The Islamic University of Gaza
Deanery of Postgraduate Studies
Faculty of Engineering
Computer Engineering Department

Metaheuristic Clustering Algorithm

by

Shadi I. Abudalfa

Supervisor

Prof. Mohammad Mikki

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master in Computer Engineering

2010 - 1431

(صفحة نتيجة الحكم على البحث) نتيجة الحكم من قبل لجنة المناقشة

DEDICATION

I would like to dedicate this work with my deep love to:

My Beloved Parents,

My Brothers and Sisters,

My Wife and Daughter

The Researcher

Shadi I. Abudalfa

Acknowledgements

I would like to thank Prof. Mohammad Mikki, my supervisor, for his support and guidance throughout the thesis. I would also like to thank my instructors in the field of computer engineering, particularly Prof. Ibrahim Abuhaiba, Prof. Hatem Hamad, and Dr. Wesam Ashour, for supporting my research and fostering my interest in the subject.

Contents

List of Tables

List of Figures

List of Abbreviations

ACDE	Automatic Clustering Differential Evolution
ANN	Approximate Nearest Neighbour
CE	Classification Entropy
CR	Crossover Rate
CRS	Controlled Random Search
CS	Clustering Scheme
CVI	Cluster Validity Index
DB	Davis-Bouldin
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DE	Differential Evolution
DERANDSF	DE with Random Scale Factor
DI	Dunn's Index
DLCKDT	Dynamic Linkage Clustering using KD-Tree
DP	Dense Point
EA	Evolution Algorithm
GA	Genetic Algorithm
GK	Gustafson-Kessel
ICEO	International Contest on Evolutionary Optimization
KD	K-Dimensional
PC	Partition Coefficient
PDF	Probability Density Function
PS-Based	Point Symmetry-Based
WEKA	Waikato Environment for Knowledge Analysis
XB	Xie-Beni's index

خوارزمية التجميع الإستدلالي

Metaheuristic Clustering Algorithm

قدم هذا البحث استكمالاً لمتطلبات الحصول على درجة الماجستير في هندسة الحاسوب

من قبل

شادي إبراهيم أبوضلفة

ملخص البحث

تتركز هذه الدراسة على واحدة من الموضوعات المهمة التي تخص الذكاء الإصطناعي في عملية تجميع البيانات ووضعها في تجمعات متشابهة. تستخدم خوارزميات التجميع على نطاق واسع ليس فقط لتنظيم وتصنيف البيانات ولكن لضغط البيانات وبناء نموذج ترتيب البيانات.

يمكن تلخيص مشكلة البحث في محاولة التعرف على تجمعات البيانات المتشابهة بطريقة سريعة وبدقة عالية خصوصاً مع مجموعة البيانات المتداخلة والكبيرة الحجم.

ولتنفيذ هذه الدراسة تم إجراء العديد من التجارب على أنواع مختلفة من البيانات الإصطناعية والحقيقية لإثبات مدى فاعلية الخورازميات التي تم تطويرها خلال البحث.

وتمثلت أهم نتائج البحث فيما يلي:

1) تطوير معادلة جديدة لقياس مدى التشابه بين البيانات لتسهيل عملية التجميع.
2) تطوير خوارزمية جديدة لتجميع البيانات بالإعتماد على تركيب البيانات في Kd-Tree.
3) تطوير أحد الخوارزميات المنشأة حديثا في مجال تجميع البيانات وجعلها قادرة على التعامل مع عدد أكبر من حزمات البيانات.

خوارزمية التجميع الإستدلالي

Metaheuristic Clustering Algorithm

by

Shadi I. Abudalfa

Abstract

In this thesis we describe an essential problem in data clustering and present some solutions for it. We investigate using distance measures other than Euclidean type for improving the performance of clustering. We also develop a new point symmetry-based distance measure and prove its efficiency. We develop a novel effective k-means algorithm which improves the performance of the k-mean algorithm. We develop a dynamic linkage clustering algorithm using kd-tree and we prove its high performance. The Automatic Clustering Differential Evolution (ACDE) is specific to clustering simple data sets and finding the optimal number of clusters automatically. We improve ACDE for classifying more complex data sets using kd-tree. The proposed algorithms do not have a worst-case bound on running time that exists in many similar algorithms in the literature.

Experimental results shown in this thesis demonstrate the effectiveness of the proposed algorithms. We compare the proposed algorithms with other famous similar algorithms. We present the proposed algorithms and their performance results in detail along with promising avenues of future research.

Keywords: *Data Clustering, Point Symmetry-Based Distance Measure, Validity Indices, Optimization, DE, ACDE, KD-Tree, Novel Effective K-Means, Dynamic Linkage Clustering*

Chapter 1

Problem Statement and Background

1.1 Introduction

A metaheuristic (Meta: in an upper level, Heuristic: to find) [1] is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space. Learning strategies are used to structure information in order to find efficiently near-optimal solutions. Metaheuristic algorithms [2] are approximate and usually non-deterministic techniques which constitute metaheuristic algorithms ranging from simple local search procedures to complex learning processes.

Clustering [3] is a division of data into groups of similar objects. Each group, called cluster, consists of objects that are similar within the cluster and dissimilar to objects of other clusters.

Representing data by fewer clusters necessarily loses certain fine details, but achieves simplification, and so may be considered as a form of data compression. It represents many data objects by few clusters models data by its clusters. Data modelling puts clustering in a historical perspective which is rooted in mathematics, statistics, and numerical analysis. Clustering is the subject of active research in several fields such as statistics, pattern recognition, artificial intelligence, and machine learning. From a practical perspective, clustering plays an outstanding role in data mining applications such as scientific data exploration, information retrieval and text mining, spatial database applications, Web analysis, marketing, medical diagnostics, computational biology, and many others.

The clustering problem has been addressed in many contexts and by researchers in many disciplines. This reflects its broad appeal and usefulness as one of the steps in exploratory data analysis.

Although classification [4] is an effective means for distinguishing groups or classes of objects, it requires the often costly collection and labelling of a large set of training tuples or patterns, which the classifier uses to model each group. It is often more desirable to proceed in the reverse direction: First partition the set of data into groups based on data similarity (e.g., using clustering), and then assign labels to the relatively small number of groups.

From a machine learning perspective clusters correspond to hidden patterns, the search for clusters is unsupervised learning [5], and the resulting system represents a data concept. Therefore, clustering is unsupervised learning of a hidden data concept.

Data mining deals with large databases that impose on clustering analysis additional severe computational requirements. These challenges led to the emergence of powerful broadly applicable data mining clustering methods.

1.1.1 Basic Definitions

The following terms are used throughout the thesis:

Definition 1.1: A Pattern (or feature vector) is a physical or abstract structure of objects, which are to be grouped properly by the clustering algorithm.

Definition 1.2: A Feature (or attribute) is an individual component of a pattern. It represents one of the traits based on which the patterns are to be classified [6].

Definition 1.3: A Cluster is a well defined collection of similar patterns and patterns from two different clusters must be dissimilar.

Definition 1.4: A Hard (or crisp) clustering algorithm assigns each pattern to one and only one cluster.

Definition 1.5: A Fuzzy clustering algorithm assigns each pattern to each cluster with a certain degree of membership.

Definition 1.6: A Distance Measure is a metric based on which the dissimilarity of the patterns are evaluated.

Now we may formalize the definition of the clustering problem in the following way. Let $P = \{P_1, P_2, ..., P_n\}$ be a set of n patterns each having *d* features. These patterns can also be represented by a profile data matrix $Z_{n\times d}$ having n d-dimensional row vectors. The i-th row vector $\vec{Z}_i$ characterises the i-th object from the set P and each element $z_{i,j}$ in $\vec{Z}_i$ corresponds to the j-th real value feature (j =1, 2, ...,d) of the i-th pattern (i=1,2,...., n). Given such an $Z_{n\times d}$ a partitional clustering algorithm tries to find out a partition $C = \{C_1, C_2,....., C_k\}$ such that similarity of the patterns in the same cluster C_i is maximum and patterns from different clusters differ as far as possible. The partitions should maintain the following properties:

1) Each cluster should have at least one pattern assigned. i.e. $C_i \neq \Phi \ \forall\, i \in \{1,2,..., k\}$
2) Two different clusters should have no pattern in common. i.e. $C_i \cap C_j = \Phi \ \forall\, i \neq j$ and $i, j \in \{1,2,..., k\}$.
3) Each pattern should definitely be attached to a cluster i.e. $\bigcup_{i=1}^{k} C_i = P$

Since the given data set can be partitioned in a number of ways maintaining all of the above properties, a fitness function or in other words some measure of the adequacy of the partitioning must be defined. Then the problem turns out to be one of finding a partition C^* of optimal or near optimal adequacy as compared to all other feasible solutions $\mathbf{C} = \{C^1, C^2,....., C^{N(n,k)}\}$ where $N(n,k) = \frac{1}{k!}\sum_{i=1}^{k}(-1)^i \binom{k}{i} (k-i)^i$ is the

number of feasible partitions. Where n and k in N(n,k) are described in the previous paragraph. This optimization problem can be represented as:

$$\text{optimize}_C \; f(Z_{\text{n}\times\text{d}}, \text{C}) \tag{1.1}$$

where C is a single partition from the set $\mathbf{C}$ and f is a fitness function that quantifies the goodness of a partition on the basis of the similarity or dissimilarity measures of the patterns.

1.1.2 Clustering Validity Indices

In most of the cases, a class of statistical-mathematical functions, based on the notion of similarity or dissimilarity between the data points, is employed for judging the soundness of the clustering solutions provided by an algorithm. The functions are collectively known as the cluster validity indices (CVIs). Generally, a cluster validity index serves two purposes. First, it can be used to determine the number of clusters, and second, it finds out the corresponding best partition. One traditional approach for determining the optimum number of classes is to run the algorithm repeatedly with different number of classes as input and then to select the partitioning of the data resulting in the best validity measure [7]. Ideally, a validity index should take care of the following aspects of the partitioning:

1) ***Cohesion:*** Patterns in one cluster should be as similar to each other as possible. The fitness variance of the patterns in a cluster is an indication of the cluster's cohesion or compactness.
2) ***Separation:*** Clusters should be well separated. Distance among the cluster centers, (may be their Euclidean distance) gives an indication of cluster separation.

For crisp clustering, some of the well-known indices available in the literature are the Dunn's index (DI) [8], Calinski-Harabasz index [9], Davis-Bouldin (DB) index [10],

PBM index [11], and the CS measure. All these indices behave somewhat like an objective function, minimization/maximization of which is expected to lead to an optimal partitioning of the data set under test. Because of their optimizing character, the cluster validity indices are best used in association with any optimization algorithm for finding out the appropriate clusters in a data set. In the following Section, we discuss one of the most validity indices used in the literature, pertinent to clustering problems.

Recently, Chou et al. [12] proposed the CS measure for evaluating the validity of a clustering scheme. Before applying the CS measure, centroid of a cluster is computed by averaging the data vectors belonging to that cluster using the formula,

$$\overrightarrow{m_i} = \frac{1}{N_i} \sum_{\overrightarrow{Z_j} \in C_i} \overrightarrow{Z_j} \tag{1.2}$$

A distance metric between any two data points $\overrightarrow{\mathrm{Z_p}}$ and $\overrightarrow{\mathrm{Z_r}}$ is denoted by $d(\overrightarrow{Z_p}, \overrightarrow{Z_r})$. Then the CS measure can be defined as,

$$CS(k) = \frac{\sum_{i=1}^{k} [\frac{1}{|C_i|} \sum_{\vec{\mathrm{Z}}_p \in C_i} \max_{\overrightarrow{\mathrm{Z_y}} \in C_i} \{d(\vec{Z}_p, \vec{Z}_y)\}]}{\sum_{i=1}^{k} \min_{j \in k, j \neq i} \{d(\vec{m}_i, \vec{m}_j)\}} \tag{1.3}$$

As can easily be perceived, this measure is a function of the ratio of the sum of within-cluster scatter to between-cluster separation and has the same basic rationale as the DI and the DB measures. According to Chou et al., CS measure is more efficient in tackling clusters of different densities and/or sizes than the other popular validity measures, the price being paid in terms of high computational load with increasing *k* and *n*.

1.1.3 Clustering Algorithms

There are thousands of clustering techniques one can encounter in the literature. Most of the existing data clustering algorithms can be classified as hierarchical or

partitional as shown in Figure 1.1. Within each class, there exists a wealth of sub-class which includes different algorithms for finding the clusters.

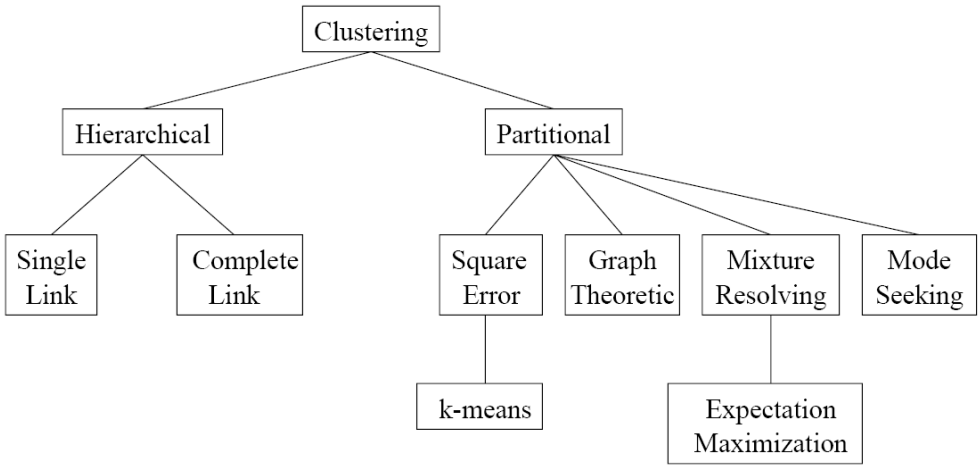


Figure 1.1: A taxonomy of clustering approaches

(adapted from Jain, Murty, and Flynn [24])

While hierarchical algorithms [13] build clusters gradually (as crystals are grown), partitioning algorithms [14] learn clusters directly. In doing so, they either try to discover clusters by iteratively relocating points between subsets, or try to identify clusters as areas highly populated with data.

Density based algorithms [15] typically regard clusters as dense regions of objects in the data space that are separated by regions of low density. The main idea of density-based approach is to find regions of high density and low density, with high-density regions being separated from low-density regions. These approaches can make it easy to discover arbitrary clusters.

Recently, a number of clustering algorithms have been presented for spatial data, known as grid-based algorithms. They perform space segmentation and then aggregate appropriate segments [16].

Many other clustering techniques are developed, primarily in machine learning, that either have theoretical significance, are used traditionally outside the data mining community, or do not fit in previously outlined categories.

So we can summarize the clustering algorithms as follows [17]:

- Hierarchical Methods
  - Agglomerative Algorithms
  - Divisive Algorithms
- Partitioning Methods
  - Relocation Algorithms
  - Probabilistic Clustering
  - K-medoids Methods
  - K-means Methods
  - Density-Based Algorithms
    - Density-Based Connectivity Clustering
    - Density Functions Clustering
- Grid-Based Methods
- Methods Based on Co-Occurrence of Categorical Data
- Constraint-Based Clustering
- Clustering Algorithms Used in Machine Learning
  - Gradient Descent and Artificial Neural Networks
  - Evolutionary Methods
- Scalable Clustering Algorithms
- Algorithms For High Dimensional Data
  - Subspace Clustering
  - Projection Techniques
  - Co-Clustering Techniques

Clustering is a challenging field of research in which its potential applications pose their own special requirements. The following are typical requirements of clustering in data mining:

- Type of attributes algorithm can handle.
- Scalability to large data sets.
- Ability to work with high dimensional data [18,19].
- Ability to find clusters of irregular shape.
- Handling outliers (noise).
- Time complexity.
- Data order dependency.
- Labelling or assignment (hard or strict vs. soft or fuzzy [20,21,22]).
- Reliance on a priori knowledge and user defined parameters.
- Interpretability of results.

However, clustering is a difficult problem combinatorially, and differences in assumptions and contexts in different communities have made the transfer of useful generic concepts and methodologies slow to occur.

1.2 Problem Statement

Many algorithms in literature like ACDE algorithm suffer from an important fault of using Euclidean distance for calculating symmetry measure between data clusters. Using Euclidean distance is improper for classifying overlapping and arbitrary shaped clusters. So many other distance measures are developed in literature for improving calculating of symmetry measure to classify complex data sets.

Symmetry is considered a pre-attentive feature which enhances recognition and reconstruction of shapes and objects. Almost every interesting area around us consists of some generalized form of symmetry. As symmetry is so common in the natural

world, it can be assumed that some kind of symmetry exists in the clusters also. Based on this, some distance measures have proposed as a symmetry-based clustering technique. Points are assigned to a particular cluster if they present a symmetrical structure with respect to the cluster center. These measures are better than using Euclidean distance for classifying symmetrical shaped clusters but improper for classifying arbitrary shaped clusters.

Some algorithms calculate connectivity of each data point to its cluster by depending on density reachability. A cluster, which is a subset of the points of the data set, satisfies two properties:

1) All points within the cluster are mutually density-connected.
2) If a point is density-connected to any point of the cluster, it is part of the cluster as well.

These algorithms can find arbitrarily shaped clusters, but they require parameters that are mostly sensitive to clustering performance. From other side, these algorithms need to detect nearest neighborhood of each data point which cause time consuming.

We tackled with this defect, and conclude that we can improve performance of classification by using other distance measures instead of Euclidean type and testing connectivity of each data point with its cluster by suitable method without increasing time complexity and without using additional parameters. By using suitable distance measure and checking density reachability of data points with its cluster, we can classify complex data sets which have overlapped and arbitrary shaped clusters.

1.3 Thesis Contribution

The contribution of the thesis is that we developed a new point symmetry-based distance measure by using *k*d-tree for classifying complex data sets, we improved the performance of K-mean and ACDE algorithms, and we developed an original algorithm

by using *k*d-tree. Experimental results are shown in this thesis to demonstrate the effectiveness of the proposed algorithms. We compared the proposed algorithms with other famous algorithms. We present the proposed algorithms and their results in detail along with promising avenues of future research.

The rest of the thesis is organized as follows: Chapter 2 describes review of literature and related studies. The chapter describes differential evolution algorithm which is used by the Automatic Clustering Differential Evolution (ACDE) algorithm for finding the optimum number of clusters. The chapter presents ACDE algorithm and describes a simple modification of classical differential evolution algorithm which is implemented in literature for improving the performance of ACDE algorithm. The chapter offers a new point symmetry-based distance measure which is described in literature for improving point symmetric distance measure and using it to cluster overlapping and arbitrary shaped clusters with variable length. The chapter presents *k*d-tree which is the most important multidimensional structure for storing a finite set of data points from k-dimensional space.

Chapter 3 illustrates our contribution by using *k*d-tree for developing an improved PS-Based distance measure. The chapter describes our contribution for improving efficiency of k-means algorithm. We called the proposed algorithm as a novel effective k-means algorithm. We used an improved PS-Based distance measure for developing the proposed algorithm. The chapter illustrates our proposed original algorithm for classifying complex data sets. We called the proposed algorithm a Dynamic Linkage Clustering using KD-Tree (DLCKDT). We used selected nodes from *k*d-tree to develop this algorithm. The Chapter illustrates our contribution for improving efficiency of ACDE to classify complex data sets automatically.

Chapter 4 illustrates our tools for testing ACDE algorithm and shows experimental results. The chapter illustrates an important fault of using ACDE algorithm and describes an insufficiency of using Euclidean distance for calculating symmetry measure to classify overlapping and arbitrary shaped clusters. The chapter shows many experiments for testing PS-Based distance measure with k-means algorithm and demonstrate that it is insufficient for classifying complex data sets. Experimental results are shown in this chapter to demonstrate the effectiveness of the proposed algorithms. We used synthetic and real data sets for testing efficiency of the proposed algorithm. Finally, Chapter 5 concludes the thesis and presents suggestions for future work.

Chapter 2

Review of Literature and Related Studies

2.1 K-Means Algorithm

K-means uses a two-phase iterative algorithm to minimize the sum of point-to centroid distances, summed over all k clusters: The first phase is "batch" updates, where each iteration consists of reassigning points to their nearest cluster centroid, all at once, followed by recalculation of cluster centroids. The second phase uses "online" updates, where points are individually reassigned. By doing so will reduce the sum of distances, and cluster centroids are recomputed after each reassignment. Each iteration during this second phase consists of one pass though all the points. K-means can converge to a local optimum which is a partition of points in which moving any single point to a different cluster increases the total sum of distances [23].

The K-means Algorithm is presented as follows:

(1) Initialize K center locations (C_1, ..., C_K).

(2) Assign each *data point* X_i to its nearest cluster center C_j.

(3) Update each cluster center C_j to be the mean of all X_i that have been assigned as closest to it.

