been studied. A grid-based clustering approach uses a multi-resolution grid structure of geometry. It quantizes the data space into a finite number of cells on which all of the operations for clustering are performed [12]. A typical example of grid-based methods is named as statistical information grid (STING). STING is a multi-resolution clustering technique in which the spatial area of data points is divided into rect-

angular cells. There are usually several levels of such rectangular cells corresponding to different levels of resolution, and these cells form a hierarchical structure. Each cell at a high level is partitioned to form a number of cells at the next lower level. Statistical information regarding the attributes in each grid cell (such as the mean, maximum, and minimum values) is pre-computed and stored.

Another non-spherical modeling technique is called density-based clustering [6]. The density-based spatial clustering of applications with noise (DBSCAN) algorithm grows the geometrical regions with sufficiently high density and discovers clusters of arbitrary shape. It defines a cluster as a maximal set of density-connected points. The ordering points to identify the clustering structure (OPTICS) algorithm computes an ordering of augmented clusters rather than produces a data set of clusters explicitly. The ordering represents the den-sity-based clustering structure of the data. The clustering based on density distribution functions, or density-based clustering (DENCLUE) algorithm is built on the ideas that (1) the influence of each data point can be formally modeled using a mathematical function, called an influence function, that describes the impact of a data point within its neighborhood; (2) the overall density of the data space can be modeled analytically as the sum of the influence functions of all data points; and (3) clusters can then be determined mathematically by identifying density attractors, which are local maxima of the overall density function. Based on these defini-tions, both centerly-defined cluster and arbitrarily-shaped cluster can be formally defined.

However, these non-spherical clustering algorithms also have their limitations. The DBSCAN algorithm clusters objects with respect to a given set of input parameters specifying the minimum number of points in each cluster and the number of neighborhood points to be searched. Thus it leaves the user with a responsi-bility of selecting proper parameter values so as to lead a discovery of acceptable clusters. Actually, this is a problem associated with a number of other partitioning based methods. The parameter settings are usually empirical and difficult to determine, especially for real-world, high-dimensional data sets. Most algorithms of this kind are very sensitive to such parameter values. Slightly different setting may lead to very different clustering results. Moreover, high-dimensional datasets often have very skewed distributions such that their intrinsic clustering structures may not be characterized by global density parameters. The OPTICS algorithm overcomes this difficulty by computing an ordering of augmented clusters for automatic and interactive cluster analysis. DENCLUE has good clustering properties for data sets with large amounts of noise, and allows a compact mathematical description of arbitrarily shaped clusters in high-dimensional data sets. However, DENCLUE also requires careful selection of the density parameters and noise threshold. The grid structure of the STING method facilitates parallel processing and incremental updating. However, the quality of STING clustering depends on the granularity of the lowest level of the grid structure. If the granularity is very fine, the cost of processing will increase substantially. If the bottom level of the grid structure is too coarse, it may reduce the quality of clustering result. Moreover, the shapes of the resulting clusters are isometric; that is, all of the cluster boundaries are either horizontal or vertical, and no diagonal boundary is detected. This may lower the quality and accuracy of the clusters despite the faster processing.

Besides the above partition based clustering techniques, a number of other clustering methods were developed that involve the use of some other geometric entities to partition a dataset [2,18]. A model-based method hypoth-esizes a curve or surface function for each of the clusters and attempts to optimize the best fit of the data to the given function [17]. A model-based algorithm may also locate clusters by constructing a density function that reflects the spatial distribution of the data points, and is often based on the assumption that the data are gener-ated by a mixture of underlying probability distribution. It leads to a way of automatically determining the num-ber of clusters based on standard statistics, taking "noise" or outliers into account, and yielding robust clustering methods. However, the model-based clustering methods have a number of limitations too. First, the assumption that probability distributions on separate attributes are statistically independent is not always true since corre-lation between attributes often exists. Second, the probability distribution is expensive to update since their time and space complexities depend on both the number of attributes and the values of each attribute.

The conceptual clustering approach, a form of clustering technique developed in machine learning research, produces a classification scheme over the data points that may or may not fit a geometrical model [16]. Unlike previous clustering methods, which primarily identify groups of similar objects, conceptual clustering process goes one step further by also finding characteristic descriptions for each group, where each group represents a concept or a class. It often creates a hierarchical clustering structure in the form of a classification tree, and uses some heuristic evaluation measures to guide the construction of this tree. However, the classification tree

may not be height-balanced for skewed input data, which may cause the time and space complexity of the computation to degrade dramatically.

Hierarchical clustering is a sequential process in which data points are grouped in a tree structure called dendrogram. Each data group is nested into the next group in the sequence, and creates a hierarchical decomposition of the given set. A hierarchical method can be classified as being either agglomerative or divisive, based on how the hierarchical decomposition is formed [11,13]. Divisive hierarchical clustering does the reverse of agglomerative hierarchical clustering by starting with all objects in one cluster. It subdivides the cluster into smaller and smaller pieces, until each object forms a cluster on its own or until it satisfies certain termination conditions, under which the desired number of clusters is obtained or the distance between the two closest clusters is above a certain threshold. The hierarchical clustering methods, though simple, often encounter difficulties with respect to the selection of merging or spliting points. Such a decision is critical because once a group of objects are merged or split, the process at the next step will operate on the newly generated clusters. Moreover, the method does not scale well since the decision of merge or split needs to examine and evaluate a large number of objects or clusters. In order to improve the clustering quality of hierarchical methods, people often integrate it with other techniques to create a multiple phased clustering process.

One important distinction between partitioning and hierarchical clustering approaches is that the hierarchical methods produce a nested series of geometric entities whereas partitioning methods produce a spatially flat configuration of entities. Existing literature [17] shows that partitioning methods have advantages over the other clustering methods, including the model-based, conceptual, and hierarchical methods, in applications involving large data sets.

In this paper, we present a partitioning based geometrical clustering approach for modeling dataset with respect to their coherent categorical characteristics. We concentrate on the development of incremental construction algorithms for generating multiple data clusters in hyper-spherical and hyper-ellipsoidal shapes (i.e., hyper-balls and hyper-ellipsoids) to model approximately the highly intertwining dataset of multiple classes.

3. Minimum enclosing hyper-balls and hyper-ellipsoids constructions

The construction of smallest enclosing hyper-ball or hyper-ellipsoid over a given data set is a classical problem in computational geometry [1]. The history can be dated back to 150 years ago when Sylvester formulated it for points in a plane [22]. Megiddo provided the first optimal linear-time algorithm for fixed dimensions [15]. In recent years, a number of randomized algorithms have been developed. These algorithms are attractive not only because of their efficiency, but also because of their appealing simplicity. One of these algorithms is Seidel's [19]. Given a set S of n points p_i in a d-dimensional space, Seidel's algorithm models the minimum enclosing ball as a solution to a linear programming (LP) problem with n constraints and d variables. The algorithm has an expected time complexity of $O(n)$, provided d is constant. Clarkson described a randomized LP algorithm where the expectation is averaged over random choices ('coin flips') made by the algorithm [5]. In particular, there is no input that may force the algorithm to perform poorly. In 1991, Welzl developed a simple randomized method to solve the problem in expected linear time complexity [24]. In contrast to Megiddo's method, Welzl's algorithm is easy to implement and very fast in practice for dimensions two and three. In higher dimensions, a heuristic move-to-front variant considerably improves the performance.

The smallest enclosing hyper-ball/hyper-ellipsoid problem of an n-point set in d-space is formally defined as follows:

Definition 3.1. Given a set $P = \{p_1, \ldots, p_n\}$ of points in R^d, the smallest enclosing hyper-ball $MB(P)$ of the points P is the hyper-ball $B \subset R^d$ of minimal radius $r_B \in R^d$ which encloses all points in P.

Definition 3.2. Given a point $c \in R^d$ and a symmetric, positive definite matrix $A \in R^{d*d}$, the set of points $p \in R^d$ satisfying $(p-c)^T A(p-c) = 1$ defines an hyper-ellipsoid with center c. The function $f(p) = (p-c)^T A(p-c)$ is called the *hyper-ellipsoid function*, the set $E = \{p \in R^d | f(p) \leqslant 1\}$ is the *hyper-ellipsoid body*. Given a point set $P = \{p_1, \ldots, p_n\}$ of points in R^d, we are interested in the hyper-ellipsoid body of the smallest volume containing P. Identifying the body with its generating hyper-ellipsoid, we call this *the smallest enclosing hyper-ellipsoid* of P, $ME(P)$.

Definition 3.3. Given two disjoint sets S and R of data points, $MBV(S,R)$ or $MEV(S,R)$ denotes the set of hyper-balls or hyper-ellipsoids of minimal volumes which enclose all points in the set $S \cup R$ and have at least one of the points in the R on their boundary. In particular, a hyper-ball $B \in MBV(\phi, R)$ or a hyper-ellipsoid $E \in MEV(\phi, R)$ is called a V-mini-ball or V-mini-ellipsoid.

Hereafter in this section, we will simply use the terms "ball" and ellipsoid" to stand for the hyper-ball and hyper-ellipsoid. For the existence and uniqueness of the mini-balls Welzl used the following construction: Given an n-point set $P = \{p_1, \ldots, p_n\} \subseteq R^d$, let $MB(P)$ denote the ball of the smallest radius that contains P, obviously $MB(P)$ exists and is unique. For $P, B \subseteq R^d$, $P \cap B = \phi$, let $MB(P,B)$ be the smallest ball that contains P and has all points of B on its boundary, we have $MB(P) = MB(P, \phi)$, and if $MB(P,B)$ exists, it is unique. Finally, define $\overline{MB}(B) := MB(\phi, B)$ to be the smallest ball with all points of B on the boundary (if it exists). A support set of (P,B) is an inclusion-minimal subset S of P such that $MB(P,B) = MB(S,B)$. If the points in B are affinely independent, there always exists a support set of size at most $(d+1)$, where d is the dimension [24].