(4) Calculate $summation\ of\ minimum\ distance\ measures$

$$D = \sum_{i=1}^{n} \left[\min_{j=(1\ldots K)} d(X_i, C_j)\right]^2$$

(5) If the value of D has converged, then return (C_1, ..., C_K); else go to Step 2.

Thus k-means has a hard membership function. Furthermore, k-means has a constant weight function, i.e. all patterns belonging to a cluster have equal influence in computing the centroid of the cluster. The k-means has two main advantages [24]:

1) It is very easy to implement.

2) The time complexity is only O(n) (n being the number of data points), which makes it suitable for large data sets.

However the k-means suffers from the following disadvantages:

1) The user has to specify the number of classes in advance.
2) The performance of the algorithm is data-dependent.
3) The algorithm uses a greedy approach and is heavily dependent on the initial conditions. This often leads k-means to converge to sub-optimal solutions.

Stephen J. Redmond and Conor Heneghan [25] presented a method for initialising the K-means clustering algorithm using *k*d-tree. The proposed method depends on the use of a *k*d-tree to perform a density estimation of the data at various locations. They used a modification of Katsavounidis' algorithm, which incorporates this density information, to choose *K* seeds for the K-means algorithm.

K. Mumtaz1 and K. Duraiswamy [26], proposed a novel density based k-means clustering algorithm to overcome the drawbacks of DBSCAN and k-means clustering algorithms. The result is an improved version of k-means clustering algorithm. This algorithm performs better than DBSCAN while handling clusters of circularly distributed data points and slightly overlapped clusters. But there is a limitation for this algorithm. It requires a prior specification of some parameters, and the clustering performance is affected by these parameters.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a data clustering algorithm proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu in 1996 [27]. It is a density-based clustering algorithm because it finds a number of clusters starting from the estimated density distribution of corresponding nodes. DBSCAN is one of the most common clustering algorithms and also most cited in scientific literature.

2.2 Differential Evolution Algorithm

This section presents differential evolution algorithm which is used by ACDE algorithm for finding the optimum number of clusters.

2.2.1 Introduction

The Differential Evolution (DE), proposed by Storn and Price in [28], [29] may be also seen as a simple real-coded Genetic Algorithm (GA). The first written article on DE appeared as a technical report in 1995. Since then, DE has proven itself in competitions like the IEEE's International Contest on Evolutionary Optimization (ICEO) in 1996 and 1997.

In DE community, the individual trial solutions (which constitute a population) are referred as parameter vectors or genomes. DE operates through the same computational steps as employed by a standard Evolution Algorithm (EA). However, unlike traditional EAs, DE employs difference of the parameter vectors to explore the objective function landscape. In this respect, it owes a lot to its two ancestors namely – the Nelder-Mead algorithm [30 , 31], and the Controlled Random Search (CRS) algorithm [32], which also relied heavily on the difference vectors to perturb the current trial solutions. Like other population-based search techniques, DE generates new points (trial solutions) that are perturbations of existing points, but these deviations are neither reflections like those in the CRS and Nelder-Mead methods, nor samples from a predefined probability density function, like those in Evolutionary Strategies (ES) [33,34]. Instead, DE perturbs current generation vectors with the scaled difference of two randomly selected population vectors. To produce a trial vector in its simplest form DE adds the scaled, random vector difference to a third randomly selected population vector. In the selection stage, the trial vector competes against the population vector of the same index. Once the last trial vector has been tested the survivors of all the pair

wise competitions become permanent for the next generation in the evolutionary cycle. In the following sections, we discuss each of these steps in sufficient details.

2.2.2 Differential Evolution: A First Glance

DE is a simple evolutionary algorithm. It works through a simple cycle of stages, presented in Figure 2.1. Below we explain each stage separately.

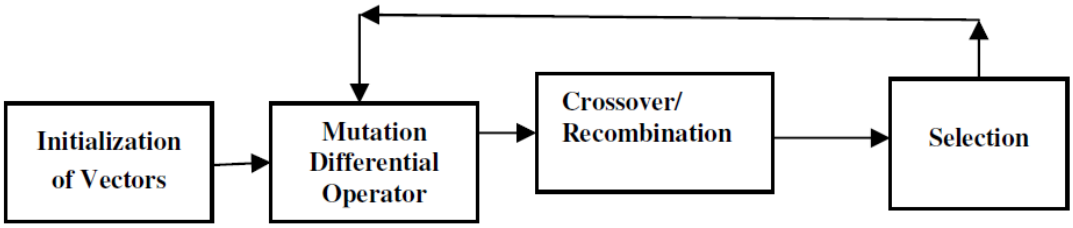


Figure 2.1: The main stages of differential evolution algorithm
(adapted from Das, Abraham and Konar [37])

2.2.2.1 Initialization of the Parameter Vectors

DE searches for a global optimum point in a D-dimensional continuous hyperspace. It begins with a randomly initiated population of N D dimensional real-valued parameter vectors. Each vector, also known as genome/chromosome, forms a candidate solution to the multi dimensional optimization problem.

We represent subsequent generations in DE by discrete time steps like t = 0, 1, 2 ...t, t+1 etc. In most of the DE literatures, the successive generations are represented by G, G+1, G+2…or g, g+1 etc. [35] but we adopt a slightly different notation in order to remain consistent with the notations used in other chapters of the thesis and also to facilitate the mathematical analysis of DE undertaken here. Since the parameter vectors are likely to be changed over different generations, we adopt the following notation for representing the *i*-th vector of the population at the current generation (i.e. at time t) as:

$$\overrightarrow{X_i}(t) = [X_{i,1}(t), X_{i,2}(t), \ldots\ldots, X_{i,D}(t)]^T \qquad (2.1)$$

where $i = 1, 2, \ldots, N$, and N is the number of initiated vectors.

For each parameter of the problem, there may be a certain range within which the value of the parameter should lie for better search results. At the very beginning of a DE run or at t = 0, problem parameters or independent variables are initialized somewhere in their feasible numerical range. So, if the *j*-th parameter of the given problem has its lower and upper bounds as $x_{min,j}$ and $x_{max,j}$ respectively and $rand_{i,j}(0,1)$ denotes the *j*-th instantiation of a uniformly distributed random number lying between 0 and 1 for the *i*-th vector, then we may initialize the *j*-th component of the *i*-th population members as,

$$x_{i,j}(0) = x_{min,j} + rand_{i,j}(0,1) * (x_{max,j} - x_{min,j}) \tag{2.2}$$

The process is illustrated in Figure 2.2 for 10 parameter vectors in two dimensional search space. Closed curves in Figure 2.2 denote constant cost contours, where a given cost function $f(x_1, x_2)$ is constant.

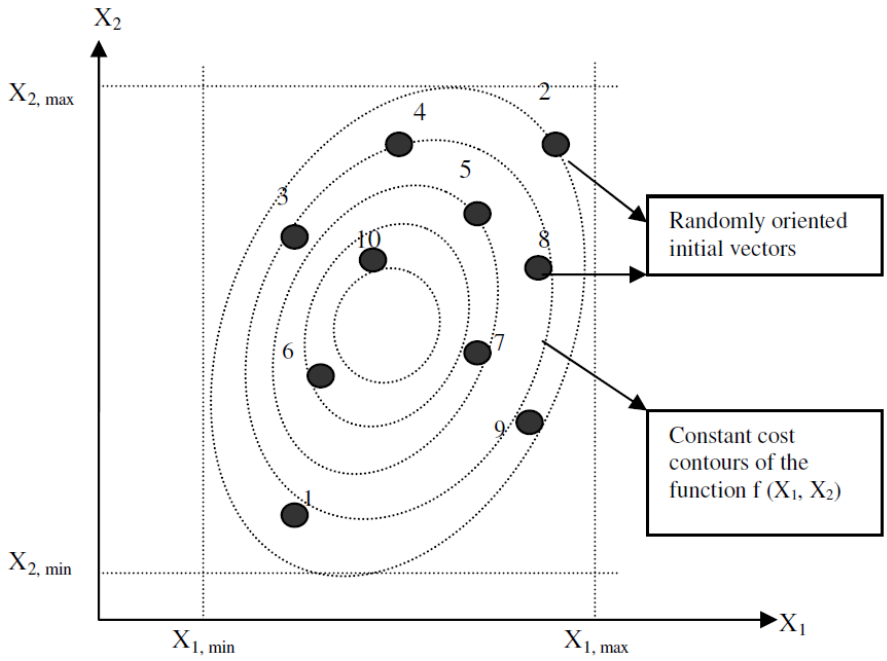


Figure 2.2: Initializing a DE population of *N* = 10, on a two-dimensional parametric space
(adapted from Das, Abraham and Konar [37])

In Figure 2.3, we show the constant cost contours for the two dimensional sphere functions on the $x_1 - x_2$ parameter plane.

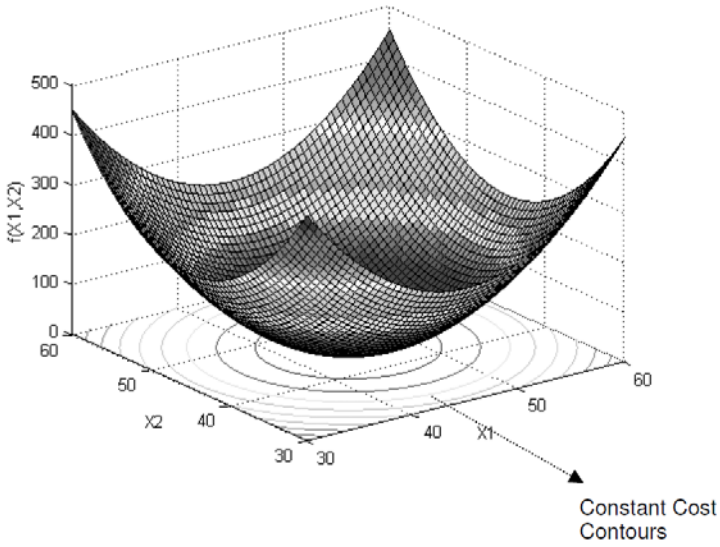


Figure 2.3: Constant cost contours for a sphere function
(adapted from Das, Abraham and Konar [37])

2.2.2.2 Mutation with Differential Operators

Biologically 'mutation' means a sudden change in the gene characteristics of a chromosome. In the context of the evolutionary computing paradigm, however, mutation is also seen as a change or perturbation with a random element. Most of the real-coded EAs typically simulate the mutation effects with additive increments, which are randomly generated by a predetermined Probability Density Function (PDF) [36]. DE, however, applies a uniform PDF not to generate increments, but to randomly sample vector differences like $\Delta\vec{X}_{r2,r3} = (\vec{X}_{r2} - \vec{X}_{r3})$. In DE, mutation amounts to creating a donor vector $\vec{V}_i(t)$ for changing each population member $\vec{X}_i(t)$, in each generation (or in one iteration of the algorithm). To create $\vec{V}_i(t)$ for each i-th member of the current population (also called the target vector), three other distinct parameter vectors, say the vectors $\vec{X}_{r_1^i}$, $\vec{X}_{r_2^i}$, and $\vec{X}_{r_3^i}$ are picked up randomly from the current population. The indices r_1^i, r_2^i and r_3^i are mutually exclusive integers randomly chosen from the range [1, N], which are also different from the base vector index i. These indices are randomly generated once for each mutant vector. Now the difference of any two of these three vectors is scaled by a scalar number *F* and the scaled difference is

added to the third one whence we obtain the donor vector $\vec{V}_i(t)$. We can express the process as,

$$\vec{V}_i(t) = \vec{X}_{r_1^i}(t) + F.(\vec{X}_{r_2^i}(t) - \vec{X}_{r_3^i}(t)) \quad (2.3)$$

Actually it is the mutation scheme that demarcates among the different kinds of DE schemes. Here presently we discuss one of the most popular schemes for the formation of the donor vector. The process is illustrated in Figure 2.4.

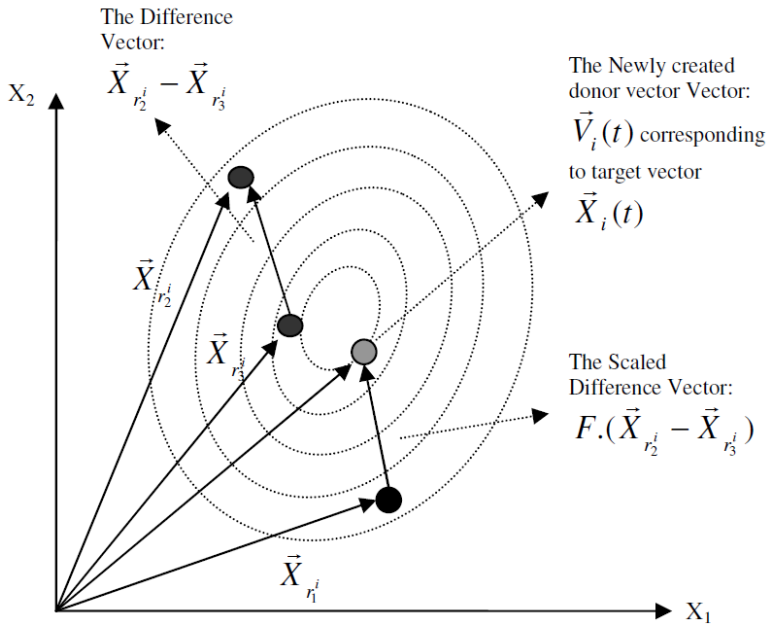


Figure 2.4: Illustrating a simple DE mutation scheme in two-dimensional parametric space
(adapted from Das, Abraham and Konar [37])

2.2.2.3 Crossover

To increase the potential diversity of the population, a crossover operation comes into play after generating the donor vector through mutation. The DE family of algorithms can use two kinds of crossover schemes - exponential and binomial. The donor vector exchanges its body parts i.e. components with the target vector $\vec{X}_i(t)$ under this operation to form the trial vector $\overrightarrow{U_i}(t) = [U_{i,1}(t), U_{i,2}(t), \ldots\ldots, U_{i,D}(t)]^T$.

In exponential crossover, we first choose an integer n randomly among the numbers [0, D-*1*]. This integer acts as a starting point in the target vector, from where the crossover or exchange of components with the donor vector starts. We also choose another integer L from the interval [1, D]. L denotes the number of components; the donor vector actually contributes to the target. After a choice of n and L the trial vector:

$$u_{i,j}(t) = \begin{cases} v_{i,j}(t), \text{for } j=\langle n\rangle_D,\langle n+1\rangle_D,\ldots,\langle n+L-1\rangle_D \\ x_{i,j}(t), \text{for all other } j\in[0,D-1] \end{cases} \quad (2.4)$$

where the angular brackets $\langle \quad \rangle_D$ denote a modulo function with modulus D. The integer L is drawn from [1,2,…, D] according to the following lines of pseudo code.

$L = 0$;

do

{

$L=L+1$;

} while ($rand$ (0, 1) < CR) AND ($L<D$));

Hence in effect Probability ($L \geq v$) = $(CR)^{v-1}$ for any $v > 0$. *'CR'* is called crossover rate and it appears as a control parameter of DE just like F. For each donor vector, a new set of n and L must be chosen randomly as shown above.

On the other hand, binomial crossover is performed on each of the D variables whenever a randomly picked number between 0 and 1 is less than or equal to the CR value. In this case the number of parameters inherited from the donor has a (nearly) binomial distribution. The scheme may be outlined as:

$$u_{i,j,G} = \begin{cases} v_{i,j,G}, & \text{if } (rand_{i,j}(0,1)\leq CR \text{ or } j=j_{rand}) \\ x_{i,j,G} & \text{otherwise} \end{cases} \quad (2.5)$$

Where $rand_{i,j}(0,1) \in [0,1]$ is a uniformly distributed random number, which is called anew for each j-th component of the i-th parameter vector. $j_{rand} \in$ [1,2,....,D] is a randomly chosen index, which ensures that $\vec{U}_{i,G}$ gets at least one component from

$\vec{V}_{i,G}$. It is instantiated once for each vector in one generation. We note that for this additional demand, *CR* is only approximating the true probability P_{Cr} that a component of the trial vector will be inherited from the donor. Also, one may observe that in a two-dimensional search space, three possible trial vectors may result from uniformly crossing a mutant/donor vector $\vec{V}_i(t)$ with the target vector $\vec{X}_i(t)$. These trial vectors are

1) $\vec{U}_i(t) = \vec{V}_i(t)$ such that both the components of $\vec{U}_i(t)$ are inherited from $\vec{V}_i(t)$.
2) $\vec{U}_i{}'(t)$, in which the first component (j = 1) comes from $\vec{V}_i(t)$ and the second one (j = 2) from $\vec{X}_i(t)$.
3) $\vec{U}_i{}''(t)$, in which the first component (j = 1) comes from $\vec{X}_i(t)$ and the second one (j=2) from $\vec{V}_i(t)$.

The possible trial vectors due to uniform crossover are illustrated in Figure 2.5.

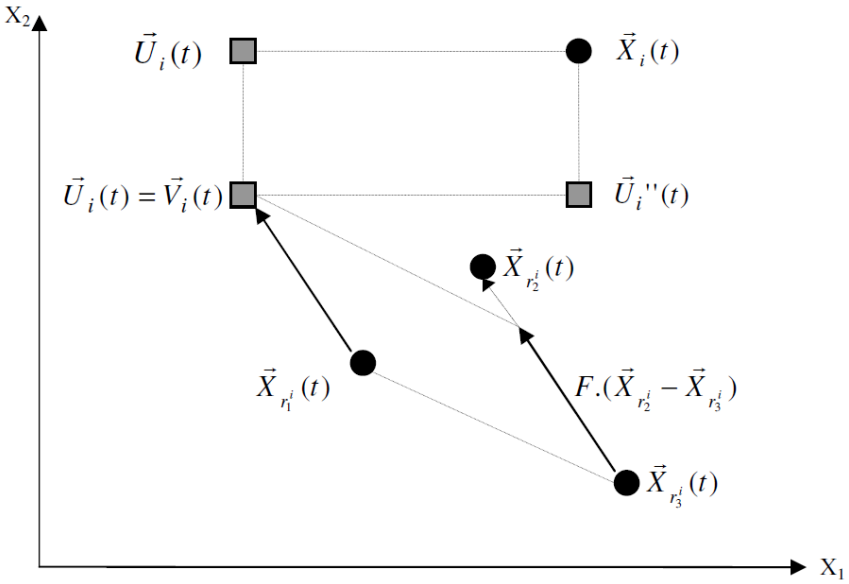


Figure 2.5: Different possible trial vectors formed due to uniform/binomial crossover between the target and the mutant vectors in two-dimensional search space
(adapted from Das, Abraham and Konar [37])

2.2.2.4 Selection

The last stage of a DE-iteration is the 'selection' i.e. deciding who between the target vector $\vec{X}_i(t)$ and the newly formed trial vector $\vec{U}_i(t)$ will survive to the next

generation. The decision whether original $\vec{X}_i(t)$ will be retained in the population or will be replaced by $\vec{U}_i(t)$ in the next time step t+1 is entirely dependent upon the 'survival of the fittest' concept. If the trial vector yields a better fitness value it will replace the target vector in the next time step. Here by better fitness value we mean a lower value of the objective function in case of a minimization problem, and a higher value of the same if it is a maximization problem. The selection operation may be outlined as:

$$\vec{\mathrm{X}}_{\mathrm{i}}(\mathrm{t}+1) = \begin{cases} \vec{\mathrm{U}}_{\mathrm{i}}(\mathrm{t}) \text{ if } f(\vec{\mathrm{U}}_i(\mathrm{t})) \leq f(\vec{\mathrm{X}}_i(\mathrm{t})) \\ \vec{\mathrm{X}}_{\mathrm{i}}(\mathrm{t}) \text{ if } f(\vec{\mathrm{U}}_i(\mathrm{t})) > f(\vec{\mathrm{X}}_i(\mathrm{t})) \end{cases} \tag{2.6}$$

where $f(\vec{\mathrm{X}})$ is the function to be minimized. Since the selection process employs a binary decision, i.e. any one between the target vector and its offspring survives the population size remains fixed throughout generations. The fitness of the population members either improves over generations or remains unchanged, but never deteriorates.

2.2.2.5 Summary of DE Iteration

An iteration of the classical DE algorithm consists of the four basic steps: initialization of a population of search variable vectors, mutation, crossover or recombination and finally selection. After having illustrated these stages, we now formally present the whole of the algorithm in a pseudo-code below. The algorithm is presented in the literature.

Pseudo-code for the DE algorithm:

Step 1. Set the generation number t = 0 and randomly initialize a population of NP individuals $P_t(t) = \left[\overrightarrow{X_1}(t), \ldots, \overrightarrow{X_{NP}}(t)\right]$ *with* $\vec{X}_i(\mathrm{t}) = [x_{i,1}(t), x_{i,2}(t), \ldots\ldots, x_{i,D}(t)]$ and each individual uniformly distributed in the range [$\vec{X}_{min}, \vec{X}_{max}$], where

$\vec{X}_{min} = \{x_{min,1}, x_{min,2}, \ldots, x_{min,D}\}$ and $\vec{X}_{max} = \{x_{max,1}, x_{max,2}, \ldots, x_{max,D}\}$ where $i = [1,2,\ldots., NP]$.