The idea behind the Welzl's algorithm was, roughly speaking, to find a minimal subset $S \subset P$ of the original balls B with the property $\overline{MB}(S) = \overline{MB}(P)$. Then $\overline{MB}(P)$ coincides with $\overline{MB}(\phi, B)$, which is much simpler to solve. Similarly, for the smallest enclosing ellipsoids, a support set of size at most $(d+3)d/2$ should exist [20].

The Welzl's "move-to-front" heuristic algorithm computes the smallest enclosing ball $MB(P,B)$ incrementally by adding one point after another. The practical efficiency comes from the fact that 'important' points (which for the purpose of the method are points outside the current ball) are moved to the front and will therefore be processed early in subsequent recursive calls [9]. The method keeps the points in an ordered list L, which gets reorganized as the algorithm runs. Let L_i denote the length-i prefix of the list, and p_i the element at position i in L. Initially, $L = L_n$ stores the points of P in random order. The algorithm is listed below.

Algorithm 3.1. Welzl's move-to-front method to compute the smallest enclosing ball

```
mtf_mb (Ln, B):
  mb:= MB(B)
  IF |B| = d + 1 THEN
    RETURN mb
  END
  FOR I = 1 TO n DO
    IF P_i ∉ mb THEN
      Mb := mtf_mb(L_{i-1}, B∪{p_i})
      Update L by moving p_i to the front
    END
  NEXT
  RETURN mb
```

During the processing of each point $p_i \in P$, all non-trivial computations take place in the primitive operation of computing $\overline{MB}(B)$ for a given set B in the calls to *mtf_mb*, which stands for "move-to-front_mini-ball". The algorithm guarantees that B is always a set of affinely independent points, from which $|B| \leqslant d+1$ follows. In that case, $\overline{MB}(B)$ is determined by the unique circum-sphere of the points in B with center restricted to the affine hull of B. This means, the center c and squared radius r^2 satisfy the following system of equations, where $B = \{q_0, \ldots, q_{m-1}\}$, $m \leqslant d+1$:

$$(q_i - c)^{\mathrm{T}}(q_i - c) = r^2, \quad i = 0, \ldots, m-1,$$

$$\sum_{i=0}^{m-1} \lambda_i q_i = c,$$
$$\sum_{i=0}^{m-1} \lambda_i = 1.$$

This formulation of the problem sounds like an instance of the abstract framework of so-called linear programming (LP)-type problems [14]. Indeed, the problem of finding the mini-ball of a given point set turns out to be exactly an LP-type problem. Defining $Q_i := q_i - q_0$, for $i = 0, \ldots, m-1$ and $C := c - q_0$, the system can be rewritten as

$$C^{\mathrm{T}} C = r^2,$$
$$(Q_i - C)^{\mathrm{T}} (Q_i - C) = r^2, \quad i = 0, \ldots, m-1,$$
$$\sum_{i=0}^{m-1} \lambda_i Q_i = C.$$

Substituting C with $\sum_{i=1}^{m-1} \lambda_i Q_i$ in the equations, we deduce a linear system in the variables $\lambda_1, \ldots, \lambda_{m-1}$ which we can write as

$$A_B \begin{pmatrix} \lambda_1 \\ . \\ . \\ . \\ \lambda_{m-1} \end{pmatrix} = \begin{pmatrix} Q_1^{\mathrm{T}} Q_1 \\ . \\ . \\ . \\ Q_{m-1}^{\mathrm{T}} Q_{m-1} \end{pmatrix},$$

where

$$A_B := \begin{pmatrix} 2Q_1^T Q_1 \cdots 2Q_1^T Q_{m-1} \\ . \\ . \\ . \\ 2Q_1^T Q_{m-1} \cdots 2Q_{m-1}^T Q_{m-1} \end{pmatrix}.$$

Computing the values of $\lambda_1, \ldots, \lambda_{m-1}$ amounts to solve the linear system, C and r^2 are then obtain via

$$C = \sum_{i=0}^{m-1} \lambda_i Q_i, \quad r^2 = C^{\mathrm{T}} C.$$

C is referred as the center of the set B.

The principle of Welzl's algorithm for computing the smallest ellipsoid $\overline{ME}(P)$ is similar to computing $\overline{MB}(P)$. It works in the following way. If P is empty, $\overline{ME}(P)$ is the empty set by definition. If not, choose a point $q \in P$ and recursively determine $E := \overline{ME}(P \setminus \{q\})$. If $q \in E$, then $E = \overline{ME}(P)$ and we are done. Otherwise, q must lie on the boundary of $\overline{ME}(P)$, and we get $\overline{ME}(P) = \overline{ME}(P \setminus \{q\}, \{q\})$, the smallest enclosing ellipsoid of $P \setminus \{q\}$ with q on the boundary. The generic call of the algorithm computes $\overline{ME}(Q, R)$, the smallest ellipsoid enclosing Q that has R on the boundary [8,19].

4. Incremental construction of hyper-spheres and hyper-ellipsoids for multi-class data clustering

Based on the preceding discussion, we introduce two procedures for incremental construction of hyper-spheres and hyper-ellipsoids for datasets of multiple classes. The main difference of our procedures from the Welzl's is that our algorithms work for datasets that contain multi-classes of data points and these data points are highly intermixed. The results of our hyper-spherical and hyper-ellipsoidal data models thus are applicable for multi-class data classifications.

4.1. Move-to-front multi-hyper-ball-clustering algorithm (MFMHBC)

To give a formal description of MFMHBC, we first define the following notations:

t – the number of different classes in a given data set.
n_i – the number of data points in class i, $i \in [1, t]$.
N – the total number of data points in a given data set (i.e. $N = \sum_{i=1}^{t} n_i$).
c – the center of a hyper-ball in D-dimension space.
r – the radius of the hyper-ball in D-dimension space.
B – the set of hyper-balls.
B_i – the set of hyper-balls for class i, $i \in [1, t]$.
b – a hyper-ball in the set of hyper-balls B.
b_{temp} – a temporary hyper-ball in D-dimension space.
$\|B\|$ – the number of hyper-balls in B.
$\|B_i\|$ – the number of hyper-ball in B_i, $i \in [1, t]$.
P_i – a subset of data set P, which contains the data points in class i, $i \in [1, t]$.
p_i – a vector in D-dimensional space, $p_i = [p_{i1}, p_{i2}, \ldots, p_{ij}, \ldots, p_{iD}]$, p_{ij} is the jth value in vector p_i.

Algorithm 4.1. Move-to-front multi-hyper-ball-clustering algorithm (MFMHBC)

Input: t, n_i $(i = 1, 2, \ldots, t)$, P_i $(i = 1, 2, \ldots, t)$.
Output: $\{B\}$.
For each class i $(i = 1, 2, \ldots, t)$ *do*
  For each example $p_{ij} \in P_i$ $(1 \leqslant j \leqslant D)$ *do*
//step One.
  For each $b \in B_i$ *do*
    If $(|p_{ij} - c| \leqslant r)$ *then* done.
    Else Find a ball $b \in B_i$ such that $|p_{ij} - c|$ is minimal.
  End for;
//step Two.
  If p_{ij} is not included in any existing hyper-ball in B_i *then*
  $\{b_{temp} \leftarrow b \cup p_{ij}$;
//step Three.
  If NOT((b_{temp} intersect with any $b' \in B$, $b' \neq b$)
  AND ($p \subset b_{temp}$, $p \in P_{i'}$, $i' \neq i$)) *then* $b \leftarrow b_{temp}$;
  Else $\{b_{temp} \leftarrow \{p_{ij}\}$; $B \leftarrow B \cup b_{temp}$;$\}$
  }
  End for;
End for;
Return $\{B\}$

The algorithm takes care of every data point in the set in a move-forward manner. For every data point in step one, we test it against every existing hyper-ball of the same class. If this data point is included by one of existing hyper-balls in the same class, in other words, if the Euclidean distance between the data point and the center of the hyper-ball is less than or equal to the radius of the hyper-ball, we are done with this point. Otherwise, we need to find a hyper-ball such that the Euclidean distance between the data point p_{ij} and the center of the hyper-ball c is minimal among the distances between the data point and the center of any other hyper-ball in the same class. In the step two, we form a new minimized hyper-ball, which is called a temporary hyper-ball covering the data point and the hyper-ball found in step one. In the step three, we have to check whether this temporary hyper-ball is intersected with any other existing hyper-ball, and whether data points in other classes are included in this temporary hyper-ball. If this hyper-ball can meet above criterion, we replace

the previous hyper-ball with this newly formed hyper-ball; otherwise, a new hyper-ball is produced, with the center being at the position of this data point and its radius being zero. After finishing all the iterations, we are provided with a whole set of hyper-balls, and the number of clusters as well.

The MFMHBC algorithm has following properties:

1. Upon the algorithm termination, there is no intersection between any two hyper-balls of different data classes (i.e. $b_1 \cap b_2 = \Phi$, while $b_1 \in B_i$, $b_2 \in B_j$, $i \neq j$).
2. After the algorithm terminates, the hyper-ball set B is the minimum set with each hyper-ball b covering the maximum number of data points.
3. The number of clusters resulting from the algorithm equals to the number of hyper-balls generated, without the need to specify the number in advance.