Step 2. WHILE stopping criterion is not satisfied

DO

FOR $i = 1$ to NP //do for each individual sequentially

Step 2.1 Mutation Step

Generate a donor vector $\vec{V}_i(t) = [v_{i,1}(t), v_{i,2}(t), \ldots\ldots, v_{i,D}(t)]$ corresponding to the i-th target vector $\vec{X}_i(t)$ via one of the mutation schemes of DE (Equation 2.3).

Step 2.2 Crossover Step

Generate a trial vector $\vec{U}_i(t) = \left[u_{i,1}(t), u_{i,2}(t), \ldots\ldots, u_{i,D}(t)\right]$ for the i-th target vector $\overrightarrow{X_1}(t)$ through exponential crossover (Equation 2.4) or binomial crossover (Equation 2.5)

Step 2.3 Selection Step

Evaluate the trial vector $\vec{U}_i(t)$

IF $f(\vec{U}_i(t)) \leq f(\vec{X}_i(t))$, THEN $\vec{X}_i(t+1) = \vec{U}_i(t)$,

$f(\vec{X}_i(t+1)) = f(\vec{U}_i(t))$

IF $f(\vec{U}_i(t)) < f(\vec{X}_{best}(t))$, THEN $\vec{X}_{best}(t) = \vec{U}_i(t)$,

$f(\vec{X}_{best}(t)) = f(\vec{U}_i(t))$

END IF

ELSE $\vec{X}_i(t+1) = \vec{X}_i(t)$, $f(\vec{X}_i(t+1)) = f(\vec{X}_i(t))$

END IF

END FOR

Increase the iteration count $t = t + 1$

END WHILE

The parameters used in the algorithm namely scale factor '*F*' and crossover rate '*CR*' should be submitted before in order to invoke the main computational part of the algorithm – the while loop. The terminating condition can be defined in two ways:

1) by a fixed number of iterations t_{max}, with a suitably large value of t_{max} depending upon the complexity of the objective function and alternatively,
2) when best fitness of the population does not change appreciably over successive iterations.

2.3 Automatic Clustering Differential Evolution

This section presents Automatic Clustering Differential Evolution (ACDE) algorithm and describes a simple modification of the classical differential evolution algorithm which is implemented in literature for improving the performance of ACDE algorithm. This chapter illustrates our implementation of ACDE algorithm and shows experimental results.

2.3.1 Introduction

Most of the material in this chapter is borrowed from [37]. It describes a Differential Evolution (DE) based algorithm for the automatic clustering of large unlabeled data sets. In contrast to most of the existing clustering techniques, the used algorithm requires no prior knowledge of the data to be classified. Rather, it determines the optimal number of clusters in the data 'on the run'.

Tremendous research effort has gone in the past few years to evolve the clusters in complex data sets through evolutionary computing techniques. However, little work has been taken up to determine the optimal number of clusters at the same time. Most of the existing clustering techniques, based on evolutionary algorithms, accept the number of clusters *k* as an input instead of determining the same on the run. Nevertheless, in many

practical situations, the appropriate number of groups in a previously unhandled data set may be unknown or impossible to determine even approximately. For example, while clustering a set of documents arising from the query to a search engine, the number of classes k changes for each set of documents that result from an interaction with the search engine. Also if the data set is described by high-dimensional feature vectors (which is very often the case), it may be practically impossible to visualize the data for tracking its number of clusters.

The objective of the research work described in this chapter is two-fold. Firstly, it aims at the automatic determination of the optimal number of clusters in any unlabeled data set. Secondly, it attempts to show that Differential Evolution (DE), with a modification of the chromosome representation scheme, can give very promising results if applied to the automatic clustering problem. DE is easy to implement and requires a negligible amount of parameter tuning to achieve considerably good search results. Authors of [38] changed the algorithm from its classical form to improve its convergence properties. In addition to that, they used a novel representation scheme for the search variables in order to determine the optimal number of clusters. They refer to the new algorithm as the ACDE (Automatic Clustering Differential Evolution) algorithm.

At this point, we would like to mention that the traditional approach of determining the optimal number of clusters in a data set is using some specially devised statistical-mathematical function (also known as a clustering validity index) to judge the quality of partitioning for a range of cluster numbers. A good clustering validity index is generally expected to provide global minima/maxima at the exact number of classes in the data set. Nonetheless, determination of the optimum cluster number using global validity measures is very expensive, since clustering has to be carried out for a variety

of possible cluster numbers. In the proposed evolutionary learning framework, a number of trial solutions come up with different cluster numbers as well as cluster center coordinates for the same data set. Correctness of each possible grouping is evaluated quantitatively with a global validity index (e.g. the CS or DB measure). Then, through a mechanism of mutation and natural selection, eventually the best solutions start dominating the population while the bad ones are eliminated. Ultimately, the evolution of solutions comes to a halt (i.e. converges), when the fittest solution represents a near-optimal partitioning of the data set with respect to the employed validity index. In this way, the optimal number of classes along with the accurate cluster center coordinates can be found out in an evolutionary search process. A downside to the proposed method is that, its performance depends heavily upon the choice of a suitable clustering validity index. An inefficient validity index may result into many false clusters (due to the over fitting of data) even when the actual number of clusters in the given data set may be very much tractable. However, with a judicious choice of the validity index, this algorithm can automate the entire process of clustering and yield near optimal partitioning of any previously unhandled data set in a reasonable amount of time. This is certainly a very desirable feature of a real-life pattern recognition task.

2.3.2 The DE-Based Automatic Clustering Algorithm

2.3.2.1 Vector Representation

In the proposed method, for n data points, each data point is d-dimensional, and for a user-specified maximum number of clusters $K_{\max}$, a chromosome is a vector of real numbers of dimension $K_{\max} + K_{\max} \times d$. The first $K_{\max}$ entries are positive floating-point numbers in [0, 1], each of which controls whether the corresponding cluster is to be activated (i.e. to be really used for classifying the data) or not. The remaining entries

are reserved for K_{max} cluster centers, each *is d*-dimensional. For example, the *i*-th vector is represented as:

$\vec{X}_i(t) =$

$T_{i,1}$	$T_{i,2}$	.....	$T_{i,K_{max}}$	$\vec{m}_{i,1}$	$\vec{m}_{i,2}$	.....	$\vec{m}_{i,K_{max}}$

The *j*-th cluster center in the *i*-th chromosome is active or selected for partitioning the associated data set if, $T_{i,j} > 0.5$. On the other hand, if, $T_{i,j} < 0.5$, the particular *j*-th cluster is inactive in the *i*-th vector in DE population. Thus the $T_{i,j}$ s behave like control genes (They are called activation thresholds) in the vector governing the selection of the active cluster centers.

The rule for selecting the actual number of clusters specified by one vector is:

IF $T_{i,j} > 0.5$ **THEN** the *j*-th cluster center $\vec{m}_{i,j}$ is **ACTIVE**

ELSE $\vec{m}_{i,j}$ is **INACTIVE**

Figure 2.6 shows an example of selecting three active centroids of five centroids. Each centroid has three values corresponding to the three dimensions of the space.

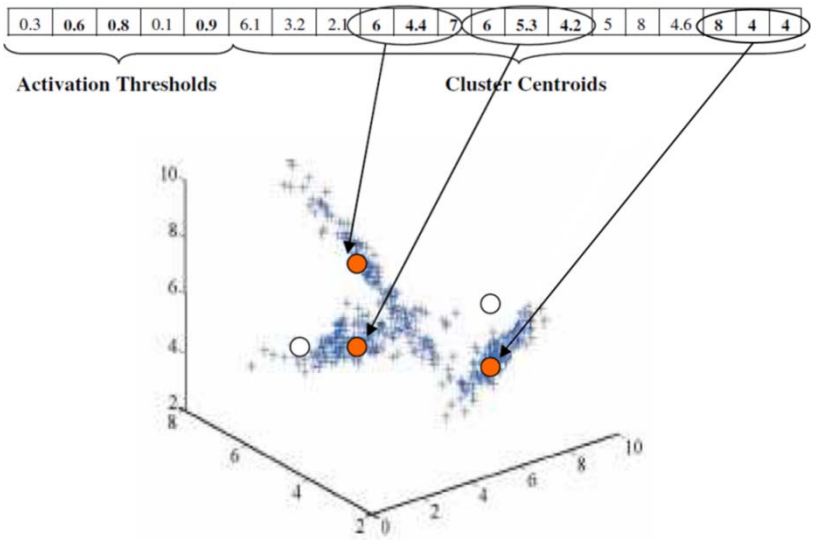


Figure 2.6: The chromosome encoding scheme in ACDE.
A total of five cluster centers have been encoded for a 3-dimensional data set.
Only the activated cluster centers have been shown as orange circles.
(adapted from Das, Abraham and Konar [37])

2.3.2.2 Designing the Fitness Function

One advantage of the ACDE algorithm is that it can use any suitable validity index as its fitness function. After experimenting with a number of validity indices (a brief review of which can be found in section 1.1.2 of chapter 1), we selected the CS measure as the basis of our fitness function, as CS measure deals with clusters of different densities and/or size more efficiently than several other existing validity indices. Before presenting some results from these experiments that establish the superiority of CS measure, we first redefine the CS measure below.

Let the centroid of a cluster be computed by averaging the data vectors belonging to that cluster using the formula,

$$\vec{\mathrm{m}}_i = \frac{1}{\mathrm{N}_i} \sum_{x_j \in C_i}^{k} \vec{\mathrm{Z}}_i \tag{2.7}$$

A distance metric between any two data points $\vec{\mathrm{X}}_\mathrm{i}$ and $\vec{\mathrm{X}}_\mathrm{j}$ is denoted by $\mathrm{d}(\vec{\mathrm{X}}_\mathrm{i}, \vec{\mathrm{X}}_\mathrm{j})$. Then the CS measure can be defined as,

$$CS(k) = \frac{\sum_{i=1}^{k} [\frac{1}{|C_i|} \sum_{\vec{X}_i \in C_i} \max_{\vec{X}_q \in C_i} \{d(\vec{Z}_i, \vec{Z}_q)\}]}{\sum_{i=1}^{k} \min_{j \in k, j \neq i} \{d(\vec{m}_i, \vec{m}_j)\}} \tag{2.8}$$

Note that the above measure is a function of the ratio of the sums of within cluster scatter to between-cluster separation and has the same basic rationale as the DB and the DI measures. That is, they are to seek clusters that have minimum within-cluster scatter (i.e. compact) and maximum between-cluster separation (i.e. well-separated). The numerator of (3.2) basically uses the largest distance between two data points lying in the same cluster to measure the scatter volume. On the other hand the denominator computes the average distance between cluster centers.

Authors of [37] presented examples on three hand-crafted data sets to illustrate the effectiveness of the CS measure in handling clusters of different geometric shapes,

densities and sizes over some well-known state-of-the-art validity indices found in literature. For comparison they used the following validity indices: the Dunn's Measure (DI), the Davies-Bouldin's measure (DB) with the parameters $q = t = 2$, the Bezdek's partition coefficient (PC) [39], Bezdek's classification entropy (CE) [40], and the Xie-Beni's index (XB) [41]. Please note that for the DI or PC validity measure, the largest value indicates a valid optimal partition. On the contrary, for the DB, CE, XB, or CS validity measures, the smallest value indicates a valid optimal partition.

The data sets were clustered with either k-means or Gustafson-Kessel (GK) [42] algorithm at each cluster number k for $k = 2$ to $k = 10$. Since both the algorithms are sensitive to initialization, during the clustering procedures authors of [37] have tried different initializations to cluster the data sets for each cluster number k. Then for each k, the clustering result happened with the highest frequency was chosen to be the clustering result for the cluster number k to be validated by the validity measures. Then all validity measures are computed from the same clustering results.

2.3.2.3 Avoiding Erroneous Vectors

There is a possibility that in our scheme, during computation of the CS and/or DB measures, a division by zero may be encountered. This may occur when one of the selected cluster centers in a DE-vector is outside the boundary of distributions of the data set. To avoid this problem authors of [37] checked to see if any cluster has fewer than two data points in it. If so, the cluster center positions of this special chromosome are re-initialized by an average computation. They put n/k data points for every individual cluster center, such that a data point goes with a center that is nearest to it.

2.3.2.4 Modification of the Classical DE

After performing a series of empirical experiments, authors of [37] proposed two parameter tuning strategies for DE in order to improve its convergence behaviour over

the clustering fitness landscape. In the original DE the difference vector $(\vec{X}_{r_2^i}(t) - \vec{X}_{r_3^i}(t))$ is scaled by a constant factor '*F*'. The usual choice for this control parameter is a number between 0.4 and 1. Authors proposed to vary this scale factor in a random manner in the range (0.5, 1) by using the following relation:

$$F = 0.5 \cdot (1 + rand(0,1)) , \tag{2.9}$$

where rand (0, 1) is a uniformly distributed random number within the range [0,1].The mean value of the scale factor is 0.75. This allows for stochastic variations in the amplification of the difference vector and thus helps retain population diversity as the search progresses. The authors of [37] have already shown that the DERANDSF (DE with Random Scale Factor) can meet or beat the classical DE. In addition to that, here they also decrease the crossover rate *CR* linearly with time from CR_{max} = 1.0 to CR_{min} = 0.5. If *CR* = 1.0, it means that all components of the parent vector are replaced by the difference vector operator according to Equation 2.5. But at the later stages of the optimizing process, if *CR* be decreased, more components of the parent vector are then inherited by the offspring. Such a tuning of *CR* helps to explore the search space exhaustively at the beginning, but adjust the movements of trial solutions finely during the later stages of search, so that they can explore the interior of a relatively small space in which the suspected global optimum lies.

The time-variation of *CR* may be expressed in the form of the following Equation,

$$CR = CR_{min} + (CR_{max} - CR_{min}) \cdot \left(\frac{MAXIT - iter}{MAXIT}\right), \tag{2.10}$$

where CR_{max} and CR_{min} are the maximum and minimum values of Crossover Rate *CR, iter* is the current iteration number and MAXIT is the maximum number of allowable iterations.

2.3.2.5 Pseudo-code of the ACDE Algorithm

The pseudo code of the complete ACDE algorithm is presented below.

Step 1: Initialize each search variable vector in DE to contain *k* number of randomly selected cluster centers and *k* (randomly chosen) activation thresholds in [0, 1].

Step 2: Find out the active cluster centers in each chromosome with the help of the rule described in section 2.3.2.1.

Step 3: For *iter* =1to *MAXITER* do // *MAXITER is maximum number of iterations*

1) For each data vector $\vec{Z}_p$, calculate its distance metric $d(\vec{Z}_i, \vec{m}_{i,j})$ from all active cluster centers of the *i*-th DE-vector $\vec{X}_i$.
2) Assign $\vec{X}_i$ to that particular cluster center $\vec{m}_{i,j}$ where $d(\vec{Z}_p, \vec{m}_{i,j}) = min_{\forall b \in \{1,2,\ldots,k\}}\{d(\vec{Z}_p, \vec{m}_{i,b})\}$
3) Check if the number of data points belonging to any cluster center $\vec{m}_{i,j}$ is less than 2. If so, update the cluster centers of the chromosome using the concept of average described earlier.
4) Change the population members according to the DE algorithm with modifications proposed in section 2.3.2.4. Use the fitness of the vectors to guide the evolution of the population.

Step 4: Report as the final solution the cluster centers and the partition obtained by the globally best vector (one yielding the highest value of the fitness function) at *iter* = *MAXITER* .

2.4 A New Point Symmetry-Based Distance Measure

This section presents a new point symmetry-based distance measure which is described in literature for improving point symmetric distance measure that is used to cluster overlapping and arbitrary shaped clusters with variable length.

Symmetry is considered a pre-attentive feature which enhances recognition and reconstruction of shapes and objects [43]. Almost every interesting area around us consists of some generalized form of symmetry. As symmetry is so common in the

natural world, it can be assumed that some kind of symmetry exists in the clusters also. Based on this, Su and Chou [44] have proposed a symmetry-based clustering technique. They assigned points to a particular cluster if they present a symmetrical structure with respect to the cluster center. But this work has some limitations.

S. Bandyopadhyay and S. Saha used a new point symmetry-based distance measure with an evolutionary clustering technique [45]. This algorithm is able to overcome some serious limitations of an earlier PS-distance proposed by Su and Chou. This algorithm is therefore able to detect both convex and non-convex clusters.

Bandyopadhyay and Saha [46] offered certain improvements of this point symmetric distance measure and used it to cluster overlapping and arbitrary shaped clusters. Let a point be $\bar{x}$. The symmetrical (reflected) point of $\bar{x}$ with respect to a particular center $\bar{c}$ is

$$\bar{x}^* = 2 \times \bar{c} - \bar{x} \tag{2.11}$$

Let knear unique nearest neighbours of $\bar{x}^*$ be at Euclidean distances of $d_i s$, i=1, 2,..., *knear*. Then the new point symmetry-based distance measure [44] is:

$$d_{ps}(\bar{x}, \bar{c}) = d_{sym}(\bar{x}, \bar{c}) \times d_e(\bar{x}, \bar{c}), \tag{2.12}$$

Where $\mathrm{d_{sym}}(\bar{\mathrm{x}}, \bar{\mathrm{c}}) = \frac{\sum_{i=1}^{\mathrm{knear}} \mathrm{d_i}}{\mathrm{knear}}$, it is a symmetry measure of $\bar{x}$ with respect to $\bar{c}$, and $d_e(\bar{x}, \bar{c})$ is the Euclidean distance between the point $\bar{\mathrm{x}}$ and $\bar{\mathrm{c}}$. We used this distance measure instead of Euclidean distance with K-means algorithm. We used Equation 2.12 by estimating $\mathrm{knear} = 2$.

In [46] authors used this measure. They used fitness function of that chromosome, fit, which is defined as the inverse of *M*, i.e,

$$\mathrm{fit} = \frac{1}{\mathrm{M}}, \tag{2.13}$$

Where *M*, is calculated as defined below:

M = 0

FOR k = 1 to K

FOR all data points $\bar{x}_i$, i = 1 to n and $\bar{x}_i \in k$th cluster, do

$M = M + d_{ps}(\bar{x}_i, \bar{c}_k)$.

END FOR

END FOR

This fitness function, fit, will be maximized by using ACDE (Note that there could be other ways of defining the fitness function).

In this part of the thesis, we concentrated our study on illustrating the performance of using different distance measures instead of Euclidean distance, so we will use k-means for simplicity, and after that we can generalize our ideas to use ACDE algorithm.

The most limiting aspect of the measures [44] is that it requires a prior specification of a parameter θ, based on whether assignment of points to clusters is done on the basis of the PS distance or the Euclidean distance. In order to compute the fitness of the chromosomes, in clustering with the PS-based distance measure, $\bar{x}_i, 1 \ll i \ll n$, is assigned to cluster k if and only if $d_{ps}(\bar{x}_i, \bar{c}_k) \leq d_{ps}(\bar{x}_i, \bar{c}_j)$, $j = 1, \ldots, K$, $j \neq k$ and $d_{ps}(\bar{x}_i, \bar{c}_k) / d_e(\bar{x}_i, \bar{c}_k) \leq \theta$. For $d_{ps}(\bar{x}_i, \bar{c}_k) / d_e(\bar{x}_i, \bar{c}_k) > \theta$ point $\bar{x}_i$ is assigned to some cluster m if and only if $d_{ps}(\bar{x}_i, \bar{c}_m) \leq d_{ps}(\bar{x}_i, \bar{c}_j), j = 1, \ldots, K$, $j \neq m$. In other words, point $\bar{x}_i$ is assigned to cluster k such that the PS-distance between $\bar{x}_i$ and center of cluster k is the minimum, and provided the total "symmetry" with respect to it is less than some threshold θ. Otherwise assignment is done based on the minimum Euclidean distance criterion. So the clustering performance is significantly affected by the choice of θ, and its best value is dependent on the data characteristics.

Su and Chou in [47] have chosen θ to be equal to 0.18. In [46] authors proposed to keep the value of θ equal to the maximum nearest neighbour distance among all the

points in the data set. Thus the computation of θ is automatic and does not require user intervention.

It is evident that the symmetrical distance computation is very time consuming because it involves the computation of the nearest neighbours. Authors of [46] described that computation of $d_{ps}(\bar{x}, \bar{c})$ is of complexity *O(nD)*, where *D* is the dimension of the data set and *n* is the total number of points present in the data set. Hence, for *K* clusters, the time complexity of computing PS-distance between all points to different clusters is $O(n^2KD)$. In order to reduce the computational complexity, an approximate nearest neighbour search using the *k*d-tree approach is adopted in their paper.