The time complexity of the MFMHBC can be derived as the following. The Step One to check whether the given data point is inside one of the existing hyper-balls of the same class has a running time $O(C_1N)$, where C_1 is a constant. In the Step Two, we compute the new center and radius if the test point is not included in any existing hyper-ball of the same class. In case the number of points is less than the dimension plus one, we choose the mean of these objects as the center, find the farthest point to the center, and calculate the radius; otherwise, we need to solve a linear system to find the smallest enclosing ball. The running time to solve this system is directly related to dimension, it is $O(D^3)$ for each data point, so the total running time is $O(ND^3)$. In the step three, we need to check whether the newly formed hyper-ball intersects with other hyper-balls of different classes, and whether data points of another class are included in this hyper-ball as well, in each iteration. The running time is $O(ND)$ for each data point, and the total time thus is $O(N^2D)$. In conclusion, the total running time of this algorithm is $O(C_1N + ND^3 + N^2D)$. Because the number of data points N grows much faster than the number of dimension D in most applications, the worst running time of this algorithm is $O(N^2D)$.

We then present the move-to-front multi-hyper-ellipsoid procedure for multi-class data clustering.

4.2. Move-to-front multi-hyper-ellipsoid-clustering algorithm (MFMHEC)

Like the MFMHBC above, we use the following notations for the MFMHEC algorithm.

t – the number of different classes in a given data set.
n_i – the number of data points in class i, $i \in [1, t]$.
N – the total number of data points in a given data set (i.e. $N = \sum_{i=1}^{t} n_i$).
c – the center of a hyper-ellipsoid in D-dimension space.
r_k – the kth principle axis of the hyper-ellipsoid in D-dimension space, $k \in [1, D]$.
E – the set of hyper-ellipsoids.
E_i – the set of hyper-ellipsoids of class i, $i \in [1, t]$.
e – a hyper-ellipsoid in the set of hyper-ellipsoids E.
e_{temp} – a temporary hyper-ellipsoid in D-dimension space.
$\|E\|$ – the number of hyper-ellipsoids in E.
$\|E_i\|$ – the number of hyper-ellipsoids in E_i, $i \in [1, t]$.
P_i – a subset of data set P, which contains the data points in class i, $i \in [1, t]$.
p_i – a vector in D-dimensional space, $p_i = [p_{i1}, p_{i2}, \ldots, p_{ij}, \ldots, p_{iD}]$, p_{ij} is the jth value in vector p_i.

The following functions are called in the algorithm.

1. *Preprocessing* (P_i) is a function to (1) calculate the mean point of the data point in the same class set i, (2) find a data point such that the distance between the mean point and the data point is minimal among the distances between the mean point and any of other data points in the same class set, and (3) find two data points such that the distance between the mean point and each of these two data points is smaller than that between theses two data points, but bigger than that between the mean point and any of other data points in the same class set.

2. *IsInsideEllipsoid* (p_i, e) is a function to test whether a vector p_i is inside a hyper-ellipsoid e; it returns a Boolean value. If it is true, the vector is inside the hyper-ellipsoid e; if it is false, the vector is outside the hyper-ellipsoid e.
3. *Merge_Ellipsoids* (E) is a function to merge the hyper-ellipsoids with data points of the same class to keep the number of hyper-ellipsoids to a minimum under the conditions that the resulting hyper-ellipsoid will not intersect with other hyper-ellipsoids of data points of different classes.

Algorithm 4.2. Move-to-front multi-hyper-ellipsoid clustering algorithm (MFMHEC)7

Input: t, n_i $(i = 1,2,\ldots,t)$, P_i $(i = 1,2,\ldots,t)$.
Output: $\{E\}$.
For each class i $(i = 1,2,\ldots,t)$ *do*
    $E_i \leftarrow Preprocessing(P_i)$;
    For each example $p_{ij} \in P_i$ $(1 \leqslant j \leqslant D)$ AND (p_{ij} *not processed*) *do*
//step One.
    For each $e \in E_i$ *do*
        If (*IsInsideEllipsoid*(p_{ij}, e)) *then* done.
        Else Find an ellipsoid $e \in E_i$ such that $\|p_{ij} - c\|$ is minimal.
    End for;
//step Two.
    If p_{ij} is not included any existing hyper-ellipsoid in *Ei then*
    $\{e_{temp} \leftarrow e \cup p_{ij}$;
//step Three.
    If NOT((e_{temp} intersect with any $e' \in E$, $e' \neq e$)
        AND ($p \subset e_{temp}$, $p \in P'_i$, $i' \neq i$)) *then* $e \leftarrow e_{temp}$;
    Else
//step four
    {settle_flag = false;
        For every ($e'' \in E_i$) AND ($e'' \neq e$) *do begin*
            $e_{temp} \leftarrow e'' \cup p_{ij}$;
            If NOT((e_{temp} intersect with any $e' \in E$, $e' \neq e''$)
                AND ($p \subset e_{temp}$, $p \in P_{i'}$, $i' \neq i$)) *then*
            {e e_{temp}; settle_flag = true; break;}
        End for;
        If settle_flag = = false *then* {e_{temp} {p_{ij}}; E $E \cup e_{temp}$;}
    }

    Mark p_{ij} *be processed*;
    End for;
End for;
Merge_Ellipsoids(E);
Return $\{E\}$

Like the MFMHBC algorithm, the MFMHEC algorithm goes through data points of each class in the datasets one by one. For each class set, three points are selected to form an initial hyper-ellipsoid. For each of the rest data points in the set, we test it against the existing hyper-ellipsoid in step one. If this data point is included in one of existing hyper-ellipsoids of the same class, tested by using function *IsInsideEllipsoid* (p_{ij}, e), we are done with this data point. Otherwise, we need to find a hyper-ellipsoid such that the distance between the data point p_{ij} and the center of the hyper-ellipsoid c is minimal among the distances between the data point and the center of any other hyper-ellipsoid in the same class. In the step two, we form a new hyper-ellipsoid, which is called a temporary hyper-ellipsoid covering the data point and the hyper-ellipsoid found in step one.

In the step three, we have to check whether this temporary hyper-ellipsoid intersects with any other existing hyper-ellipsoid, and whether data points in another class are included in this hyper-ellipsoid. If the hyper-ellipsoid passes above check, we update the previous hyper-ellipsoid with this newly formed hyper-ellipsoid; otherwise, we go to the step four. In the step four, we try to join the data point with the other hyper-ellipsoids of the same class, and check the results against the non-intersection criteria. If such a hyper-ellipsoid can be formed, we just update it; otherwise, a new hyper-ellipsoid is produced with the center being at the position of this data point, and with all its principle axes being equal to zero. After iterating all the data points in the set, we get a set of hyper-ellipsoids that cover all the data points. A post-processing step tries to merge the hyper-ellipsoids with data points of the same class to maintain a minimum number of hyper-ellipsoids.

The MFMHEC algorithm has the properties similar to those of MFMHBC. An extra pre-processing step is included in MFMHEC for producing evenly distributed hyper-ellipsoids in the resulting set. In order to maintain the min–max feature at the termination of the algorithm, the Merge-Ellipsoids function is utilized to serve as the post-processing step.

The time complexity analysis of MFMHEC is as follows. The preprocessing step takes $O(n_i)$ time to calculate the mean point for each data set S_i that contains data points of the same class. It then takes $O(n_i)$ time to find three data points to initiate the ellipsoid set E_i. So, the total running time is $O(\sum_{i=1}^{k} n_i) = O(N)$. In the Step One, the check of whether the current data point is inside one of the existing hyper-ellipsoids of the same class takes a running time $O(C_1 N)$. In the Step Two, the computation of the new center and a set of principle axes takes $O(n_i^3 D + D^3)$ for each hyper-ellipsoid, therefore the total time complexity is $O(t((N/t)^3 D + D^3)) = O(N^3 D/t^2 + tD^3)$. In the step three, we check whether the newly formed hyper-ellipsoid intersects with other hyper-ellipsoids of different classes, and whether data points of other classes are included in this hyper-ellipsoid. The running time is $O(ND)$ for every data point, and the total time is $O(N^2 D)$. In step four, we basically repeat step two and step three, so, the time complexity is $O(\|Ei\|(N^3 D/t^2 + tD^3 + N^2 D))$. Since $\|E_i\|$ is relatively small compared to the total number of data points in the given set, the worst case requires a running time of $O(C_2(N^3 D/t^2 + tD^3 + N^2 D))$. In the post-processing step, we perform merging and checking

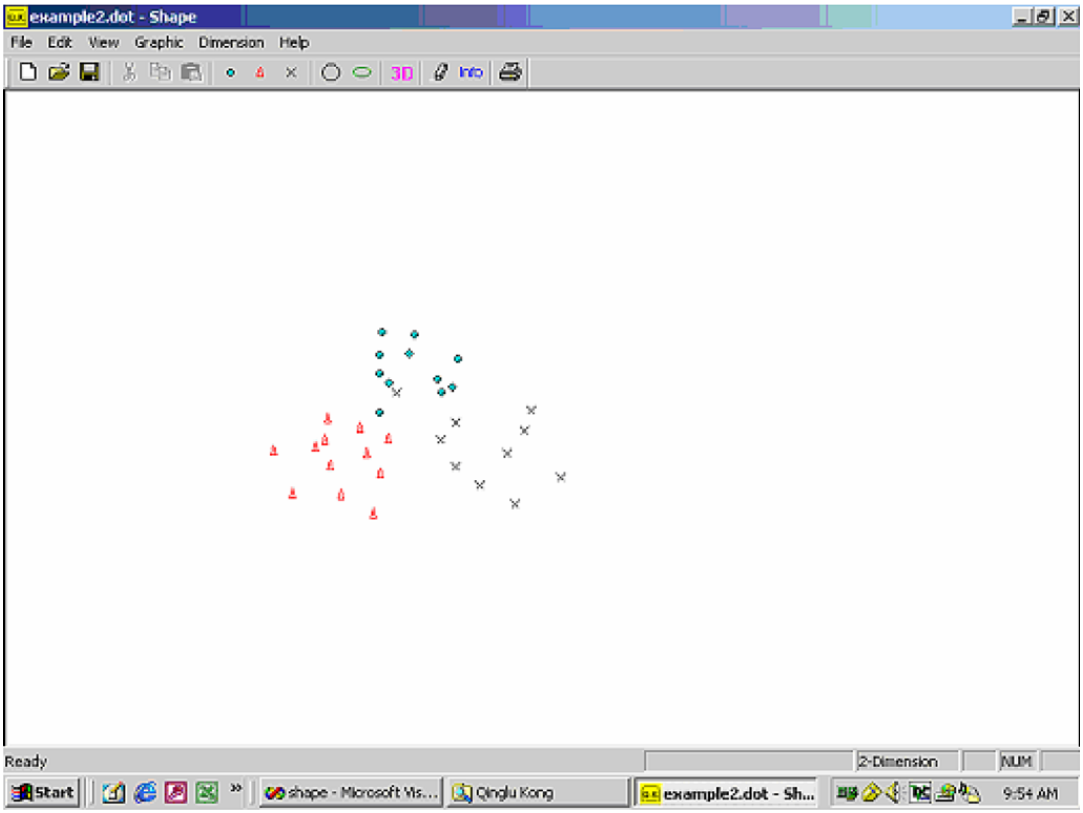

Fig. 2. Data point set of case one.

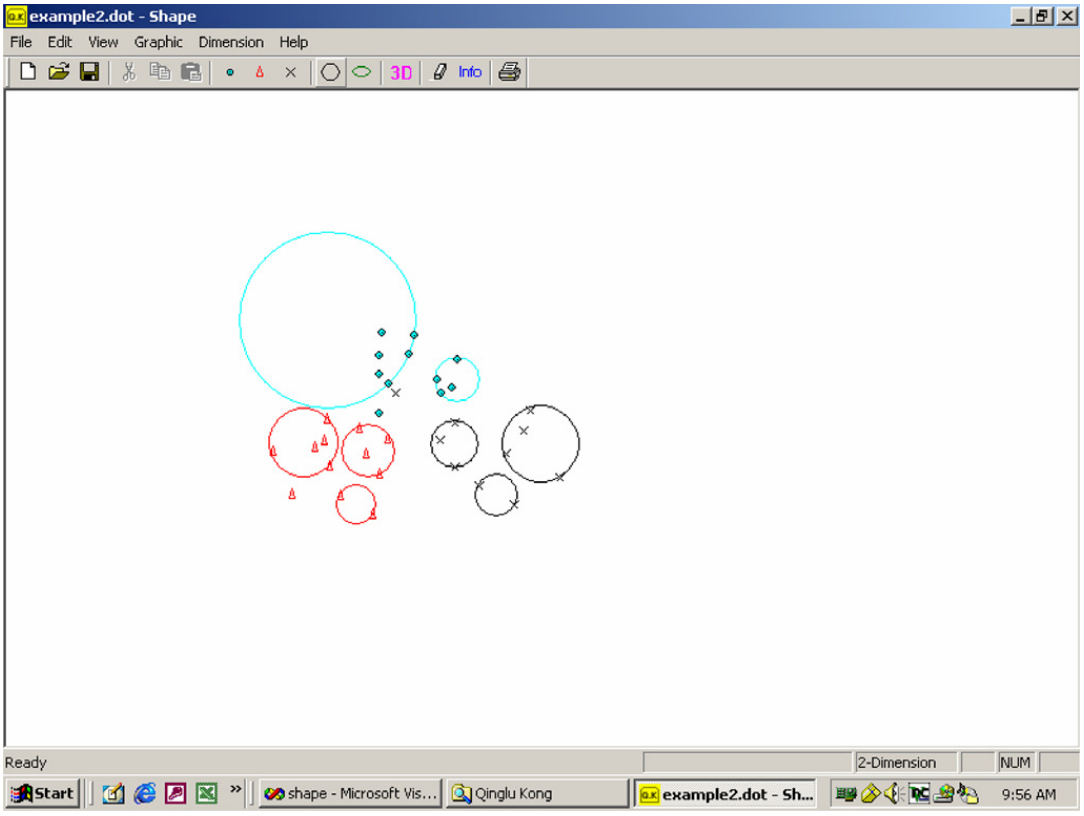

Fig. 3. Clustering result of MFMHBC.

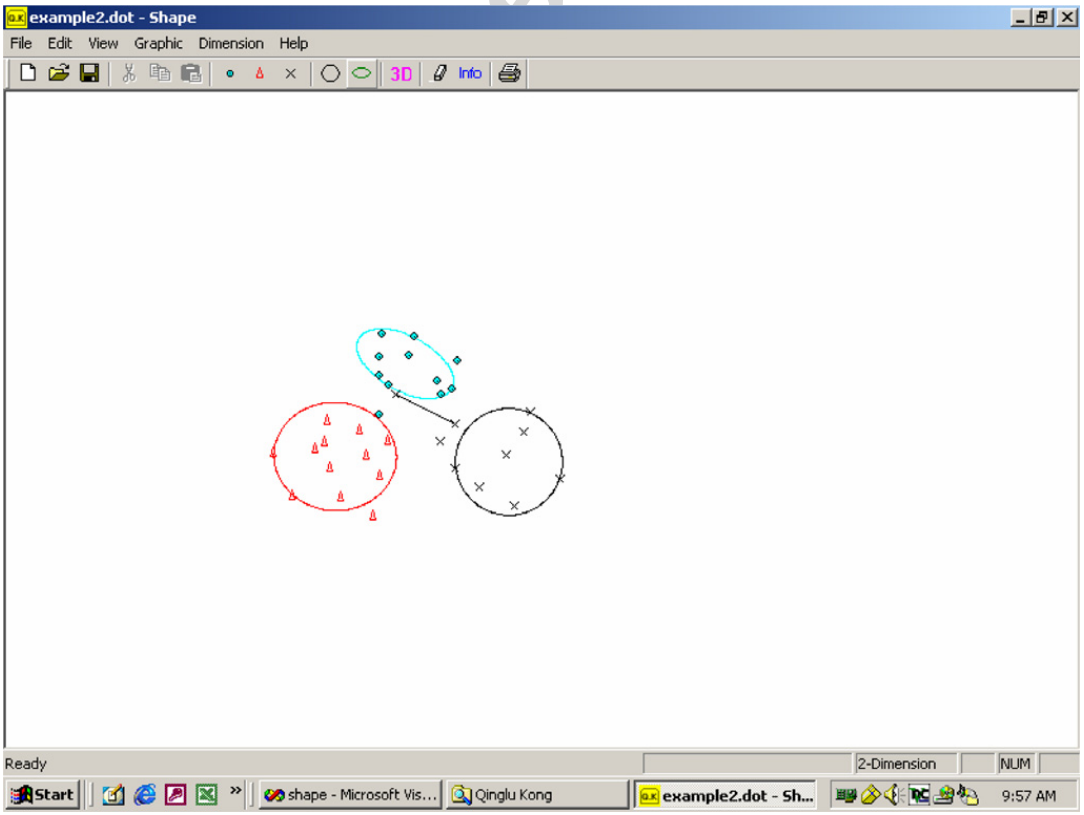

Fig. 4. Clustering result of MFMHEC.

for each hyper-ellipsoid. So, the time complexity is $O(\|E_i\|^2)$. In conclusion, the total running time of MFMHEC is $O(N + C_1N + N^3D/t^2 + tD^3 + C_2(N^3D/t^2 + tD^3 + N^2D) + \|E_i\|^2) = O(N^3D/t^2 + tD^3)$. Because the number of data points grows much faster than the number of dimensions in a data mining context, the worst running time of this algorithm is dominated by $O(N^3D/t^2)$.

5. Experiment results

Since the scalability is one critical problem to most geometrical computation algorithms, the MFMHBC and MFMHEC algorithms are tested with experimental datasets of two, three, and multiple (up to 20) dimensions with the number of classes in the datasets ranging from 2 to 20. In the graphical display of data sets in the subsequent case studies of this section, we use "○" to represent the data point of the first class, "△" the data point of the second class, and "×" the data point of the third class. Data sets and results with higher dimensions are only shown in summarization tables without graphical display.

5.1. Case one moderately intermixed data set of two classes in two-dimensional space

In this example, the data points in each class are in Gaussian distributions. In order to achieve the goal of no intersection between any two hyper-balls or hyper-ellipsoids of different classes, both MFMHBC and

Table 1
Clustering results on moderately intermixed data set of two classes in two-dimension

Type of class	# of Data points	# of Clusters with MFMHBC	# of Clusters with MFMHEC
A	11	3	3
B	12	4	2
C	10	4	3
Total	33	11	8