From aforementioned introduction, we can conclude the problems of using PS-distance as follows:

1) This measure is suitable only to classify clusters of symmetrical shape.
2) Using PS-distance for data clustering requires a prior specification of a parameter θ.
3) The clustering performance is significantly affected by the choice of θ, and its best value is dependent on the data characteristics.
4) The symmetrical distance computation is very time consuming because it involves the computation of the nearest neighbours.

In chapter 3, we present an improved PS-distance for distance measure. We present our approach to tackle previous problems by using *k*d-tree.

2.5 KD-Tree-Based Nearest Neighbor Computation

This section presents *k*d-tree which is the most important multidimensional structure for storing a finite set of data points from k-dimensional space. In addition, the section illustrates the usage of *k*d-tree. We use *k*d-tree for improving performance of clustering algorithms and developing a new effective clustering algorithm.

A K-dimensional tree, or *k*d-tree [48] is a space-partitioning data structure for organizing points in a K-dimensional space. The *k*d-tree is a top-down hierarchical scheme for partitioning data. Consider a set of n points, $(x_1...x_n)$ occupying an m dimensional space Each point x_i has associated with it m coordinates $(x_{i1}, x_{i2}, ..., x_{im})$. There exists a bounding box, or bucket, which contains all data points and whose extrema are defined by the maximum and minimum coordinate values of the data points in each dimension. The data is then partitioned into two sub-buckets by splitting the data along the longest dimension of the parent bucket. These partitioning processes may then be recursively repeated on each sub-bucket until a leaf bucket is created, at which point no further partitioning will be performed on that bucket. A leaf bucket is a bucket which fulfils a certain requirement, such as, it only contains one data point.

Kd-tree is the most important multidimensional structure for storing a finite set of data points from k-dimensional space. It decomposes a multidimensional space into hyper-rectangles. A *k*d-tree is a binary tree with both a dimension number and splitting value at each node. Each node corresponds to a hyper-rectangle. A hyper-rectangle is represented by an array of minimum coordinates and an array of maximum coordinates (e.g. in 2 dimensions (k = 2), (x_{min}, y_{min}) and (x_{max}, y_{max})). When searching for the nearest neighbour we need to know if a hyper-rectangle intersects with a hyper-sphere. The contents of each node are depicted in Table 2.1.

Table 2.1: The fields of *k*d-tree node

Field	***Description***
Type	Type of node tree (node or leaf)
Parent	The index of parent node in *k*d-tree
splitdim	The splitting dimension number
Splitval	The splitting value
left *k*d-tree	A *k*d-tree representing those points to the left of the splitting plane
right *k*d-tree	A *k*d-tree representing those points to the right of the splitting plane
Hyperrect	The coordinates of hyperrectangle
Numpoints	The number of points contained in hyperrectangle

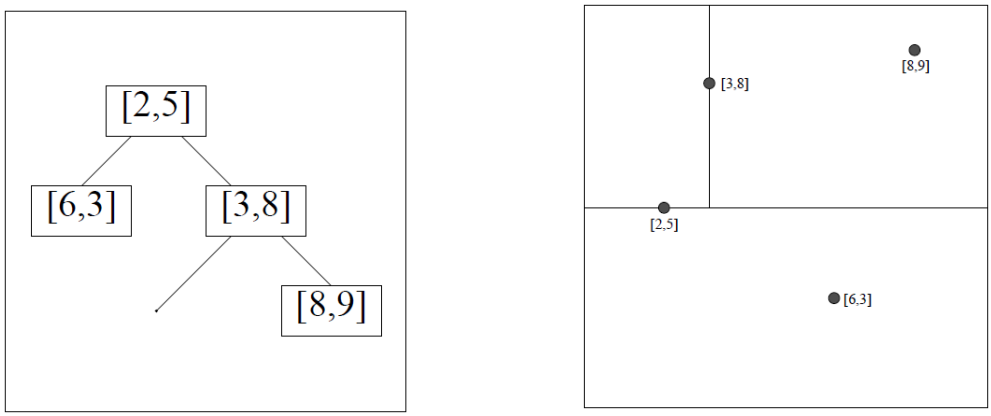
An interesting property of the kd-tree is that each bucket will contain roughly the same number of points. However, if the data in a bucket is more densely packed than some other bucket we would generally expect the volume of that densely packed bucket to be smaller.

Approximate Nearest Neighbour (ANN) is a library written in C++ [49], which supports data structures and algorithms for both exact and approximate nearest neighbour searching in arbitrarily high dimensions. The ANN library implements *k*d-tree data structure. In this thesis, we used ANN to find exact values of $d_i s$, in Equation 2.12 in an efficient way.

The function performing the *k*-nearest neighbor search in ANN is given a query point q, a nonnegative integer k, an array of point indices, nn_{idx}, and an array of distances, *dists*. Both arrays are assumed to contain at least k elements. This procedure computes the k nearest neighbours of q in the point set and stores the indices of the nearest neighbours in the array nn_{idx}. Here, k is set to be equal to *knear*. In this thesis, it is set to 2. In order to find PS-distance of a particular point $\bar{x}$ with respect to the center $\bar{c}$, we have to find the first *knear* nearest neighbours of $\bar{x}^*$ which is equal to $2 \times \bar{c} - \bar{x}$. Therefore, the query point q is set to be equal to $\bar{x}^*$. After getting the *knear* nearest neighbours of $\bar{x}^*$, the symmetrical distance of $\bar{x}$ with respect to a center $\bar{c}$ is calculated using Equation 2.12.

Each node splits the space into two subspaces according to the splitting dimension of the node, and the node's splitting value. Geometrically this represents a hyper-plane perpendicular to the direction specified by the splitting dimension. Figure 2.7 (a) demonstrates a *2*d-tree representation of the four data points (2,5), (3,8), (6,3), and (8,9). The root node (2,5) splits the plane in the y-axis into two subspaces along y=5. The point (3,8) lies in the upper subspace, that is {(x,y) | y>5} and splits along the x=3

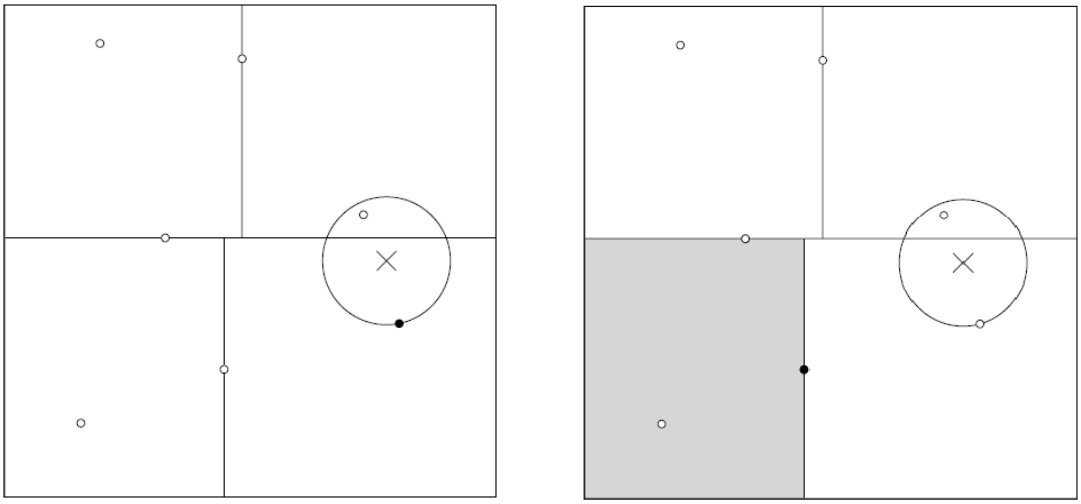
plane. And so is in the right sub-tree. Figure 2.7 (b) shows how the nodes partition the plane.



(a) **(b)**

Figure 2.7: (a) A 2d-tree of four elements. (b) How the tree splits up the x,y plane. (adapted from Moore [47])

Searching for a point in the data set that is represented in a *k*d-tree is accomplished in a traversal of the tree from root to leaf which is of complexity O(log(n)) (if there are *n* data points). The first approximation is initially found at the leaf node which contains the target point. In Figure 2.8 (a) the target is marked × and the leaf node of the region containing the target is colored black. As is exemplified in this case, this first approximation is not necessarily the nearest neighbour, but at least we know any potential nearer neighbour must be closer, and so it must be within the circle centerd on × and passing through the leaf node. We now back up to the parent of the current node. In Figure 2.8 (b) this parent is the black node. We compute whether it is possible for a closer solution to that so far found to exist in this parent's other child. Here it is not possible, because the circle does not intersect with the (shaded) space occupied by the parent's other child. If no closer neighbour can exist in the other child, the algorithm can immediately move up a further level, else it must recursively explore the other child. In this example, the next parent which is checked will need to be explored, because the area it covers (i.e. everywhere north of the central horizontal line) does intersect with the best circle so far.



(a) **(b)**

Figure 2.8: Searching for a point in the data set
(a) The leaf node containing the target.
(b) The parent of the closest found so far.
(adapted from Moore [47])

Chapter 3

Methodology

In this chapter we illustrate our original work for improving efficiency of classification and tackling the problem which is presented in chapter 1. This chapter illustrates usage of *k*d-tree for developing an improved PS-Based distance measure. This chapter describes our contribution for improving efficiency of k-means. We called the proposed algorithm as a novel effective k-means algorithm. We used an improved PS-Based distance measure for developing the proposed algorithm. This chapter illustrates our proposed original algorithm for classifying complex data sets. We called the proposed algorithm a Dynamic Linkage Clustering using KD-Tree (DLCKDT). We used selected nodes from *k*d-tree to develop this algorithm. Finally, this chapter illustrates our contribution for improving efficiency of ACDE to classify complex data sets automatically.

3.1 Selecting Dense Points

We proposed to use *k*d-tree for checking the connectivity of each data point with its cluster. We used *k*d-tree to determine the collections of dense regions in dimensional space. Using *k*d-tree will reduce computation cost and its results will be better than using other methods that are presented in literature for determining the dense regions. We selected some points of *k*d-tree which denote the dense centers of dense regions in the data set. We called these points as Dense Points (DPs).

Selecting leaf nodes as DPs is not suitable because each leaf node in *k*d-tree is a bucket contains only one data point and will cause selecting all data points in the data set. So selecting leaf nodes as DPs will not form dense centers of dense regions in the data set.

Selecting parent of leaf nodes in *k*d-tree as DPs is not suitable also because parent of leaf node contains only two data points (two leaf nodes) and will cause sensitivity to noise (outlying data points) in the data set.

Depending on the previous analysis, we selected DPs by searching for leaf nodes in the *k*d-tree and then finding the grandparent of the leaf nodes. Grandparent of the leaf nodes contains more than two data points, so selecting them as DPs will reduce sensitivity to noise in the data set and will form small number of centers to denote to dense regions in the data set for reducing processing time in classification. Figure 3.1 shows the structure of *k*d-tree and the position of DPs. We note that the DPs (which are shown as shaded nodes) denote to 2^{nd} and 3^{rd} levels of the *k*d-tree. We note that more than two nodes fork from nodes of DPs, this indicates to DPs contains more than two data points.

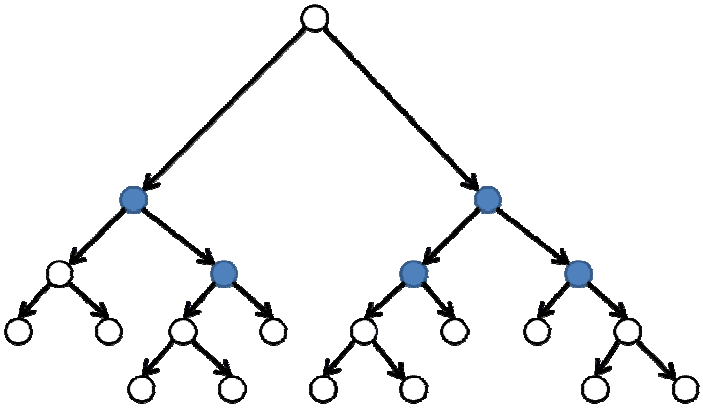


Figure 3.1: Selected Dense Points

Figure 3.2 (a) shows the position of DPs on synthetic data set which has one cluster. We note that these points form almost the shape of cluster with little number of points. We note that DPs distributed on the whole data set and they exist in dense regions of data points. Figure 3.2 (b) shows the rectangular regions which are covered by DPs of *k*d-tree. We note that these regions almost cover all the data set, so we can conclude that DPs correspond to the dense regions of the data set.

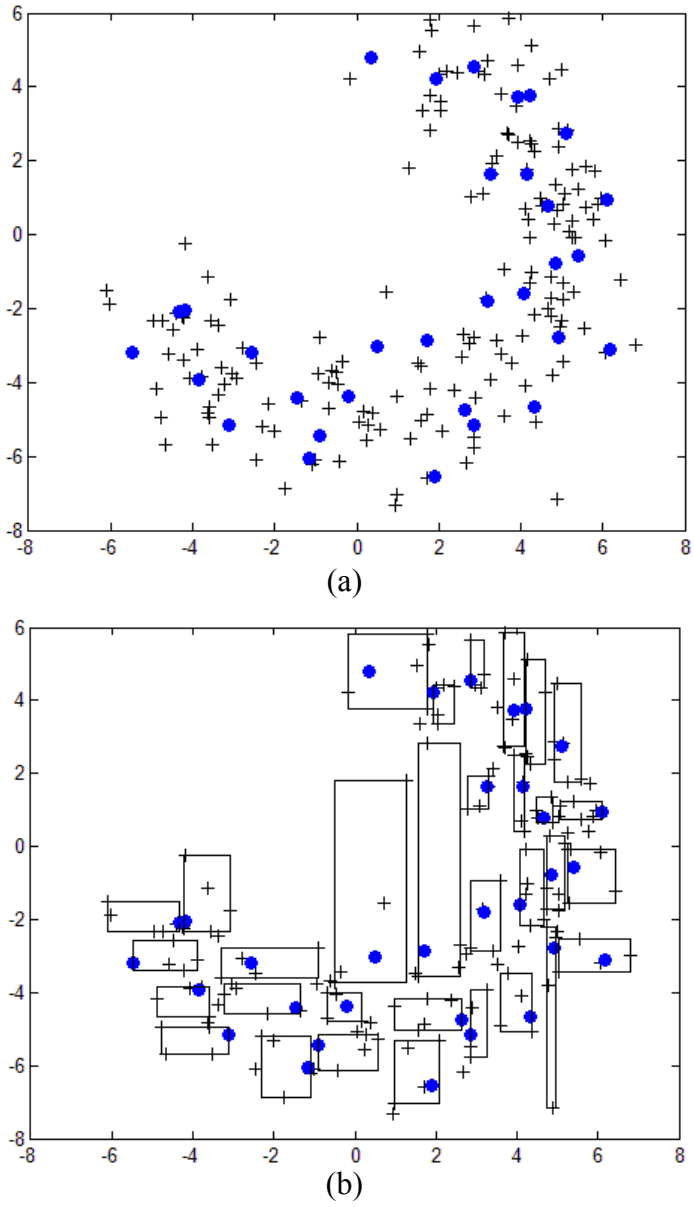


Figure 3.2: (a) DPs of *k*d-tree. (b) Rectangular regions covered by DPs.

Using upper levels in *k*d-tree (more than the 3rd level) for selecting DPs will decrease number of DPs for representing dense regions, but in the same time rectangular regions will be larger and will cover some parts of space which are empty from data points. Figure 3.3 shows selecting nodes from various levels in *k*d-tree and showing the corresponding of these nodes to data points in a data set of one cluster.

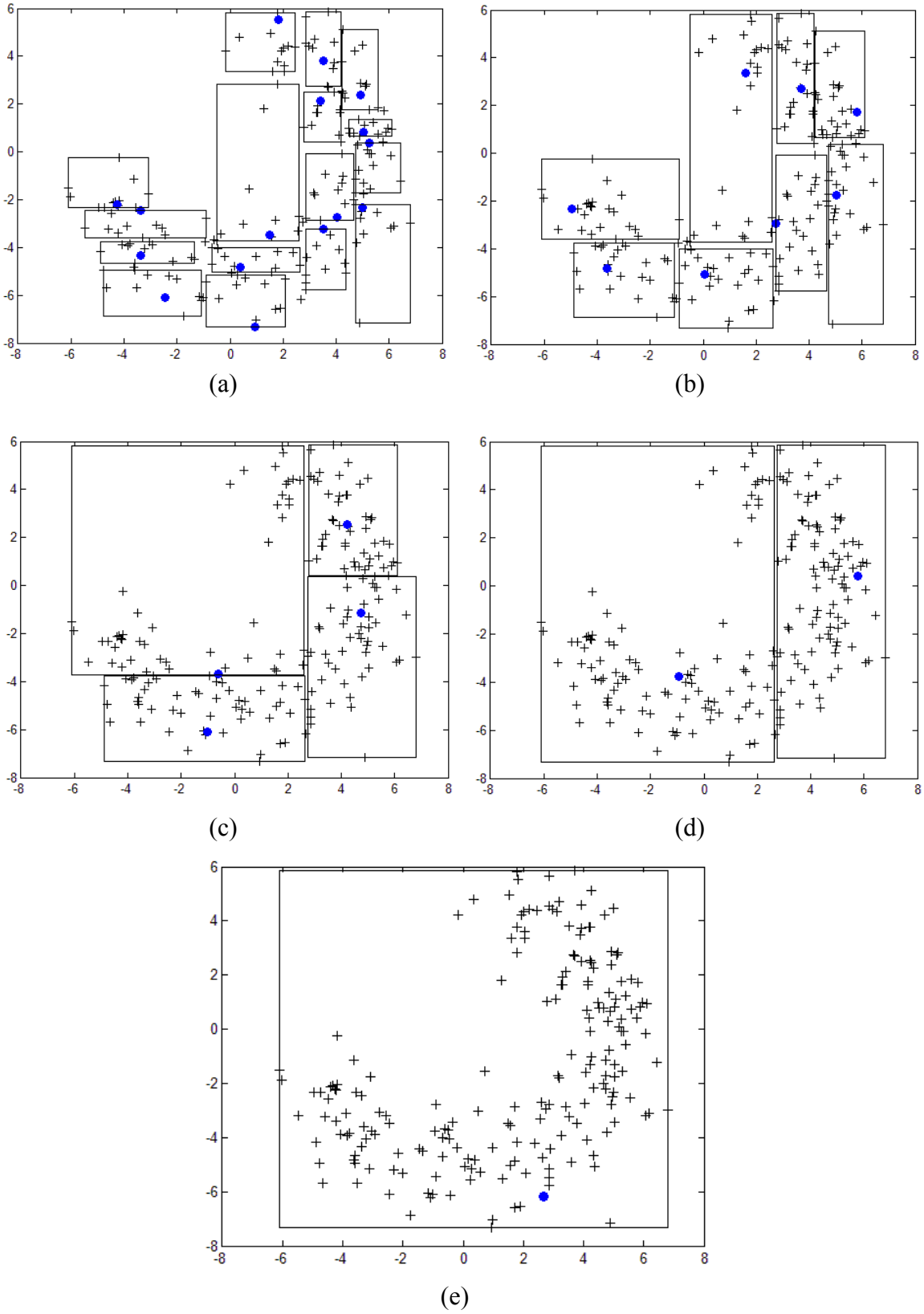


Figure 3.3: Selecting nodes from various levels in *k*d-tree upper than the 3rd level (a) Nodes of the 4th level. (b) Nodes of the 5th level. (c) Nodes of the 6th level. (d) Nodes of the 7th level. (e) Nodes of the 8th level.

We note from Figure 3.3 (a) that the number of nodes which denote to dense regions are smaller than number of DPs which are shown in Figure 3.2(b) but the size of rectangular regions are increased, this caused covering empty regions of data points. These effects are increased gradualness from Figure 3.3 (c) to Figure 3.3 (e). Figure 3.3 (e) shows that only one node represents all data points in cluster and covers empty space outside the cluster. So we inferred, if we use upper levels for representing DPs then the shape which is formed by rectangular regions for covering the cluster will be rough, and many data points will be selected from other clusters if there are overlapped clusters in the data set.

We can conclude that selecting the grandparent of the leaf nodes in *k*d-tree for representing DPs is the best choice to determine the collections of dense regions in dimensional space. We used this concept for selecting DPs in our experiments for increasing performance of classifying clusters in complex data set.

Selecting DPs have many advantages. First of all, using DPs reduces the number of data points used for classification, so this method will reduce time complexity. From other side, using DPs will reduce the effect of noise (outlying data points) on classification. Figure 3.4 shows position of DPs (plotted as circles) in data set having one cluster with outlying data points. We note that the outlying data points, which denoted as + symbols in the four corners of the Figure, is not selected as DPs. We note also that all DPs are concentrated in spaces which have density of data points.