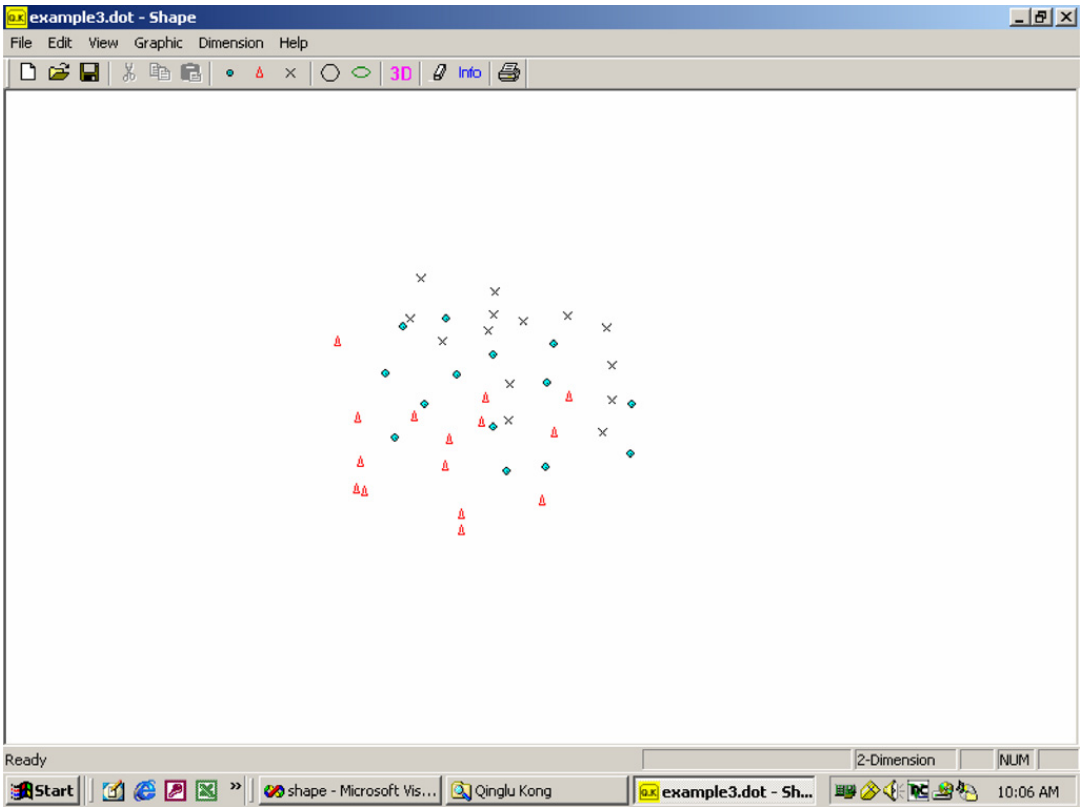

Fig. 5. Data point set of case two.

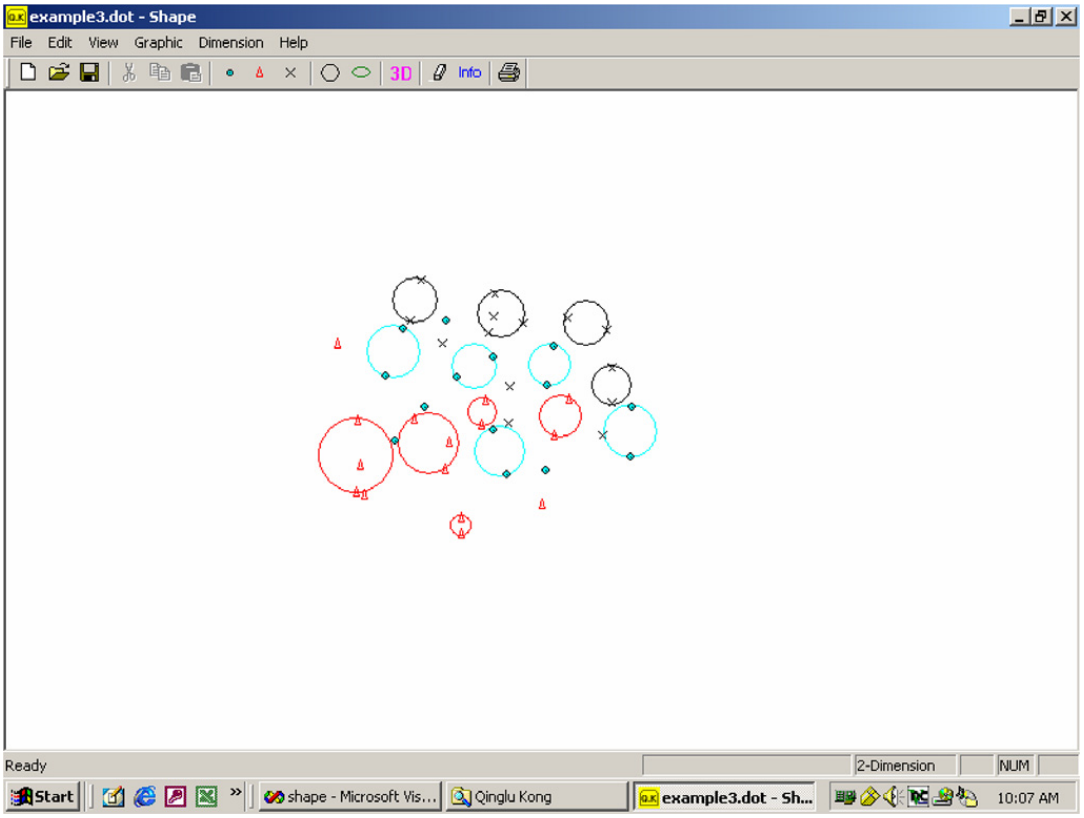

Fig. 6. Clustering result of MFMHBC.

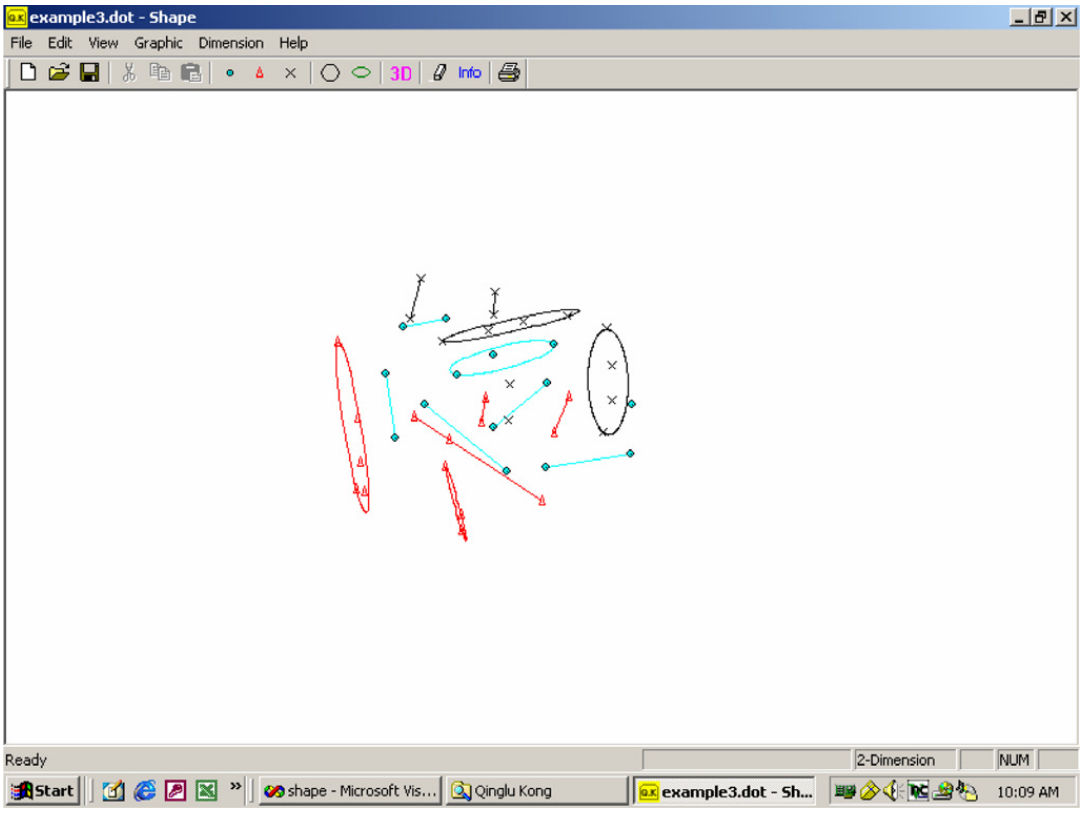

Fig. 7. Clustering result of MFMHEC.

MFMHEC use more than one hyper-balls or hyper-ellipsoids to cluster the data points of each class. As expected, the MFMHEC generates a smaller number of hyper-ellipsoids than the hyper-balls generated by the MFMHBC. The data distribution and the clustering results are shown in Figs. 2–4, respectively. In these figures some of the points seem not in any cluster because these are clusters with only a single data point in each of them (the radius of the hyper-ball or hyper-ellipsoid equals to zero). Table 1 summarizes the clustering results in terms of the number of clusters generated by each algorithm. Generally speaking, the smaller number of clusters means a better approximation of the underlying distribution of the dataset because the original data were generated in single Gaussian distribution for each class.

5.2. Case two – three highly intermixed classes in two-dimensional space

Fig. 5 is an example with data points in three classes, with each class in a single Gaussian distribution. However, the data points of different classes are highly mixed up in this example. The clustering results using MFMHBC and MFMHEC are presented in Figs. 6 and 7, respectively.

The clustering results are summarized in Table 2. In this example, we can see that MFMHEC is again more effective than MFMHBC in terms of the (smaller) number of the clusters needed to partition the dataset.