So we can use DPs for checking density reachability of each data point with its cluster. Using DPs will be effective for classifying complex data sets which have overlapped and arbitrary shaped clusters.

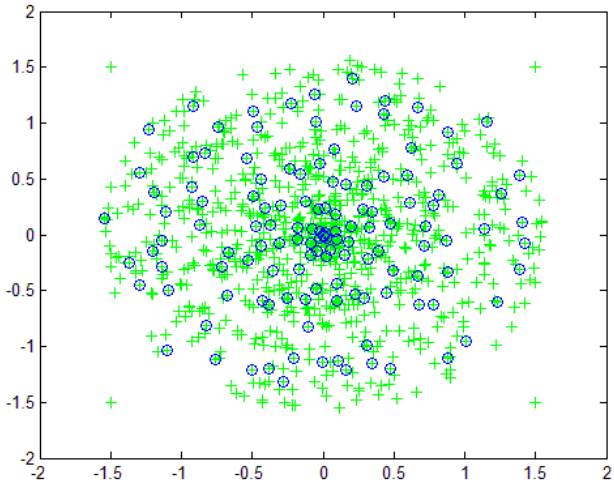


Figure 3.4: Selecting DPs form data set having noise

Next, we will use DPs for improving efficacy of some clustering algorithms and developing a new effective clustering algorithm.

3.2 Improved PS-Based Distance Measure

In this section, we illustrate our method for enhancing PS-Based distance measure by using *k*d-tree. This enhancement is used for classification and overcoming pervious limitations which were presented in section 2.4.

When we used k-means with Euclidean distance to calculate distances between data points and centroids and then classify data points to the nearest centroid, we noted that the results of classification were bad with using complex data sets. Of course, the results will be better when we use PS-based distance measure, but the clusters also were not classified correctly.

We dissected this problem and deduced that we must include the density of points with the distance measure to classify this type of data sets. When using k-mean with Euclidean distance and PS-based distance measure, then all points are assigned to the nearest cluster despite of some of them are connected to other clusters. So if we study connectivity of these pointes with nearest clusters then we will tackle the problem and classify all clusters correctly.

Our proposed method uses DPs of *k*d-tree for determining the connectivity instead of using other methods which are presented in literature like DBSCAN. We developed a simple algorithm for selecting points which are classified incorrectly when using Euclidean distance with k-means. This algorithm is as follows:

FOR each data point *X* do

Find the nearest 2 centroids of *X* (i.e. C_1, C_2)

Find the nearest 2 *DPs* of *X* (i.e. DP_1, DP_2)

IF $d_{ps}(DP_1, C_2) < d_{ps}(DP_1, C_1)$ OR $d_{ps}(DP_2, C_2) < d_{ps}(DP_2, C_1)$

Select *X*

END IF

END FOR

In the algorithm above, d_{ps} is the PS-distance measure which is calculated by Equation 2.12. We used PS-distance measure instead of Euclidean distance to improve clustering performance. We used value of 2 for *knear* in Equation 2.12, because this value gives good results when merging connectivity with distance measure. Of course, using value greater than 2 for *knear* will increase accuracy of classification, but it will increase time complexity. Our approach merges connectivity with distance measure, so checking connectivity of each data point with its cluster will increase accuracy and using value greater than 2 for knear is necessary.

The work described in this thesis concerns crisp (hard) clustering algorithms only. So each pattern (data point) will assign to one and only one cluster. So we concerns classification to the nearest 2 clusters of each pattern. If the data point is not followed to the nearest cluster (nearest cenriod), then it will be connected to the 2nd nearest cluster.

The algorithm above finds the nearest 2 clusters (centroids) of each pattern, and then finds the nearest 2 DPs of this pattern. After that, the algorithm checks the

connectivity of this pattern with the 2nd nearest cluster by checking if any point of nearest 2 DPs is followed to that cluster or not. If any point of nearest 2 DPs is followed to the 2nd nearest cluster then we sure that the pattern follows to the 2nd nearest cluster despite of it nears to the 1st nearest cluster.

We used the nearest 2 DPs to check connectivity of each pattern to its cluster. Of course using more number of nearest DPs for checking connectivity will increase accuracy of classification, but will increase time complexity. Using two DPs is enough for checking connectivity of each data point with its cluster, because our study depends on calculating distances to the nearest clusters. Catching only one DP followed to the 2nd nearest cluster is enough to decide that the data point is connected to the 2nd nearest cluster, and for increasing accuracy of classification we used 2 DPs instead of one.

Depending on previous concepts we developed a new distance measure for improving the performance of PS-distance measure as follows:

$$d_{IPS}(\bar{x}, \bar{c}) = \frac{d_{ps}(DP_1, \bar{c})}{d_e(DP_1, \bar{c}_2)} + \frac{d_{ps}(DP_2, \bar{c})}{d_e(DP_2, \bar{c}_2)} \tag{3.1}$$

Where d_{ps} denotes the PS-based distance measure which is calculated by Equation 2.12, and d_e denotes the Euclidean distance. DP_1 and DP_2 denote the first and second DPs of *k*d-tree which are selected as demonstrated in the previous section. $\bar{c}_2$ denotes the second nearest centroid of $\bar{x}$.

First part of Equation 3.1 ($\frac{\mathrm{d_{ps}(DP_1,\bar{c})}}{\mathrm{d_e(DP_1,\bar{c}_2)}}$) checks connectivity of 1st nearest DP ($\mathrm{DP_1}$) to the 2nd nearest centroid of $\bar{x}$ ($\bar{c}_2$). If $\mathrm{d_e(DP_1,\bar{c}_2) > d_{ps}(DP_1,\bar{c})}$ then the value of $\frac{\mathrm{d_{ps}(DP_1,\bar{c})}}{\mathrm{d_e(DP_1,\bar{c}_2)}}$ will be small, this will cause attaching pattern $\bar{x}$ to cluster $\bar{c}$ because classification of data points depends on assigning each data point $\bar{x}$ to cluster which has the smallest distance $d_{IPS}(\bar{x}, \bar{c})$. If $\mathrm{d_e(DP_1,\bar{c}_2) < d_{ps}(DP_1,\bar{c})}$ then the value of $\frac{\mathrm{d_{ps}(DP_1,\bar{c})}}{\mathrm{d_e(DP_1,\bar{c}_2)}}$

will be big, this will cause attaching pattern $\bar{x}$ to cluster $\bar{c}_2$ because the distance between $\bar{x}$ and $\bar{c}$ ($d_{IPS}(\bar{x}, \bar{c})$) will be bigger than distance between $\bar{x}$ and $\bar{c}_2$ ($d_{IPS}(\bar{x}, \bar{c}_2)$) when calculating all distances between pattern $\bar{x}$ and all centroids in the data set. Second part of Equation 3.1 ($\frac{\mathrm{d_{ps}(DP_2, \bar{c})}}{\mathrm{d_e(DP_2, \bar{c}_2)}}$) checks connectivity of 2nd nearest DP ($\mathrm{DP_2}$) to the 2nd nearest centroid of $\bar{x}$ ($\bar{c}_2$) same as checking connectivity of $\mathrm{DP_1}$ in first part. The two parts of Equation 3.1 are added corresponding to OR operation in the above algorithm.

Next, we use Equation 3.1 to develop a novel effective K-means algorithm for classifying complex data sets. We use it also for improving the performance of ACDE algorithm for classifying more complex data sets.

3.3 A Novel Effective K-Means Algorithm

This section describes our contribution for improving efficiency of k-means. We called the proposed algorithm a novel effective k-means algorithm. We used an improved PS-Based distance measure for developing the proposed algorithm.

We present the pseudo code of a novel Effective K-means algorithm that we have developed as follows:

1. Initialize K center locations (C_1, ..., C_K).
2. Select DPs of kd-tree.
3. FOR each cluster center C_j do

   FOR each data point X_i do

   Calculate $d_{IPS}(X_i, C_j)$ by using Equation 3.1.

   END FOR

   END FOR
4. Assign each data point X_i to its cluster center C_j by selecting the minimum distance of $d_{IPS}(X_i, C_j)$.
5. Update each cluster center C_j as the mean of all X_i that have been assigned to it.

6. Calculate $D = \sum_{i=1}^{n} \left[\min_{j=(1\ldots K)} d_{IPS}(X_i, C_j)\right]^2$.
7. If the value of D has converged, then return (C_1, ..., C_K); else go to Step 3.

This algorithm has three main advantages:

1) It is very easy to implement.
2) It does not use additional parameters like other algorithms which are proposed in literature for improving the efficiency of K-means algorithm. Most of parameters which are used by other algorithms are sensitive to the performance of classification.
3) Its performance is better than the performance of K-means. It classified more data sets which were classified incorrectly by K-means algorithm.

However this algorithm suffers from the following disadvantages:

1) The user has to specify the number of classes in advance.
2) Processing time is increased compared to k-means using Euclidean distance.
3) The algorithm uses a greedy approach and is heavily dependent on the initial conditions. This often leads the results to converge to sub-optimal solutions.

We propose to use [25] in step 1 of our novel algorithm to eliminate the dependency on the initial conditions.

3.4 Dynamic Linkage Clustering using KD-Tree

In this section we develop a new clustering algorithm depending on the *k*d-tree. We called the proposed algorithm as a Dynamic Linkage Clustering using KD-Tree (DLCKDT). We used selected nodes from *k*d-tree to develop this algorithm. Our goal for developing this algorithm is classifying complex data sets more accurately than other algorithms which are presented in the literature.

The new developed clustering algorithm depends on the *k*d-tree. It consists of three phases: The first phase selects DPs of *k*d-tree for using them as initial seeds. The

second phase assigns each data set to its nearest DP. So the output of this phase is a collection of small clusters whose number is equal to the number of DPs. The last phase merges the small clusters (output of the second phase). During the third phase, each iteration consists of merging the nearest two clusters. This phase continues until the number of clusters is equal to the value which is specified by the user in advance (denoted by the number of classes).

The pseudo code of our novel clustering algorithm using *k*d-tree is:

(1) Input the number of clusters K.

(2) Select DPs of *k*d-tree (DP_1,...,DP_n).

(3) Assign each data point X_i to its nearest DP_j to form initial clusters of data points.

(4) Merge every two adjacent clusters of step 3.

(5) Find the nearest two clusters and merge them.

(6) If the number of merged clusters N > K, then go to Step 5; else return the merged clusters.

Step 2 is the first phase in the algorithm for selecting DPs of *k*d-tree. Step 3 is the second phase. It creates a large number of small groups which are used as initial clusters. Steps 4, 5, and 6 form the final phase. This phase merges clusters which are generated in step 3. The merging is terminated when the number of merged clusters is equal to the value of K, where K is an input parameter which is defined as the target number of clusters. Step 4 decreases to half the number of selected clusters which are generated by step 3. This step is used for reducing time complexity.

We used the nearest neighbor distance [50] to calculate the distance between each two clusters C1 and C2 which is denoted by $D(C_1, C_2)$ where:

$$D(C_1, C_2) = \min_{1 \le i \le r, 1 \le j \le s} d(y_i, z_i) \quad (3.2)$$

Figure 3.5 gives an example of the nearest neighbor distance in the two-dimensional case. We note that $D(C_1, C_2)$ is the Euclidean distance between the nearest points between clusters C_1 and C_2. For calculating this equation we need to calculate all distances between each point in C_1 and C_2 and then finding the minimum distance. We can find the nearest two clusters for step 5 of the algorithm by calculating the minimum distance between all clusters.

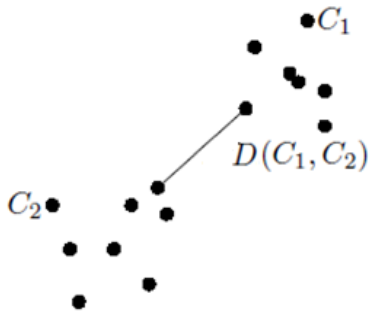


Figure 3.5: Nearest neighbour distance between two clusters.

The Dynamic Linkage Clustering using KD-Tree (DLCKDT) has three main advantages:

1) It is easy to implement.
2) The algorithm does not depend on the initial conditions. This forces the algorithm to converge to global solution.
3) It doesn't reliance on a priori knowledge and user defined parameters.
4) It can classify very complex data sets which cannot be classified by our novel effective k-means algorithm.

However the proposed algorithm suffers from the following disadvantages:

1) The user has to specify the number of classes in advance.
2) The elapsed time is increased when comparing it with our novel effective k-means algorithm.

3.5 Improved ACDE Algorithm

In this section we propose some modifications and enhancements to ACDE algorithm to improve its performance and classify more complex data sets.

We propose using median value instead of mean values (Equation 1.2) for calculating $\vec{m}_{i,j}$ of the chromosome, so we used medoids instead of centroids to represent each cluster. This modification decreases the sensitivity of noise (outlying data points). We also suggest initializing the parameter vectors by selecting the values randomly form the data points of the data set. These two modifications decrease the time for searching for the global solution. We also propose using Equation 3.1 to calculate the improved PS-based distance between the medoids and the data points instead of using Euclidean distance. So our improvements to the ACDE algorithm are as follows:

Pseudo-code of the improved ACDE Algorithm (Phase I)

1. Initialize each search variable vector in DE to contain a number , *k* , of randomly selected data points of the data set, and select randomly *k* activation thresholds in [0, 1].

2. Find out the active cluster medoid in each chromosome with the help of the rule described in section 2.3.2.1.

3. For *iter* =1to *MAXITER* do

3.1 For each data vector $\vec{Z}_p$, calculate its distance metric $d_{IPS}(\vec{Z}_i, \vec{m}_{i,j})$ from all active cluster centers of the *i*-th DE-vector $\vec{X}_i$.

3.2 Assign $\vec{X}_i$ to that particular cluster medoid $\vec{m}_{i,j}$ where $d_{IPS}(\vec{Z}_p, \vec{m}_{i,j}) = min_{\forall b \in \{1,2,\dots,k\}}\{d_{IPS}(\vec{Z}_p, \vec{m}_{i,b})\}$

3.3 Check if the number of data points belonging to any cluster medoid $\vec{m}_{i,j}$ is less than 2. If so, update the cluster medoid of the chromosome using

the concept of average described earlier by calculating median value instead of mean value.

3.4 Change the population members according to the DE algorithm with modifications proposed in Section 2.3.2.4. Use the fitness of the vectors to guide the evolution of the population.

4. Report as the final solution the cluster medoids and the partition obtained by the globally best vector (one yielding the highest value of the fitness function) at *iter* = *MAXITER*.

We calculated the fitness function by finding the maximum value of the summation of distances between clusters. We calculated the distance between every two clusters by using Equation 3.2. We can also calculate the distance between two clusters by calculating the Euclidean distance between its mediods. The used fitness function is as follows:

$$fit = \sum_{1 \le i \le k, 1 \le j \le k} D(C_i, C_j) \tag{3.3}$$

Where k is the number of selected clusters which is equal to the number of active mediods that is calculated by using rule which is described in section 2.3.2.1.

We tested the enhanced algorithm with many data sets but some clusters of large size did not classify correctly. This fault is consisted because Equation 3.3 is not suitable for classifying complex data set. Developing suitable fitness function to classify complex data sets by ACDE is not easy work. For tackling this problem we created a connected graph to connect the sub-clusters which classified incorrectly by using Equation 3.3.

We developed a connected graph by applying steps 2, 3, and 4 of the DLCKDT algorithm (see section 3.4). Firstly, we selected DPs of *k*d-tree. Secondly, we assigned

each data point to its nearest DP. Finally, we connected every two adjacent DPs clusters by connecting the nearest two data points of the adjacent DPs clusters.

To implement connected graph by software, we can create a matrix and fills it with all connected points in the connected graph. This matrix is created once in the beginning of program and then it can be used for many times in the same program. This will save time consuming when running the program. Size of matrix depends on number of points in the data set and the number of dimensions.

Figure 3.6 shows example of connected graph which is created on a data set has three clusters (marked by dashed circles) in two dimensions. We note that most points in each cluster are connected in the connected graph, so we can connect them to form one cluster if they are classified to more than one sub-cluster after using Equation 3.3 with phase I of enhanced ACDE.

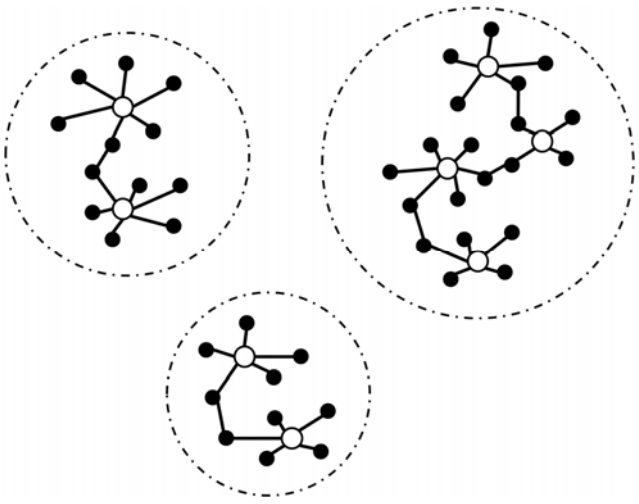


Figure 3.6: The Connected Graph

Figure 3.6 shows that, the connected graph is created by connecting each data point (mark as shaded circles) to nearest DP (mark as unshaded circles) to form small groups of points (large number of small clusters), then every two adjacent groups are connected by connecting the nearest 2 points between every two adjacent groups. We note that every cluster has its connected graph and all connected graphs are separated.

So every cluster is modelled by connected graph. Based on this, we can use the connected graph to classify all clusters correctly.

So the improved ACDE has two phases. The first phase is as described previously and the second phase merges the sub-clusters by using the connected graph. We tested the connection between every two adjacent clusters (C_1, C_2) by using the following pseudo-code:

1. FOR each data point X_i in cluster C_1 do

    1.1 FOR each data point X_j in cluster C_2 do

        1.1.1 Find X_i *and* X_j in the connected graph.

        1.1.2 IF X_i *and* X_j are connected, then

            1.1.2.1 Merge C_1 and C_2.

            1.1.2.2 Update the medoids of each cluster by calculating the median value of its points.

        END IF

    END FOR

END FOR

We tested the improved ACDE algorithm with many data sets. It classified all the data sets which were tested by the novel effective k-means algorithm and the novel clustering algorithm using *k*d-tree.

The improved ACDE has the following advantages:

1) Its performance is better than the novel effective k-means algorithm and the novel clustering algorithm using *k*d-tree.
2) It is able to automatically find the optimal number of clusters (i.e., the number of clusters does not have to be known in advance).

3) It is not sensitive to the initial conditions. This forces the algorithm to converge to global solution.

However the improved ACDE suffers from one main disadvantage. It increased the elapsed time compared to the Novel Effective K-Means Algorithm and the novel clustering algorithm using *k*d-tree.

Chapter 4

Experimental Results

4.1 Performance of Automatic Clustering Differential Evolution

We implemented ACDE algorithm by using MATLAB 7.3 (R2006b) for illustrating its performance. We tested this algorithm with synthetic data set which has four clusters that are well separated. Figure 4.1 (a) shows the data set and the trial locations of maximum number of cluster centers. In this experiment we initialized each search variable vector in DE to contain 7 numbers of randomly selected cluster centers and 7 (randomly chosen) activation thresholds in [0, 1].

Figure 4.1 (b) shows the result of clustering by using ACDE. We note that this algorithm determined the optimal number of clusters and labelled them correctly. Points corresponding to each cluster are marked in specific symbols.

This result demonstrates that ACDE is a powerful clustering algorithm. It clustered unlabeled data set in automatic way and without prior knowledge of the data. But there are some defects that will arise when testing other forms of data sets. Next we will investigate fundamental problems.

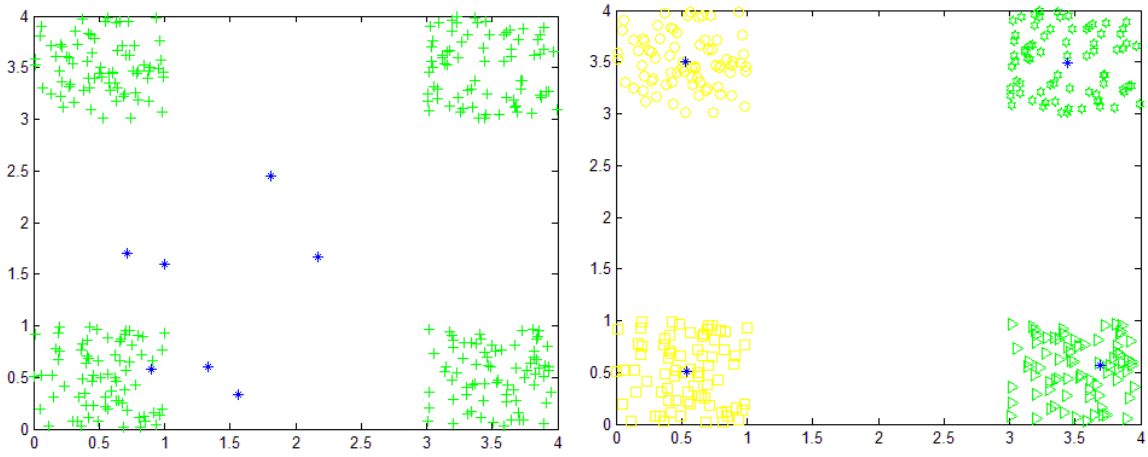


Figure 4.1: (a) Unlabelled data set with trial cluster centroids.
(b) Clustering result with ACDE

Previous experiment illustrates using of ACDE algorithm and some of its performance results which demonstrated its efficiency. We used in that experiment a well separated data set, so we obtained a good result. But when we use more complex data sets, the performance results of the ACDE algorithm will be bad.