Table 2
Clustering results on data set of three highly intermixed classes in two-dimension

Type of class	# of Data points	# of Clusters with MFMHBC	# of Clusters with MFMHEC
A	14	9	7
B	15	7	5
C	14	8	6
Total	43	24	18

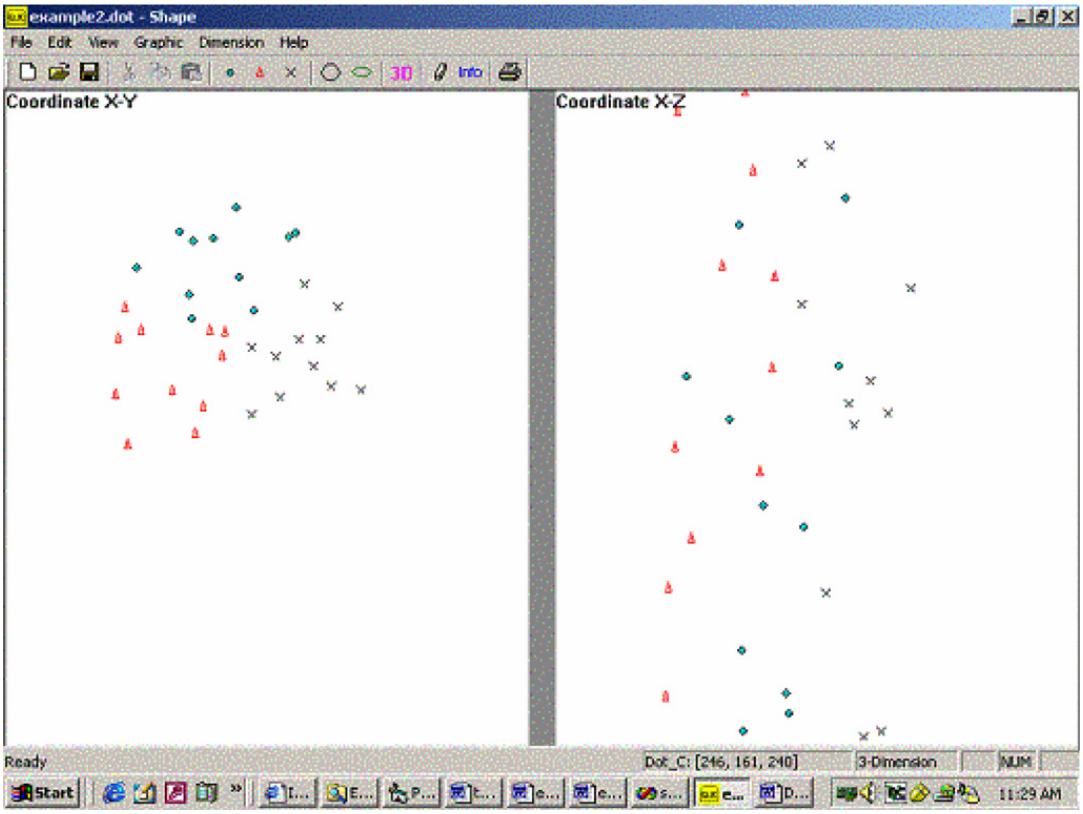

Fig. 8. Data in two 2D views.

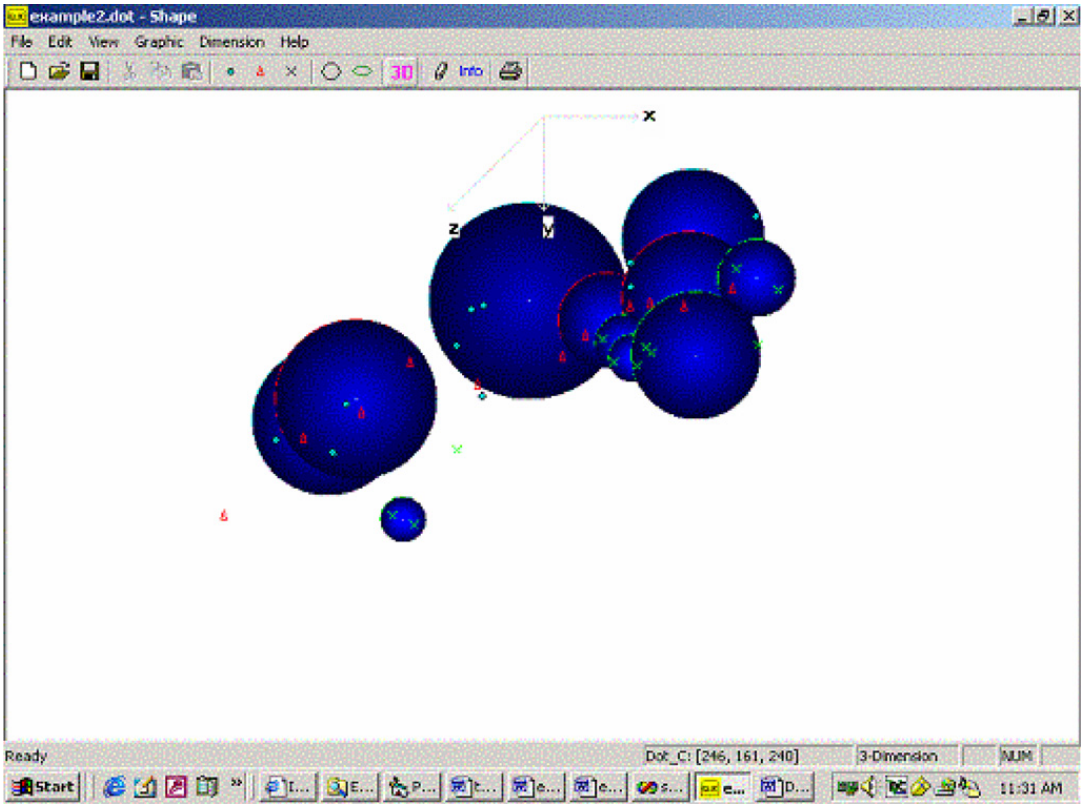

Fig. 9. Clustering result of MFMHBC in 3D view.

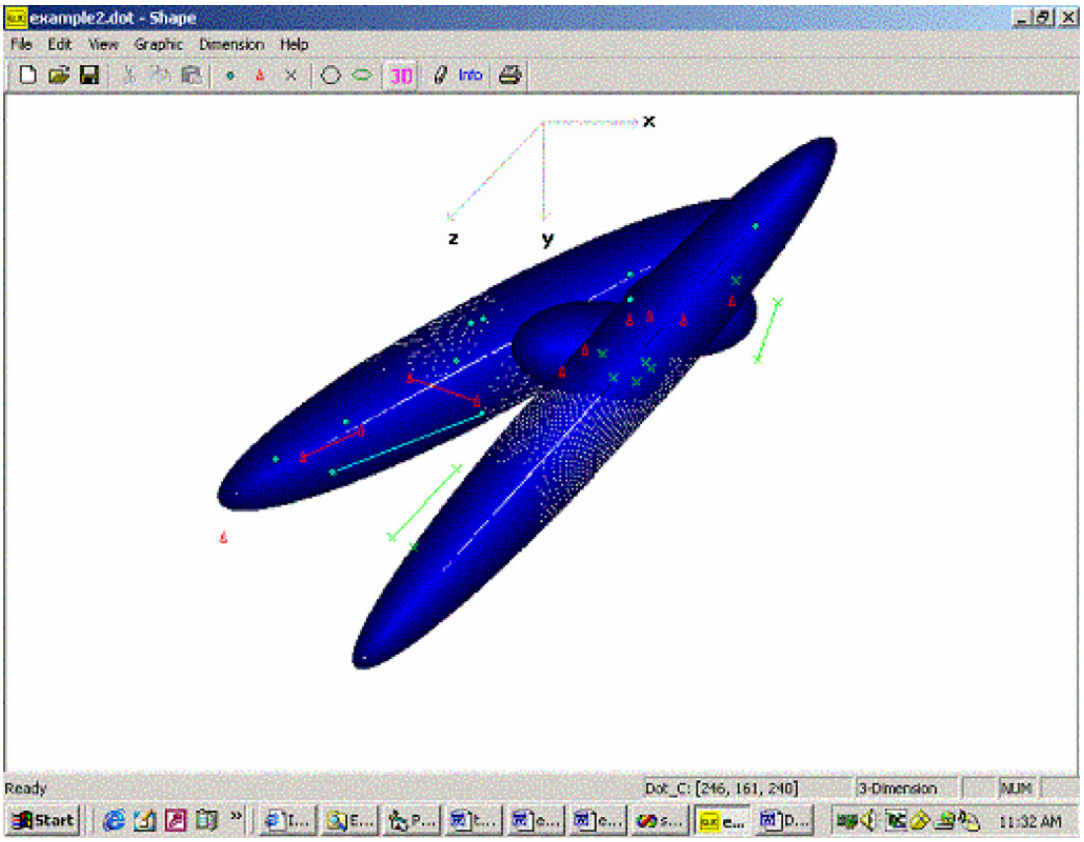

Fig. 10. Clustering result of MFMHEC in 3D view.

5.3. Case three – Intermixed data set of three classes in three-dimensional space

In this example, the data points in each class are Gaussian distributed in a three-dimensional space. We plotted the dataset in the X–Y and X–Z feature space, respectively, in Fig. 8 to show that the data points are intermixed in the space. Figs. 9 and 10 show the clustering results of the MHMHBC and MHMHEC algorithms, respectively. The clustering results on the data set are summarized in Table 3.

5.4. Comparison of MFMHBC with MFMHEC in multiple dimensions

Three series of experiments were conducted to observe the behavior and compare the performance of the MFMHBC and MFMHEC in multi-dimension spaces. In the first experiment, we changed the number of data points in the data sets with the dimension but with the number of classes fixed. In the second experiment, we set different dimensions for the data sets with the fixed number of data points and the distribution density. In the third experiment, we change the density of the data distributions so the level of intermixing increases.