We tested ACDE with more complex synthetic data set which is shown in Figure 4.2 (a). This data set has three clusters with regular and symmetrical shapes. The result of classification is shown in Figure 4.2 (b). Result of classification became worse when comparing it with result of previous experiment. We note that the data set was classified incorrectly and ACDE algorithm did not detect the correct number of clusters. ACDE algorithm classified data set to four clusters instead of three. The largest cluster which is shown in the bottom of Figure 4.2 (b) is separated to two clusters. The two clusters (its points are marked as stars and triangles symbols) which are shown in the bottom of Figure 4.2 (b) must merge to form one cluster. We note also that the points of clusters did not classify correctly. The upper left cluster (its points are marked as square symbols) was classified correctly, but the upper right cluster (its points are marked as circles) has some points which are classified incorrectly and assigned to the lower cluster (its points are marked as stars).

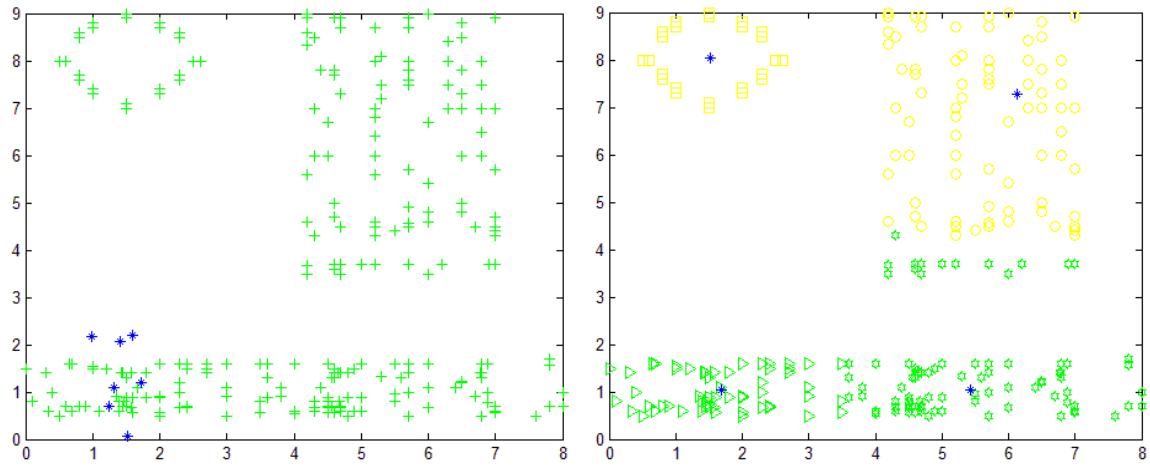


Figure 4.2: (a) Unlabelled data set with trial cluster centroids.
(b) Clustering result with ACDE.

4.2 Performance of a New Point Symmetry-Based Distance Measure

We implemented experiment to test efficiency of using Euclidean and a new point symmetry-based distance measures for classification. We focused our study in this experiment for testing these distances only, so we used k-means algorithm instead of ACDE for simplicity. We used Euclidean distance and a new point symmetry-based distance for measuring symmetry between clusters. We used in this experiment the same data set which is used previously for testing performance of ACDE algorithm.

Figure 4.3 (a) shows the used data set and the initial locations of centroids which are used by k-means algorithm. The result of classification by using K-means with Euclidean distance is shown in Figure 4.3 (b). Of course, all the clusters are classified incorrectly. We can use Euclidean distance for classifying clusters with spherical shape, so this data set will not be suitable. As shown in the Figure; the right upper cluster is classified into three groups where some of the points that are denoted as triangles are assigned to the left upper cluster, and some of the other points which are denoted as stars are assigned to lower cluster.

We can conclude that using Euclidean distance is not suitable for measuring symmetry distances between clusters which are non-globular shapes and have different sizes.

We repeated this experiment by using the new point symmetry-based distance measure instead of Euclidean distance with k-means algorithm. The new point symmetry-based distance measure is suitable for measuring symmetry between clusters of regular and symmetrical shapes.

Figure 4.3 (c) shows result of classification the same data set by using k-means algorithm with a new point symmetry-based distance measure. We note that the data set

is classified correctly to three clusters. Points corresponding to each cluster are marked in specific symbols.

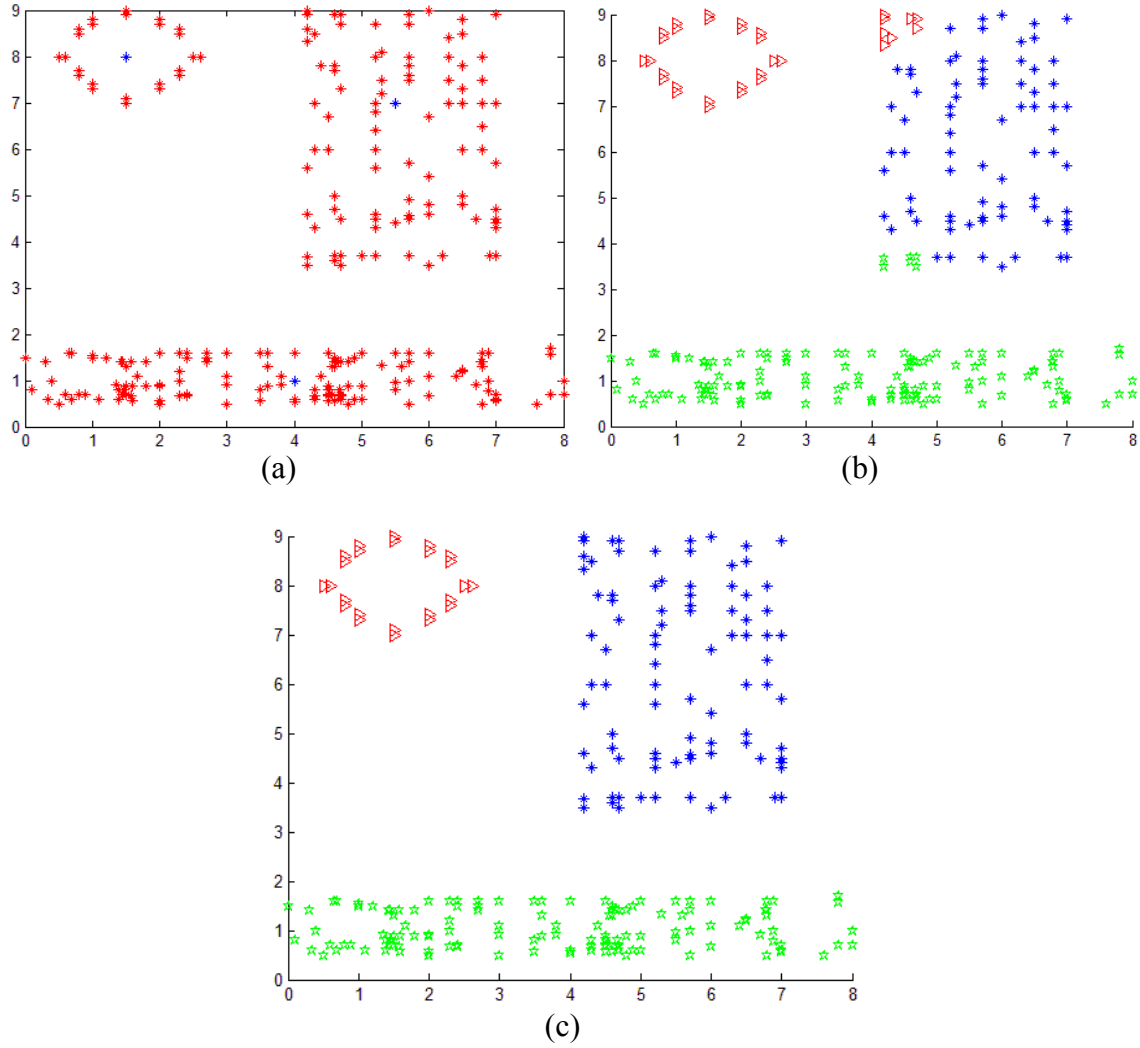


Figure 4.3: (a) Unlabelled data set with trial cluster centroids.
(b) Clustering result by K-means with Euclidean distance.
(c) Clustering result by K-means with the PS-based distance measure.

From the other side, we can classify this data set with ACDE algorithm by using suitable index as its fitness function like Equation 2.8.

At this moment, we can say that we have a good distance measure in view of the result shown in Figure 4.3(c). But this result will change when classify more complex data set as shown in Figure 4.4.

Figure 4.4 (a) shows a complex synthetic data set in two dimensions contains four clusters which have irregular and unsymmetrical shapes and have different sizes. As

shown in the figure, the clusters are slightly overlapped which cause difficulty for classifying by many algorithms which are described in the literature.

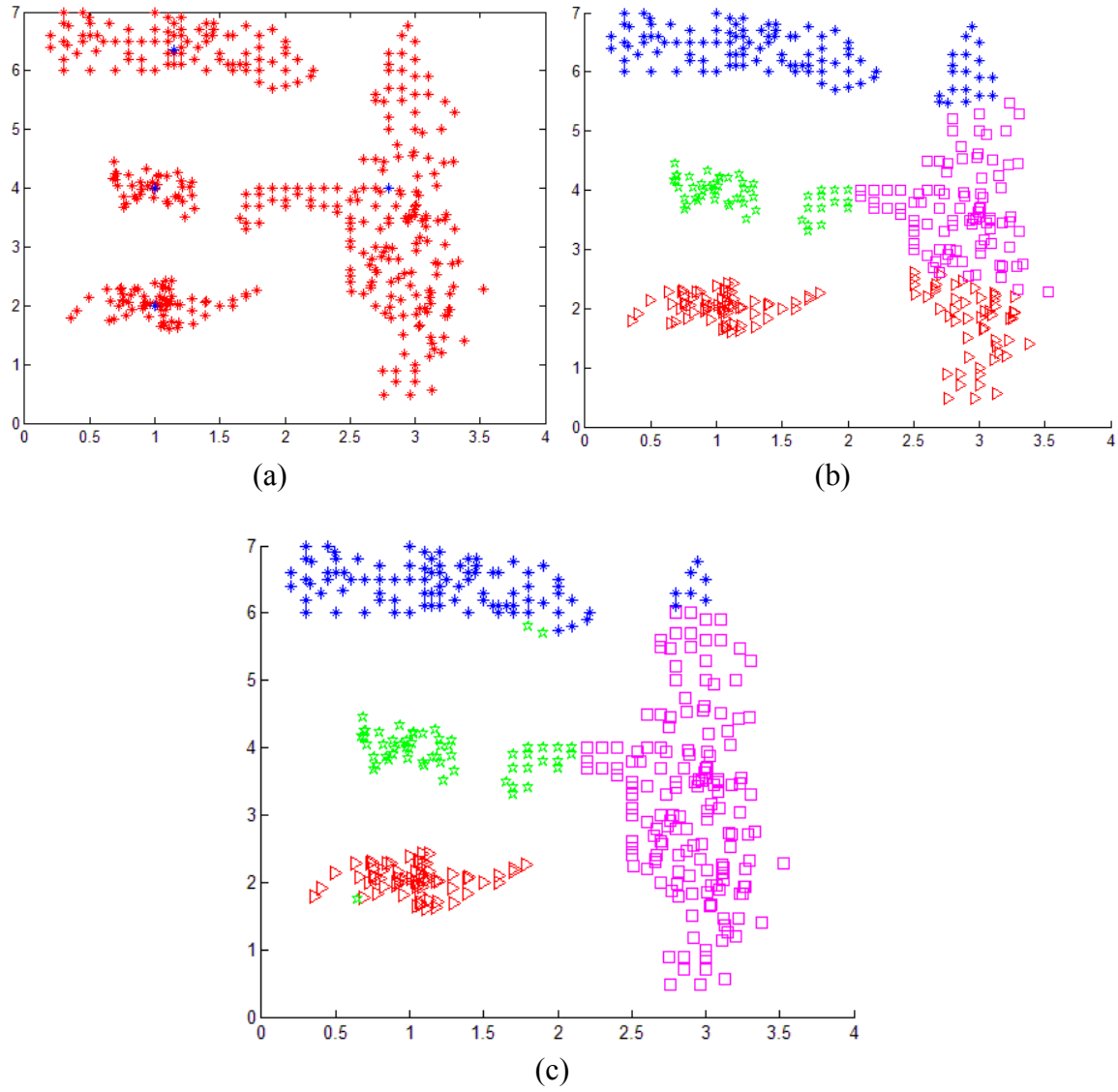


Figure 4.4: (a) Unlabelled data set with trial cluster centroids.
(b) Clustering result by K-means with Euclidean distance measure.
(c) Clustering result by K-means with the PS-based distance measure.

We classified the data set of Figure 4.4 (a) by using k-means with Euclidean distance and a new point symmetry-based distance. Figure 4.4 (b) shows the result of classification this data set by using k-means with Euclidean distance. As shown in the figure, all clusters are classified incorrectly. We note that the largest cluster is classified into four groups and most of its points are assigned to the other three clusters. The main

explanation for this result is that the data points are assigned to the nearest centroid in spite of they are connected to other cluster.

We repeated this experiment by using the new point symmetry-based distance measure instead of Euclidean distance with k-means algorithm. Figure 4.4(c) shows the result of classification by using k-means with the new point symmetry-based distance measure. We note that clusters did not classify correctly, but of course this result is better than result of using Euclidean distance which is shown in Figure 4.4(b). Many of data points are classified incorrectly by using Euclidean distance while they are classified correctly by using the new point symmetry-based distance measure.

We tried manually to choose different values of θ to enhance the result of classification by using the new point symmetry-based distance measure. Figure 4.5 shows the best results after changing the value of θ to 0.8. We note that one of the four clusters (its points are marked as triangle symbols) is classified correctly. We also noted that If θ is a small value, then percentage of data points which are classified incorrectly will increase. And vice versa, if θ is a large value then percentage of data points which are classified incorrectly will decrease.

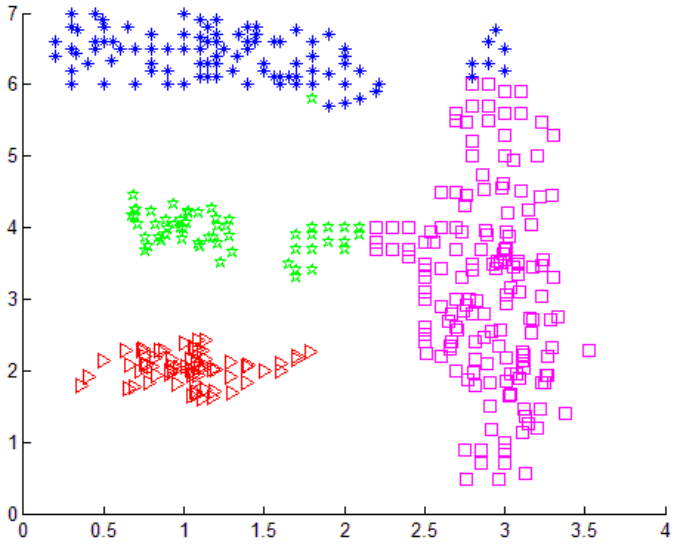


Figure 4.5: The best result achieved by clustering with changing θ when using the PS-based distance.

We used also concept of [46] for calculating θ in our experiments, but as shown in Figure 4.4 (c), it is not effective with this data set and clusters are classified incorrectly.

We conclude that using the new point symmetry-based distance measure with k-means algorithm is insufficient for classifying complex data sets which have clusters of irregular and unsymmetrical shapes. When we used this distance measures instead of Euclidean distance, we noted that it is appropriate for some data sets which have only clusters of symmetrical shapes.

4.3 Performance of Improved PS-Based Distance Measure

In this section, we illustrate performance of using improved PS-Based distance measure in classification and overcoming pervious limitations which were presented in section 2.4. We use the *k*d-tree structure for decreasing computation cost in searching on nearest points and improving the performance of classification. We use DPs of *k*d-tree for checking connectivity of each data point with its cluster.

For simplicity we fixed the value of each centroid of clusters to be the mean value of cluster's points. And then we studied the effect of improved PS-Based distance measure on classifying data points to the nearest centroid. Figure 4.6 shows the used data set, and centroid of each cluster.

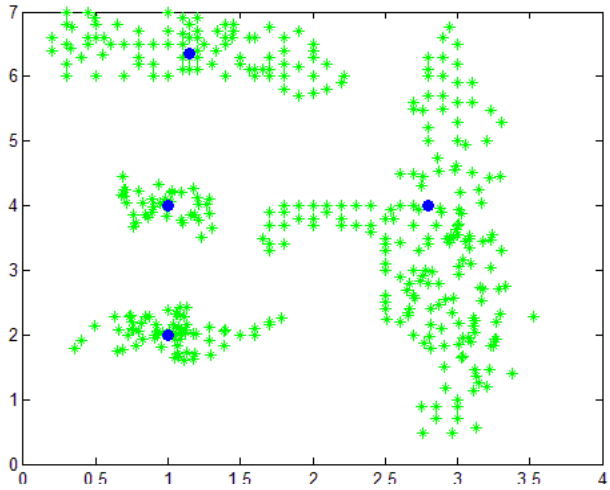


Figure 4.6: Complex synthetic data set

We used in the this experiment the same data set which classified incorrectly by using k-means with Euclidean distance and a new point symmetry-based distance measure. In this experiment we use improved PS-Based distance for measuring symmetry between clusters. We measure improved PS-Based distance between each data point $\bar{x}$ and all centroids, and then we assign the data point $\bar{x}$ to cluster which gives the minimum improved PS-Based distance between its centroid and $\bar{x}$. Using improved PS-Based distance measure produced excellent result and all clusters are classified correctly.

We tested algorithm of section 3.2. Figure 4.7 shows the output of applying this algorithm on the same data set of Figure 4.6. When we compare this Figure with Figure 4.4 (b), we note that points, which are shown as shaded squares in this Figure, correspond to most of points which are classified incorrectly in Figure 4.4 (b).

We can conclude that we fixed the main problem of wrong classification which results by using k-means with Euclidean distance measure and a new point symmetry-based distance measure. So we can use this algorithm to determine if the data point is connected to farther cluster, and then assign it to that cluster instead of nearest cluster.

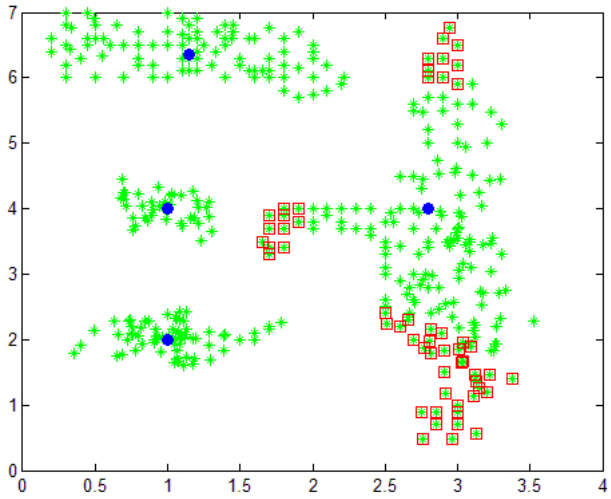


Figure 4.7: Result of using our enhancement of the PS-Based distance measure for selecting incorrectly classified points

4.4 Performance of a Novel Effective K-Means Algorithm

We tested k-means algorithm for classification a complex synthetic data set which is shown in Figure 4.8 (a), where the initial values of centroids are shown as shaded circles. We used the same data set which is used for testing performance of Euclidean distance measure, a new point symmetry-based distance measure, and improved PS-Based distance measure. We used this data set because it has clusters of irregular shapes, different sizes and slightly overlapped. This data set did not classify by many algorithms which are described in literature, and using it in this experiment we illustrate the contrast of using different distance measures with k-means.

We assume that the algorithm has converged when no change in the values of centroids. Firstly, we used Euclidean distance for measuring the distance between data points and centroids in step 2 of k-means algorithm and assigning each data point to the nearest centroid (corresponding to the minimum distance). The results were worse as shown in Figure 4.8 (c). We note that all clusters are classified incorrectly and the largest cluster is classified to four groups. We note also that the centroid of the lower cluster is moved to region which is empty of data points.