Table 3
Clustering results on intermixed data set of three classes in three-dimension

Type of class	# of Data points	# of Clusters with MFMHBC	# of Clusters with MFMHEC
A	11	5	2
B	11	6	5
C	11	6	4
Total	33	17	11

Table 4
Clustering result associated with different size of data sets in 20 dimensions

# of Data point	# of Data dimension	# of Data classes	# of Clusters MFMHBC	# of Clusters MFMHEC
100	20	10	11	10
200	20	10	10	10
500	20	10	17	10
1000	20	10	11	11
2000	20	10	47	12
5000	20	10	26	12
10,000	20	10	54	15
25,000	20	10	454	17

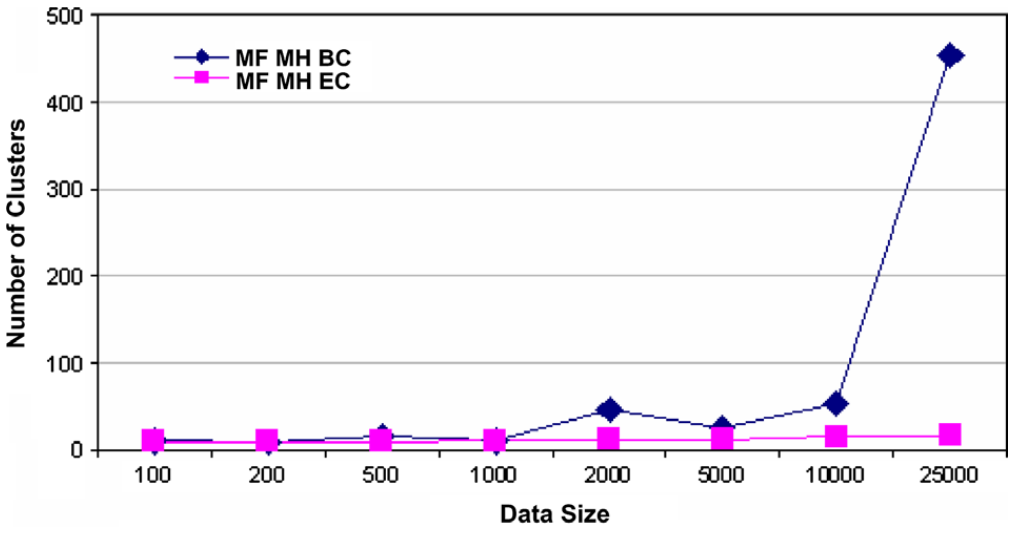

Fig. 11. Number of clusters resulted with different size of data sets in 20 dimensions.

5.4.1. Experiments with respect to number of data points in multiple dimensional space

In this experiment, we compared MFMHBC with MFMHEC on a group of data sets that range from small size to large size with 20 dimensions. We applied both algorithms to data sets that have 100, 200, 500, 1000, 2000, 5000, 10,000, and 25,000 points in 10 classes. In each experiment on the data set, we recorded the running time and the resulting number of clusters produced by each algorithm. The results are shown in Table 4, Figs. 11 and 12.

As described in Section 4, both MFMHBC and MFMHEC algorithms prevent the resulting hyper-balls and hyper-ellipsoids to contain data points of different classes. As the penalty of this restriction, the number of clusters accordingly increases as the number of data points goes up. It is possible that the increase of cluster

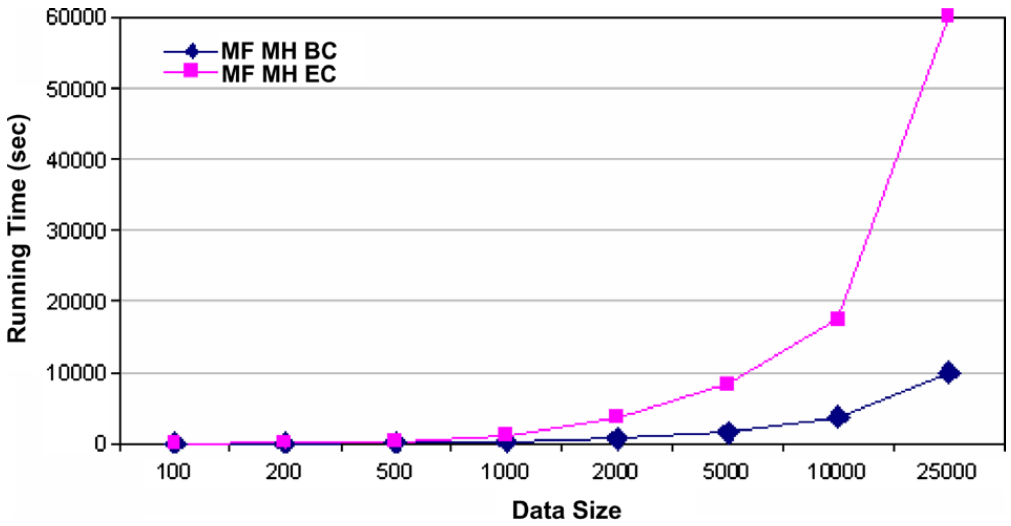

Fig. 12. Running time with different size of data sets in 20 dimensions.

Table 5
Clustering result associated with different dimensions of data sets with 5000 points

# of Data point	# of Data dimension	# of Data classes	# of Clusters MFMHBC	# of Clusters MFMHEC
5000	4	10	477	18
5000	8	10	445	16
5000	12	10	503	10
5000	16	10	694	12
5000	20	10	542	13

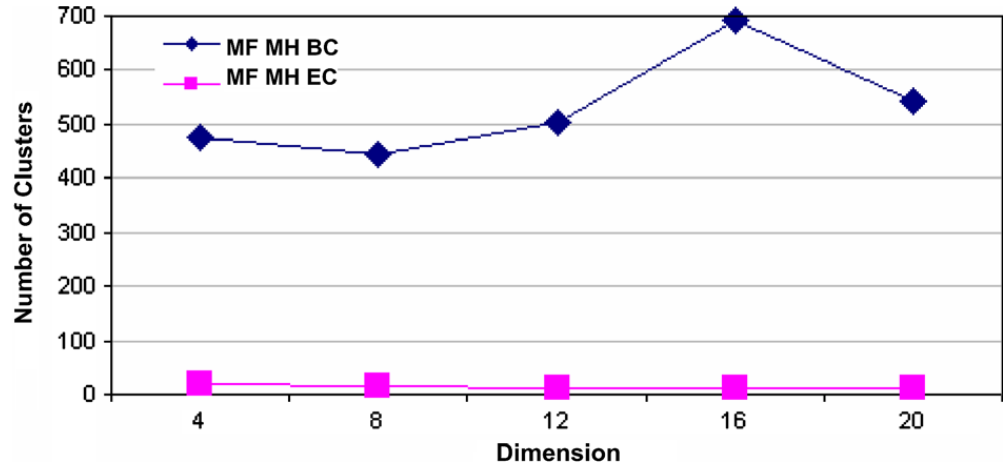

Fig. 13. Number of clusters resulted with different dimensions of data sets with 5000 points.

numbers is also attributable to the increased mix-up degree associated with the increase of the number of data points. However, the difference between the numbers of clusters generated by the algorithms is significant in these high-dimensional cases. Fig. 11 shows that the MFMHEC generates fewer clusters than MFMHBC. By contrast, Fig. 12 shows that MFMHBC requires less running time than MFMHEC. As a matter of fact, these results do not surprise us as that discussed in the algorithm analyses of Section 4.

5.4.2. Experiments with respect to different dimensions of the data sets

In this case, we compared MFMHBC with MFMHEC on a data set of 5000 points in different dimensions. We applied both algorithms to data sets that have 4, 8, 12, 16, 20 dimensions in 10 classes. The results are shown in Table 5, Figs. 13 and 14.

The results show again that the resulting number of clusters of the MFMHBC algorithm is significantly larger than that of the MFMHEC algorithm, while the computation time of the MFMHEC increases more rapidly than the MFMHBC.

5.4.3. Experiment with respect to data sets of different distribution densities

In this case, we compared MFMHBC with MFMHEC on a group of data sets in varying distribution densities. This is done by increasing the variance parameters of the Gaussian random number generator. The effect is the increased level of the mix-up of the data points between different classes. We applied both algorithms to data sets that have 5000 points in 20-dimensions with 10 classes. The data points in a specific class are dispersed in Gaussian distribution with different variances. The results are shown in Table 6, Figs. 15, and 16. The results follow the general trend of the above two experiments. It clearly indicates that hyper-ellipsoidal clustering is a viable model for partitioning large datasets intermixed with different classes in high-dimensional spaces, at the expanses of computational cost.

We must admit that all experiments were performed with synthetic data. The dimensions of the data are also not very high in these experiments (except the last) but in practical applications they would grow expo-

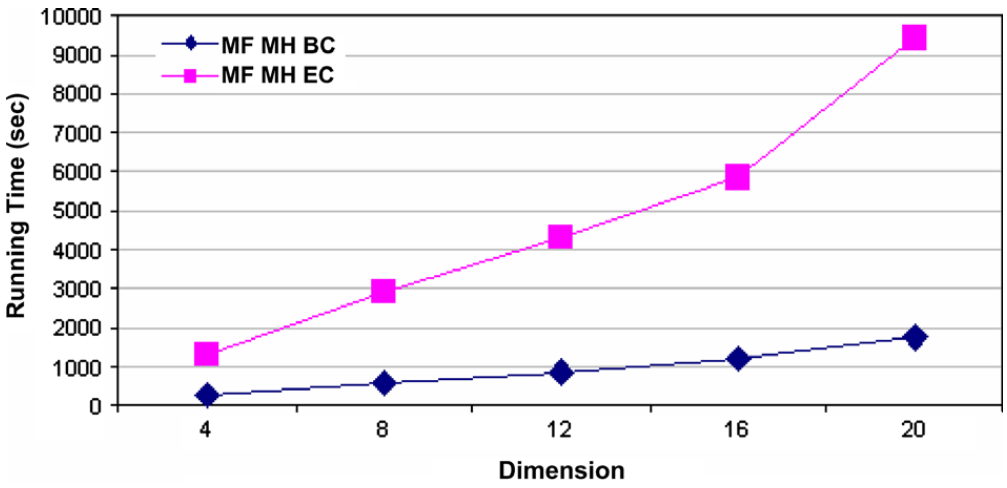

Fig. 14. Running time with respect to dimensionality on dataset of 5000 points.