After that, we used PS-distance measure (Equation 2.12) instead of Euclidean distance. Of course the results were better as shown in Figure 4.8 (d), but only one cluster (its points are marked as square symbols) is classified correctly.

Finally, we used improved PS-distance measure (Equation 3.1) to calculate the distance between data points and centroids (step 2 of k-means Algorithm), and select the minimum distance between each data point and centroids. Figure 4.8 (b) shows selected DPs of *k*d-tree (marked as circles) which are used by Equation 3.1. The results were the best as shown in Figure 4.8 (e). We note that all clusters are classified correctly and all centroids, which are shown as shaded squares, are calculated correctly.

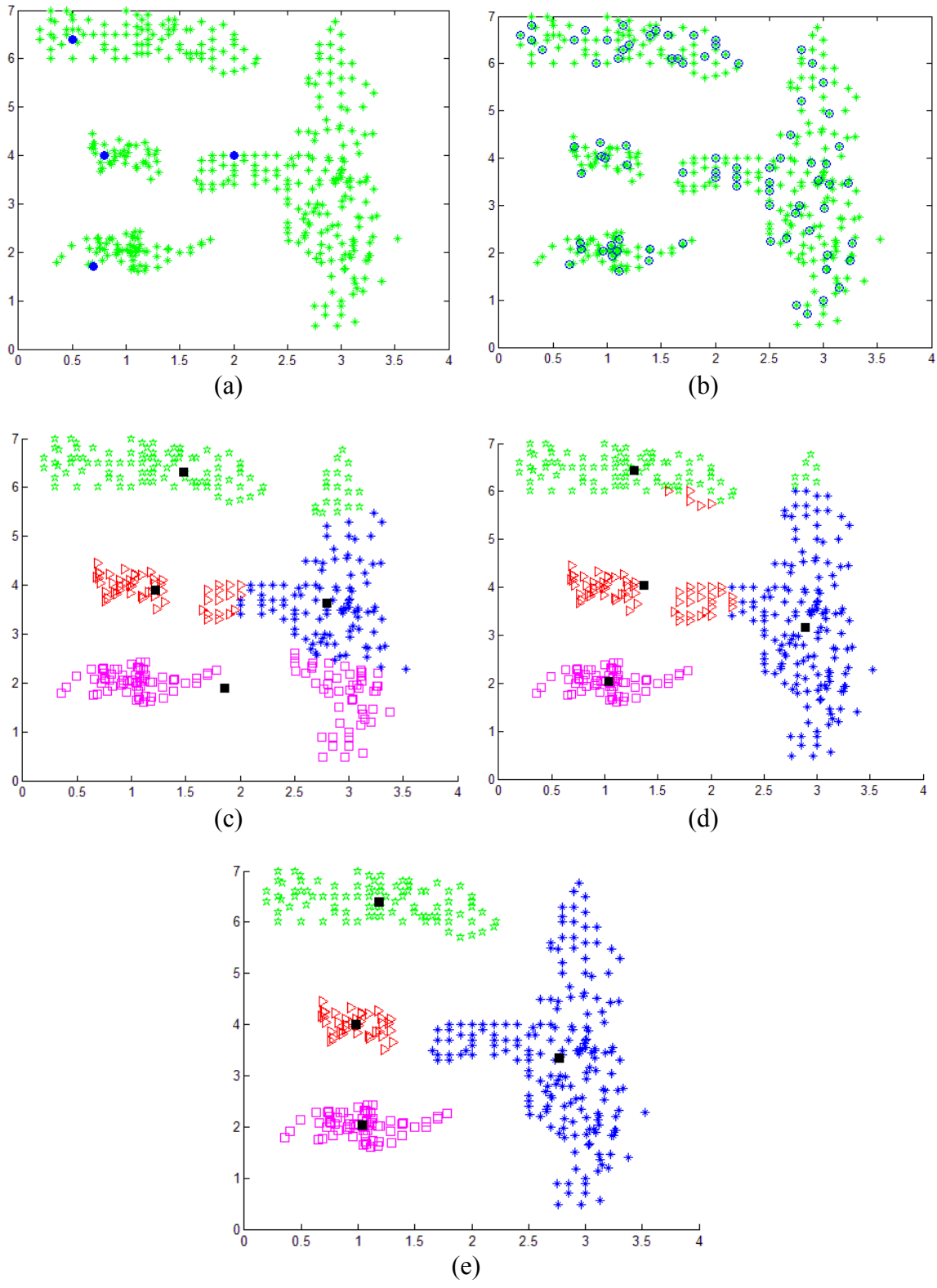


Figure 4.8: (a) Complex Synthetic data set (b) DPs of *k*d-tree.
(c) Using Euclidean distance (d) Using PS-distance measure
(e) Using improved PS-distance measure

We tested the performance of our novel algorithm and compared it with k-means using Euclidean distance and PS-distance measure. We measured elapsed time,

percentage of data points which are classified incorrectly, and the number of iterations executed until the algorithm converges. The algorithms were applied to synthetic data set which is shown in Figure 4.8 (a). The algorithms were implemented in MATLAB 7.3 (R2006b) on laptop Intel(R) Core(TM) 2 CPU. The clock speed of the processors is 1.66 GHz, and the memory size is 1.00 GB of RAM. Table 4.1 shows the results.

Table 4.1: Performance Analysis of K-Means and Novel Effective K-Means

Algorithm	***Elapsed Time (s)***	***Error (%)***	***Iterations (#)***
k-means using Euclidean distance	0.029259	0.2436	8
k-means using PS-distance measure	7.113533	0.0872	26
Novel Effective K-Means Algorithm	3.383083	0.0	3

We note that using Euclidean distance with k-means takes the smallest elapsed time, but the percentage of data points (percentage of error) which are classified incorrectly is the largest as shown in Figure 4.8 (c). Percentage of error is decreased when k-means is used with PS-Based distance measure, but it takes more elapsed time. It needs more number of iterations until the convergence takes place.

Our novel effective k-means algorithm gave the best performance. It takes the smallest number of iterations, and it classified all data points correctly. It takes more time for classification when comparing it with Euclidean distance and it takes less time when comparing it with PS-Based distance measure.

The time elapsed of step 2 of a novel effective k-means algorithm is 0.123331 seconds, so the time elapsed by our novel effective k-means algorithm is 3.259752 seconds without calculating the time of selecting DPs of *k*d-tree. We can conclude that selecting DPs of *k*d-tree did not influence the total elapsed time of using a novel effective k-means algorithm, because the *k*d-tree is created only once and then its nodes are used many times in the algorithm. This also matches the using of high dimensional data set.

We compared performance of our novel effective k-means algorithm with classical k-means algorithm. The following real-life data sets [51] are used for testing performance of our novel effective k-means algorithm and classical k-means algorithm. Here, n is the number of data points, d is the number of features, and K is the number of clusters.

1. Pima Indians diabetes data set (n=768, d=8, K=2): This data were sampled from two clusters. The first cluster has 268 objects and the second cluster has 500 objects. All patients here are females at least 21 years old of Pima Indian heritage. The data contains eight relevant features: 1) number of times pregnant; 2) plasma glucose concentration a 2 hours in an oral glucose tolerance test; 3) diastolic blood pressure; 4) triceps skin fold thickness; 5) 2-hour serum insulin; 6) Body mass index; 7) diabetes pedigree function; and 8) age.
2. Echocardiogram data set (n=131, d=7, K=2): The problem is to predict whether or not the patient will survive at least one year. The most difficult part of this problem is correctly predicting that the patient will not survive. This data were sampled from two clusters. The first cluster has 43 objects and the second cluster has 88 objects. The data contains seven relevant features: 1) age when heart attack occurred; 2) pericardial-effusion; 3) fractional-shortening; 4) E-point septal separation; 5) left ventricular end-diastolic dimension; 6) wall motion score; and 7) wall motion index.
3. Ecoli data set (n=336, d=7, K=8): The data were sampled from eight different classes: 1) cp (143 objects); 2) im (77 objects); 3) imS (2 objects); 4) imL (2 objects); 5) imU (35 objects); 6) om (20 objects); 7) omL (5 objects); and 8) pp (52 objects). The data contains seven relevant features: 1) McGeoch's method for signal sequence recognition; 2) von Heijne's method for signal

sequence recognition; 3) von Heijne's Signal Peptidase II consensus sequence score; 4) Presence of charge on N-terminus of predicted lipoproteins; 5) score of discriminant analysis of the amino acid content of outer membrane and periplasmic proteins; 6) score of the ALOM membrane spanning region prediction program; and 7) score of ALOM program after excluding putative cleavable signal regions from the sequence.

4. Hayes-Roth data set (n=132, d=4, K=3): This data were sampled from three clusters: The first cluster has 51 objects; the second cluster has 51 objects; and the third cluster has 30 objects. The data set contains 4 numeric-valued attributes: 1) hobby; 2) age; 3) educational level; and 4) marital status.
5. Statlog (Heart) data set (n=170, d=13, K=2): This data set is a heart disease database. The data set contains 13 attributes: 1) age; 2) sex; 3) chest pain type; 4) resting blood pressure; 5) serum cholestoral; 6) fasting blood sugar > 120 mg/dl?; 7) resting electrocardiographic results; 8) maximum heart rate achieved; 9) exercise induced angina; 10) ST depression induced by exercise relative to rest; 11) the slope of the peak exercise ST segment; 12) number of major vessels; and 13) thal (normal; fixed defect; or reversable defect). The objective is to classify each data vector into present (120 objects) or absent (150 objects).
6. Post-Operative patient data set (n=90, d=8, K=3): This is a data set of patient features. The classification task of this data set is to determine where patients in a postoperative recovery area should be sent to next. Because hypothermia is a significant concern after surgery, the attributes correspond roughly to body temperature measurements. The data set contains eight attributes: 1) patient's internal temperature; 2) patient's surface temperature; 3) oxygen saturation; 4) last measurement of blood pressure; 5) stability of patient's surface temperature;

6) stability of patient's core temperature; 7) stability of patient's blood pressure; and 8) patient's perceived comfort at discharge. The problem is to predict the current discharge decision: 1) patient sent to Intensive Care Unit (2 objects); 2) patient prepared to go home (24 objects); and 3) patient sent to general hospital floor (64 objects).

7. Statlog (image segmentation) data set (n=2310, d=19, K=7): The instances were drawn randomly from a database of 7 outdoor images. The images were hand segmented to create a classification for every pixel. Each instance is a 3x3 region. The data set contains 19 attributes: 1) the column of the center pixel of the region; 2) the row of the center pixel of the region; 3) the number of pixels in a region; 4) the results of a line extractoin algorithm that counts how many lines of length 5 with low contrast, less than or equal to 5, go through the region; 5) same as short-line-density-5 but counts lines of high contrast, greater than 5; 6) measure the contrast of horizontally adjacent pixels in the region. The mean is given; 7) measure the contrast of horizontally adjacent pixels in the region. The standard deviation is given; 8) measures the contrast of vertically adjacent pixels. The mean is given; 9) measures the contrast of vertically adjacent pixels. The standard deviation is given; 10) intensity mean; 11) the average over the region of the R value; 12) the average over the region of the B value; 13) the average over the region of the G value; 14) measure the excess red; 15) measure the excess blue; 16) measure the excess green; 17) value mean; 18) saturation mean; and 19) hue mean. The data were sampled from seven different Classes: 1) brickface; 2) sky ; 3) foliage; 4) cement; 5) window; 6) path; and 7) grass. The number of objects that belong to each cluster is 330.

We used Waikato Environment for Knowledge Analysis (WEKA) [52] version 3.6.2 for classifying data sets by classical k-means algorithm. Figure 4.9 shows graphical user interface of Weka Explorer.

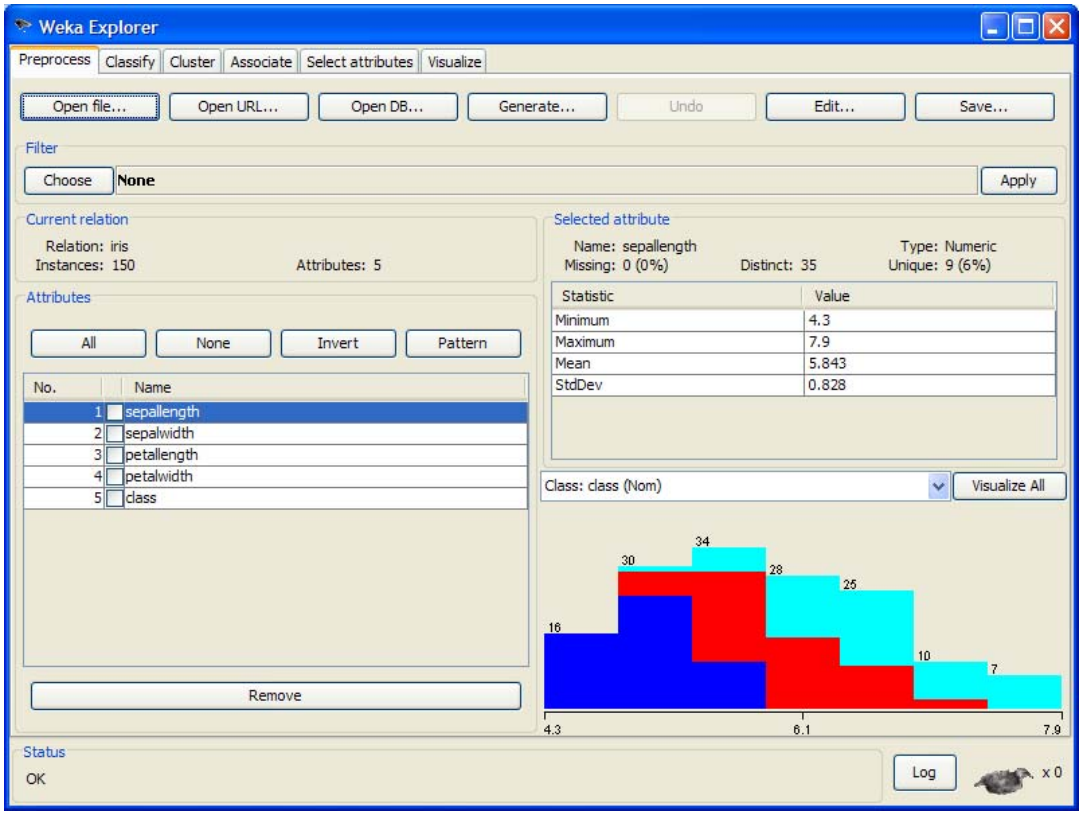


Figure 4.9: Weka Explorer

Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes.

We tested performance of classical k-means and novel effective k-means by counting the data points which are classified incorrectly. The data set description and the individual performance of classical k-means algorithm and our novel effective k-means algorithm are summarized in Table 4.2.

Table 4.2: The data sets description, the number of incorrectly clustered instances by k-means and novel effective k-means

#	*Data set name*	*n*	*D*	*K*	*# Datapoints incorrectly clustered by k-means*	*# Datapoints incorrectly clustered by novel effective k-means*
1	pima Indians diabetes	768	8	2	261	236
2	echocardiogram	131	7	2	43	41
3	ecoli	336	7	8	130	54
4	hayes-roth	132	4	3	78	60
5	statlog (heart)	270	13	2	110	99
6	post-operative patient	90	8	3	52	26
7	statlog (image segmentation)	2310	19	7	950	580

We observed that our novel effective k-means algorithm performed very well. We found that classical k-means algorithm failed to classify 0.43% of the average number of all instances in the data sets while our novel effective k-means algorithm performed 0.31%. So we can conclude that our novel effective k-means performs better performance than classical k-means.

4.5 Performance of Dynamic Linkage Clustering using KD-Tree

Experimental results are shown in this section to demonstrate the effectiveness of the Dynamic Linkage Clustering algorithm using KD-Tree (DLCKDT). We used synthetic and real data sets for testing efficiency of the proposed algorithm. We tested our algorithm with many data sets.

Figure 4.10 (a) shows a complex synthetic data set in two dimensions to be classified by our proposed algorithm. This data set contains two clusters which are slightly overlapped.

DPs of *k*d-tree (marked as circles) which are used for classifying are shown in Figure 4.10 (b). We note that DPs are distributed though whole data set, and located in the dense regions.

Figure 4.10 (c) shows the output of step 3 in the algorithm. We note that every data point is connected to the nearest DP. We note that this step generates a collection

of small clusters where DPs form their centroids. We note that 32 groups (marked with different colors and shapes) are created, and number of these groups equal to number of used DPs.

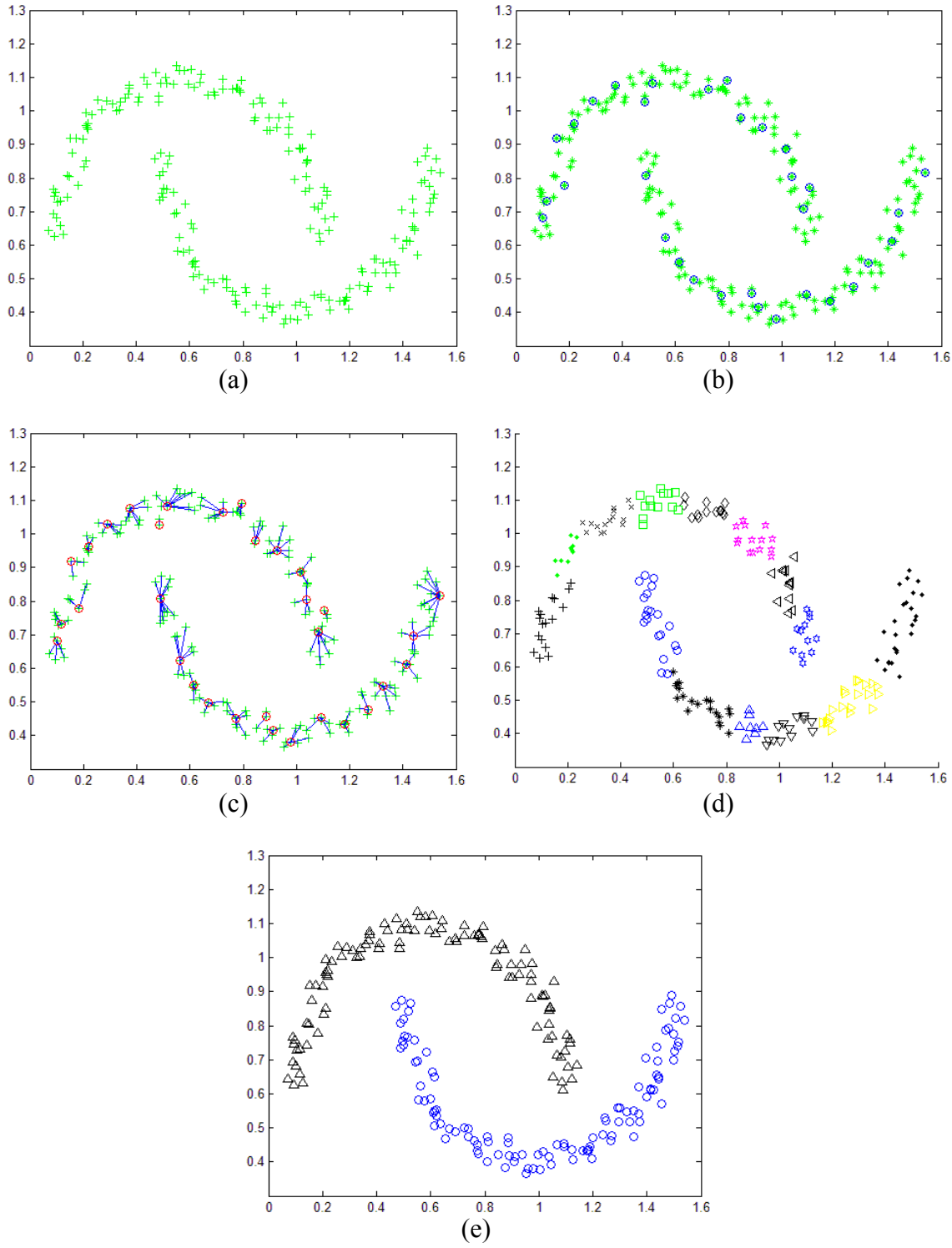


Figure 4.10: (a) Synthetic data set (b) DPs of *k*d-tree.(c) Clusters of DPs (d) Merging every two adjacent clusters of DPs (e) The output of algorithm

The output of step 4 in the algorithm is shown in Figure 4.10 (d). It merges every two adjacent clusters of the previous step. We note that the number of groups is decreased to 14 groups. This number is equal approximately to half number of groups which are created by previous step (step 3) in the algorithm. This step reduces elapsed time for running the algorithm.

The final output of the algorithm is shown in Figure 4.10 (e). We note that all data points are classified correctly despite of clusters are overlapped. Points of the first cluster are labelled as triangles and points of the second cluster are labelled as circles.