Table 6
Clustering result with different densities of data sets with 5000 points in 20 dimensions

# of Data point	# of Data dimension	# of Data classes	Density (variance)	# of Clusters MFMHBC	# of Clusters MFMHEC
5000	20	10	3	10	10
5000	20	10	4.5	26	12
5000	20	10	5	542	13
5000	20	10	6	1856	22
5000	20	10	8	2854	33

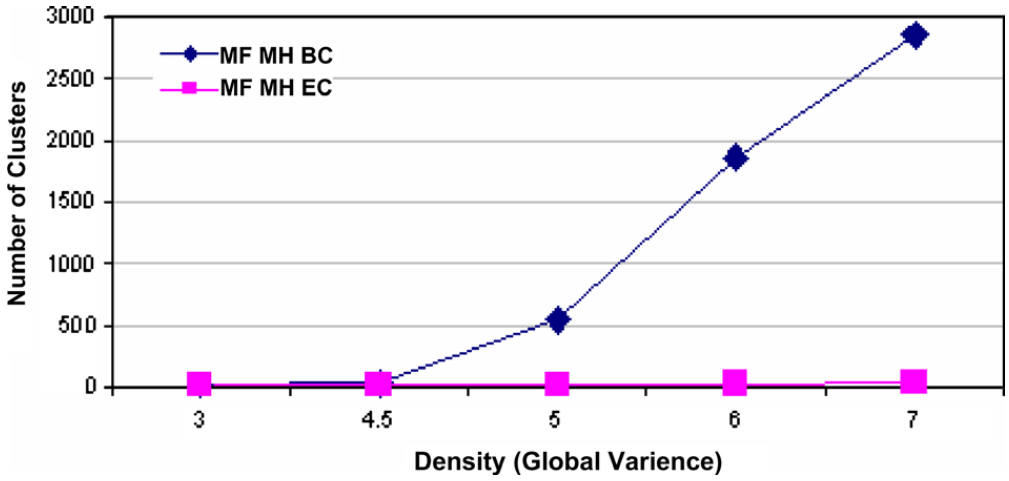

Fig. 15. Number of clusters resulted with different densities on data sets in 20 dimensions.

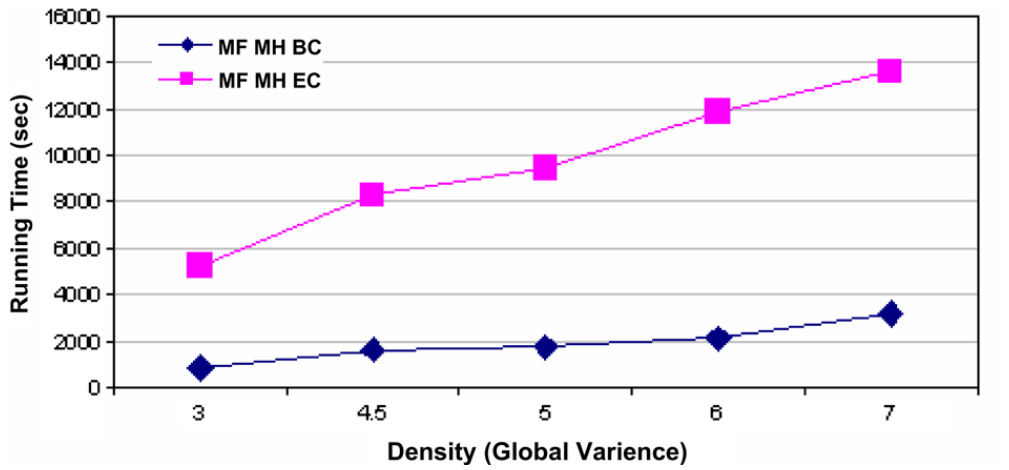

Fig. 16. Running time with different densities on data set in 20 dimensions.

nentially. Experiments with real data at this point will be a topic of future work. It would also be desirable to have an empirical evaluation with other algorithms such as DBSCAN, with a range of parameters.

6. Summary and conclusions

Geometrical clustering is to identify clusters embedded in the dataset using a set of geometrical entities to partition or group the data points in subsets. The technique has emerged as a key for business applications to make the huge amount of data stored in databases more meaningful and valuable [7]. The existing clustering methods such as partitioning methods, hierarchical methods, density-based methods, grid-based methods, and model-based methods all use certain kinds of geometrical primitives or structures. These algorithms shed some light on our new developed clustering algorithms. Geometric clustering utilizes proper spherical or ellipsoidal body to group similar objects in some high-dimensional space, with a metric defined appropriately for the similarity, and provides a concrete technique for these applications. There are still many interesting topics and challenging aspects that are worth to continue the research in this area, such as the association of the geometrical methods with statistical estimation and interpretations, the assessment of the clustering errors with respect to the different geometric models, and the application of the methods to real world situations.

References

[1] P.K. Agarwal, M. Sharir, Efficient algorithms for geometric optimization, ACM Computing Surveys 30 (4) (1998) 412–458.
[2] T. Asano, B. Bhattacharya, M. Keil, F. Yao, Clustering algorithms based on minimum and maximum spanning trees, in: Proceedings of the 4th Annual Symposium on Computational Geometry, 1988, pp. 252–257.
[3] G. Calafiore, Approximate of *n*-dimensional data using spherical and ellipsoidal primitives, IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans 32 (2) (2002) 269–278.
[4] V. Capoyleas, G. Rote, G. Woeginger, Geometric clustering, Journal of Algorithms (12) (1991) 341–356.
[5] K.L. Clarkson, Las Vegas algorithms for linear and integer programming when the dimension is small, Journal of the ACM 42 (2) (1995) 488–499.
[6] M. Ester, H. Kreigel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial database with noise, in: Proceedings of 2nd International Conference on Knowledge Discovery in Databases and Data Mining, August, 1996, pp. 226–231.
[7] T. Fukuda, Y. Morimoto, S. Morishita, T. Tokuyama, Data mining using two-dimensional optimized association rules: scheme, algorithms, and visualization, Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data 25 (2) (1996) 13–23.
[8] B. Gärtner, S. Schönherr, Exact primitives for smallest enclosing ellipses, in: Proceedings of the 13th Annual Symposium on Computational Geometry, 1997, pp. 430–432.
[9] B. Gärtner, S. Schönherr, Smallest enclosing balls – fast and exact, Technical Report, Free University Berlin, 1997. http://www.infethz.ch/personal/gaertner/publications.html.
[10] T.F. Gonzalez, Clustering to minimize the maximum inter-cluster distance, Theoretical Computer Science (38) (1985) 293–306.
[11] A. Guénoche, P. Hansen, B. Jaumard, Efficient algorithms for divisive hierarchical clustering with the diameter criterion, Journal of Classification (8) (1991) 5–30.
[12] S. Hasegawa, H. Imai, M. Inaba, N. Katoh, J. Nakano, Efficient algorithms for variance-based *k*-clustering, in: Proceedings of the 1st Pacific Conference on Computer Graphics and Applications, World Scientific Publishing Co., 1993, pp. 75–89.
[13] M. Inaba, N. Katoh, H. Imai, Applications of weighted Voronoi diagrams and randomization to variance-based *k*-clustering, in: Proceedings of the 10th Annual Symposium on Computational Geometry, 1994, pp. 332–339.
[14] J. Matoušek, Micha Sharir, Emo Welzl, A subexponential bound for linear programming, in: Proceedings of the 8th Annual Symposium on Computational Geometry, 1992, pp. 1–8.
[15] N. Megiddo, Linear-time algorithms for linear programming in R^3 and related problems, SIAM Journal of Computing 12 (4) (1983) 756–776.
[16] R.S. Michalski, R.E. Stepp, Automated construction of classifications: conceptual clustering versus numerical taxonomy, IEEE Transactions on Pattern Analysis and Machine Intelligence 5 (4) (1983) 396–410.
[17] R.T. Ng, J. Han, Efficient and effective clustering methods for spatial data mining, in: Proceedings of the 20th VLDB Conference, September, 1994, pp. 144–155.
[18] M.J. Post, Minimum spanning ellipsoids, in: Proceedings of the 16th Annual Symposium on Theory of Computing, 1984, pp. 108–116.
[19] R. Seidel, Linear programming and convex hulls made easy, in: Proceedings of the 6th Annual Symposium on Computational Geometry, 1980, pp. 211–215.
[20] B.W. Silverman, D.M. Titterington, Minimum covering ellipses, SIAM Journal of Scientific & Statistic Computing (1) (1980) 401–409.
[21] S. Skyum, A simple algorithm for computing the smallest enclosing circle, Information Processing Letters (3) (1991) 121–125.
[22] J.J. Sylvester, A question on the geometry of situation, Journal of Mathematics (1) (1857) 79.
[23] S.J. Wan, S.K.M. Wong, P. Prusinkiewicz, An algorithm for multidimensional data clustering, ACM Transactions on Mathematical Software 14 (2) (1988) 153–162.
[24] E. Welzl, Smallest enclosing disks (balls and ellipsoids), New Results and New Trends in Computer Science 555 (1991) 359–370.

Qinglu Kong received her Master of Science degree in the Department of Computer Science, University of Nebraska at Omaha in year 2002. Her research interests include database design and management, database related application development and maintenance, and data mining and knowledge discovery.

Qiuming Zhu is a professor of computer science at the University of Nebraska at Omaha. He received his Ph.D. in computer and systems engineering from Rensselaer Polytechnic Institute in 1986. His research interests include artificial intelligence, digital image processing and computer vision, pattern recognition, and neural networks.