We used our algorithm for classifying a lot of complex data sets. Figure 4.11 illustrates the power of our algorithm. We used a data set that consists of two circles (two clusters); one of them is inside the other as shown in Figure 4.11 (a).

Figure 4.11 (b) shows the DPs (marked as circles) of *k*d-tree. We note that DPs are distributed though whole data set, and located in the dense regions. We note that the DPs formed the shape of clusters with small number of data points.

Figure 4.11 (c) shows the clusters of DPs. We note that every data point is connected to the nearest DP. We note that this step generates a collection of small clusters where DPs form their centroids. We note that many groups, which are marked with different colors and shapes, are created, and number of these groups equal to number of used DPs.

The results of merging every two adjacent clusters of DPs are shown in Figure 4.11 (d). We note that the number of group is reduced to only 24 groups.

The output of the algorithm is shown in Figure 4.11 (e). We note that data set is classified correctly into to clusters. Data points of the first cluster (the smallest circle) are marked as circles and the data points of the second cluster (the largest circle) are marked as triangles.

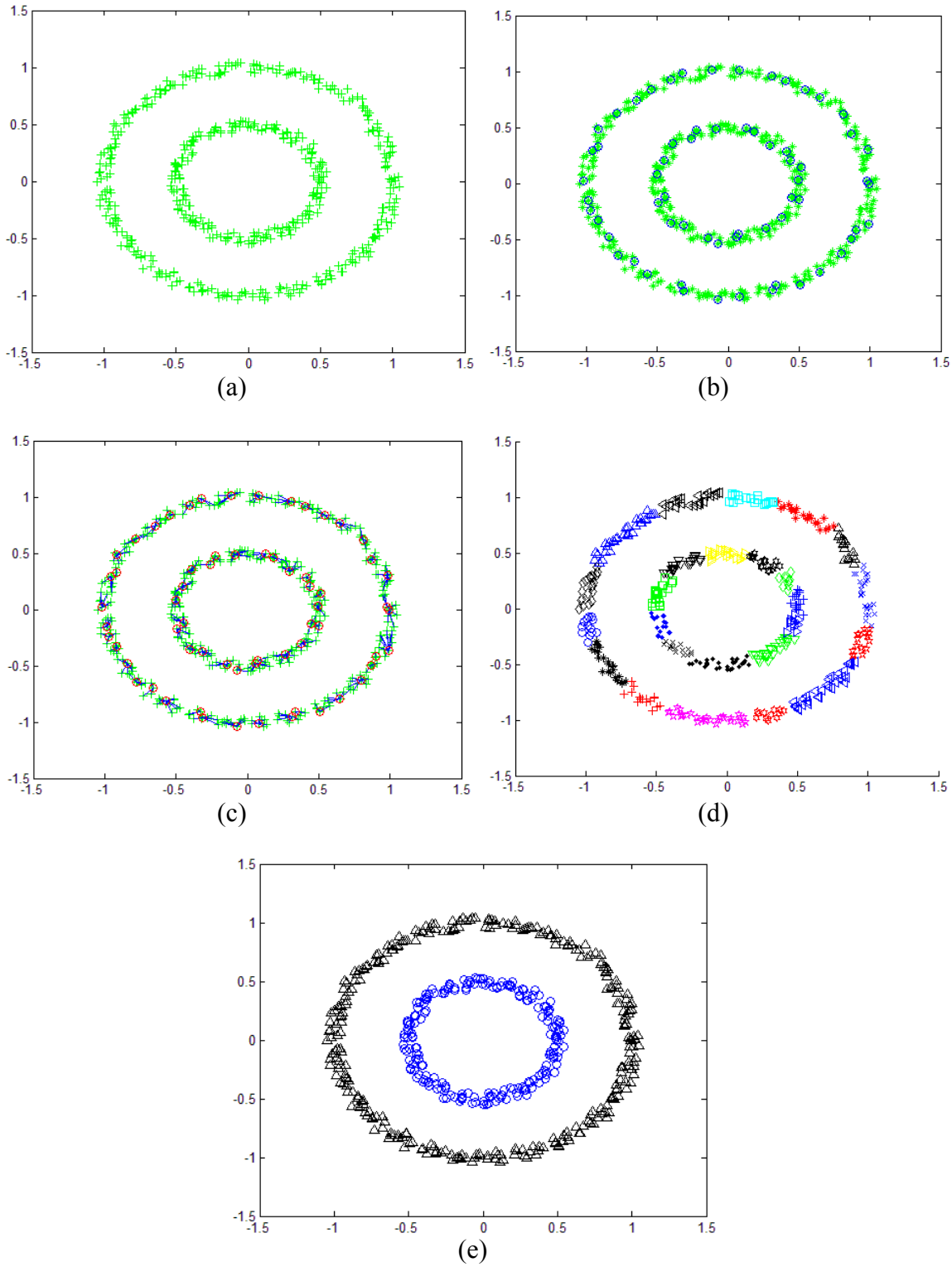


Figure 4.11: (a) Complex Synthetic data set (b) DPs of *k*d-tree. (c) Clusters of DPs (d) Merging every two adjacent clusters of DPs (e) The output of algorithm

The data sets of Figure 4.9 (a) and Figure 4.10 (a) can not be classified by our novel effective k-means algorithm which is presented in section 3.3 or by a lot of

algorithms which are described in the literature. So we can conclude that our algorithm classify complex data sets.

We tested the time complexity of our proposed algorithm. We measured the elapsed time of classifying data set of Figure 4.8 (a). Table 4.3 shows a comparison between k-means, our novel effective k-means algorithm and our novel clustering algorithm DLCKDT. We tested this algorithm with the same system which is used for testing algorithms in section 4.4. As expected, the novel clustering algorithm requires more time than the novel effective k-means algorithm. But of course, DLCKDT algorithm is more robust because it does not depend on the initial conditions like k-means and the novel effective k-means.

Table 4.3: Performance Analysis of K-Means, Novel Effective K-Mean, and DLCKDT

Algorithm	***Elapsed Time (s)***
k-means using Euclidean distance	0.029259
k-means using PS-distance measure	7.113533
A Novel Effective K-Means Algorithm	3.383083
A Dynamic Linkage Clustering using KD-Tree	5.603688

Our proposed algorithm is a density-based clustering algorithm because it finds a number of clusters starting from the estimated density distribution of corresponding nodes. So we compared performance of our proposed algorithm with DBSCAN algorithm. The following real-life data sets [51] are used for testing performance of our proposed algorithm DLCKDT and DBSCAN. Here, n is the number of data points, d is the number of features, and K is the number of clusters.

1. Iris plants data set (n = 150, d = 4, K = 3): This is a well-known data set with 4 inputs, 3 classes, and 150 data vectors. The data set consists of three different species of iris flower: Iris setosa, Iris virginica, and Iris versicolour. One class is linearly separable from the other 2; the latter are not linearly separable from each other. For each species, 50 samples with four features each (sepal length,

sepal width, petal length, and petal width) were collected. The number of objects that belong to each cluster is 50.

2. Abalone data set (n=1253, d=8, K=3): This is a data set for predicting the age of abalone from physical measurements. The data were sampled from three clusters: the first cluster has 397 objects, the second cluster has 434 objects, and the last cluster has 422 objects. The data contains eight relevant features: 1) sex; 2) length; 3) diameter; 4) height; 5) whole weight; 6) shucked weight; 7) viscera weight; and 8) shell weight.
3. Contraceptive method choice data set (n=1473, d=9, K=3): This data set is a subset of the 1987 National Indonesia Contraceptive Prevalence Survey. The samples are married women who were either not pregnant or do not know if they were at the time of interview. The problem is to predict the current contraceptive method choice: 1) no use (629 objects); 2) long-term methods (333 objects); or 3) short-term methods (511 objects) of a woman based on her demographic and socio-economic characteristics. The data contains nine relevant features: 1) wife's age; 2) wife's education; 3) wife's education; 4) number of children ever born; 5) wife's religion; 6) wife's now working?; 7) husband's occupation; 8) standard-of-living index; and 9) media exposure.
4. Haberman's survival data set (n=306, d=3, K=2): The data set contains cases from a study that was conducted between 1958 and 1970 at the University of Chicago's Billings Hospital on the survival of patients who had undergone surgery for breast cancer. The objective is to classify each data vector into: 1) the patient survived 5 years or longer (225 objects); or 2) the patient died within 5 year (81 objects). The data contains three relevant features: 1) age of

patient at time of operation; 2) patient's year of operation; and 3) number of positive axillary nodes detected.

5. Heart disease data set (n=303, d=13, K=2): This is a data set with 13 inputs, 2 classes, and 303 data vectors. The "goal" field refers to the presence of heart disease in the patient. The problem is to predict the diagnosis of heart disease (angiographic disease status) by classifying each data vector into: 1) < 50% diameter narrowing (164 objects); or 2) > 50% diameter narrowing (139 objects). The data contains 13 relevant features: 1) age; 2) sex; 3) chest pain type; 4) resting blood pressure; 5) serum cholestoral; 6) fasting blood sugar > 120 mg/dl?; 7) resting electrocardiographic results; 8) maximum heart rate achieved; 9) exercise induced angina?; 10) ST depression induced by exercise relative to rest; 11) the slope of the peak exercise ST segment; 12) number of major vessels; and 13) thal (normal, fixed defect, or reversable defect);

We used Waikato Environment for Knowledge Analysis (WEKA) for classifying data sets by DBSCAN algorithm. We assigned input parameters of DBSCAN algorithm to epsilon=0.9 and minPoints= 6 for classifying all data sets. We tested performance of DBSCAN and our algorithm DLCKDT by counting the data points which are classified incorrectly. The data set description and the individual performance of DBSCAN algorithm and our algorithm DLCKDT are summarized in Table 4.4.

Table 4.4: The data sets description, the number of incorrectly clustered instances by DBSCAN algorithm and DLCKDT algorithm

#	***Data set name***	***n***	***d***	***K***	***# Datapoints incorrectly clustered by DBSCAN***	***# Datapoints incorrectly clustered by DLCKDT***
1	Iris	150	4	3	100	39
2	abalone	1253	8	3	819	616
3	contraceptive method choice	1473	9	3	876	851
4	haberman's survival	306	3	2	269	79
5	heart disease	303	13	2	179	133

We observed that the proposed algorithm performed very well. We found that the DBSCAN algorithm failed to classify 0.68% of the average number of all instances in the data sets while DLCKDT performed 0.41%.

We can conclude that the proposed algorithm performs better performance than DBSCAN algorithm and it doesn't reliance on a priori knowledge and user defined parameters like DBSCAN.

4.6 Performance of Improved ACDE

We tested the enhanced algorithm by classifying the data set shown in Figure 4.8 (a). The result of classifying the data set is shown in Figure 4.12. We note that three of clusters in the data set are classified correctly, but the largest cluster is divided into small clusters.

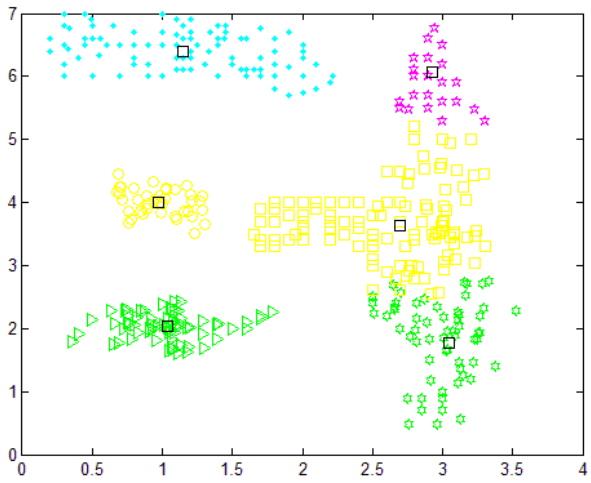


Figure 4.12: Classification resulted by phase I of improved ACDE

Figure 4.13 shows the connected graph of the data set used in Figure 4.12. By comparing Figures 4.12 and 4.13 we note that some pointes in sub-clusters of Figure 4.12 are connected with other pointes in the adjacent sub-clusters as shown in Figure 4.13, so we can connect them to form one cluster.

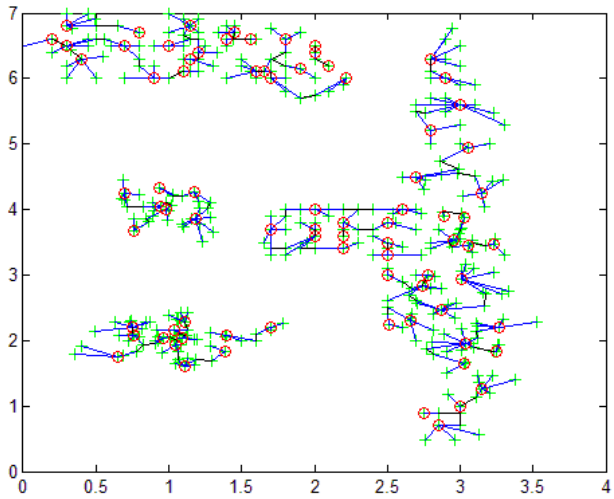


Figure 4.13: The connected graph

We compared performance of classical ACDE algorithm with our improved ACDE algorithm. The following real-life data sets [51] are used for testing performance of classical ACDE algorithm and our improved ACDE algorithm. Here, n is the number of data points, d is the number of features, and K is the number of clusters.

1. Glass (n = 214, d = 9, K = 6): The data were sampled from six different types of glass: 1) building windows float processed (70 objects); 2) building windows non float processed (76 objects); 3) vehicle windows float processed (17 objects); 4) containers (13 objects); 5) tableware (9 objects); and 6) headlamps (29 objects). Each type has nine features: 1) refractive index; 2) sodium; 3) magnesium; 4) aluminum; 5) silicon; 6) potassium; 7) calcium; 8) barium; and 9) iron.
2. Statlog (vehicle silhouettes) data set (n = 846, d = 18, K = 4): The purpose is to classify a given silhouette as one of four types of vehicle, using a set of features extracted from the silhouette. The data were sampled from four different types of vehicle: 1) a double decker bus (218 objects); 2) Cheverolet van (199 objects); 3) Saab 9000 (217 objects); and 4) an Opel Manta 400 (212 objects). Each type has 18 features: 1) compactness; 2) circularity; 3) distance circularity

area; 4) radius ratio; 5) pr.axis aspect ratio; 6) max.length aspect ratio; 7) scatter ratio; 8) elongatedness area; 9) pr.axis rectangularity area; 10) max.length rectangularity area; 11) scaled variance major; 12) scaled variance minor; 13) scaled radius of gyration; 14) skewness about major; 15) skewness about minor; 16) kurtosis about major; 17) kurtosis about minor; and 18) hollows ratio.

3. Yeast data set (n = 1484, d = 8, K = 10): The purpose is to localize site of protein. The data contains eight relevant features: 1) McGeoch's method for signal sequence recognition; 2) von Heijne's method for signal sequence recognition; 3) score of the ALOM membrane spanning region prediction program; 4) score of discriminant analysis of the amino acid content of the N-terminal region of mitochondrial and non-mitochondrial proteins; 5) Presence of "HDEL" substring; 6) peroxisomal targeting signal in the C-terminus; 7) score of discriminant analysis of the amino acid content of vacuolar and extracellular proteins; and 8) score of discriminant analysis of nuclear localization signals of nuclear and non-nuclear proteins. The data were sampled from 10 classes: 1) 244 objects; 2) 429 objects; 3) 463 objects; 4) 44 objects; 5) 35 objects; 6) 51 objects; 7) 163 objects; 8) 30 objects; 9) 20 objects; and 10) 5 objects;
4. Zoo data set (n = 101, d = 16, K = 7): The data were sampled from seven sets of animals: 1) aardvark, antelope, bear, boar, buffalo, calf, cavy, cheetah, deer, dolphin, elephant, fruitbat, giraffe, girl, goat, gorilla, hamster, hare, leopard, lion, lynx, mink, mole, mongoose, opossum, oryx, platypus, polecat, pony, porpoise, puma, pussycat, raccoon, reindeer, seal, sealion, squirrel, vampire, vole, wallaby, wolf (41 objects); 2) chicken, crow, dove, duck, flamingo, gull, hawk, kiwi, lark, ostrich, parakeet, penguin, pheasant, rhea, skimmer, skua,

sparrow, swan, vulture, wren (20 objects); 3) pitviper, seasnake, slowworm, tortoise, tuatara (5 objects); 4) pitviper, seasnake, slowworm, tortoise, tuatara (13 objects); 5) frog, frog, newt, toad (4 objects); 6) flea, gnat, honeybee, housefly, ladybird, moth, termite, wasp (8 objects); and 7) clam, crab, crayfish, lobster, octopus, scorpion, seawasp, slug, starfish, worm (10 objects). The data contains 16 relevant features: 1) hair; 2) feathers; 3) eggs; 4) milk; 5) milk; 6) aquatic; 7) predator; 8) toothed; 9) backbone; 10) backbone; 11) venomous; 12) fins; 13) fins; 14) tail; 15) domestic; and 16) catsize.

5. Lenses data set (n = 24, d = 4, K = 3): The data contains 4 relevant features: 1) age of the patient; 2) spectacle prescription; 3) astigmatic; and 4) tear production rate. The objective is to classify each data vector into: 1) the patient should be fitted with hard contact lenses (4 objects); 2) the patient should be fitted with soft contact lenses (5 objects); and 3) the patient should not be fitted with contact lenses (15 objects).
6. Wine (n = 178, d = 13, K = 3): This is a classification problem with "well-behaved" class structures. The data contains 13 relevant features: 1) Alcohol; 2) Malic acid; 3) Ash; 4) Alcalinity of ash; 5) Magnesium; 6) Total phenols; 7) Flavanoids; 8) Nonflavanoid phenols; 9) Proanthocyanins; 10) Color intensity; 11) Hue; 12) OD280/OD315 of diluted wines; and 13) Proline. The data were sampled from three types of wine: 1) 59 objects; 2) 71 objects; and 3) 48 objects.
7. Soybean (small) data set (n = 47, d = 35, K = 4): The data were sampled from four different types of soybean: 1) 10 objects; 2) 10 objects; 3) 10 objects; and 4) 17 objects. The data contains 35 relevant features: 1) date; 2) plant-stand; 3) precip; 4) temp; 5) hail; 6) crop-hist; 7) area-damaged; 8) severity; 9) seed-

tmt; 10) germination; 11) plant-growth; 12) leaves; 13) leafspots-halo; 14) leafspots-marg; 15) leafspot-size; 16) leaf-shread; 17) leaf-malf; 18) leaf-mild; 19) leaf-mild; 20) lodging; 21) stem-cankers; 22) stem-cankers; 23) fruiting-bodies; 24) external decay; 25) mycelium; 26) int-discolor; 27) sclerotia; 28) fruit-pods; 29) fruit spots; 30) seed; 31) mold-growth; 32) seed-discolor; 33) seed-size; 34) shriveling; and 35) roots.

We tested performance of classical ACDE and our improved ACDE algorithm by counting the data points which are classified incorrectly. The data set description and the individual performance of the classical ACDE algorithm and the improved ACDE algorithm are summarized in Table 4.5.

Table 4.5: The data set description, the number of incorrectly clustered instances by classical ACDE algorithm and improved ACDE algorithm

#	*Data set name*	*N*	*d*	*K*	*# Datapoints incorrectly clustered by classical ACDE*	*# Datapoints incorrectly clustered by Improved ACDE*
1	glass	214	9	6	148	132
2	vehicle	846	18	4	534	498
3	yeast	1484	8	10	900	756
4	zoo	101	16	7	55	18
5	lenses	24	4	3	19	9
6	wine	178	13	3	53	50
7	soybean	47	35	4	18	7

We observed that our improved ACDE algorithm performed very well. We found that classical ACDE algorithm failed to classify 0.56% of the average number of all instances in the data sets while our improved ACDE algorithm performed 0.39%. So we can conclude that our improved ACDE algorithm performs better performance than classical ACDE.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis we described an essential problem in data clustering and presented some solutions for it. We investigated using distance measures other than Euclidean type for improving the performance of clustering. We also developed a new distance measure and proved its efficiency. We developed a novel effective k-means algorithm which improved the performance of the k-mean algorithm. We developed a novel clustering algorithm by using *k*d-tree and we proved its performance. The ACDE algorithm that we presented is specific to clustering simple data sets and finding the optimal number of clusters automatically. We improved ACDE for classifying more complex data sets using kd-tree. The proposed algorithms did not have a worst-case bound on running time.

Experimental results are shown in this thesis to demonstrate the effectiveness of the proposed algorithms. We illustrated the time complexity and the performance of classifying complex data sets. We proved that the proposed algorithms can classify complex data sets more accurately than other algorithms presented in the literature.

5.2 Future Work

The work reported in this thesis may be extended in a number of ways, some of which are discussed below:

1) If the number K_{max} which is used in ACDE algorithm is small then the results will be bad when classifying complex data set, and if it is large then the elapsed time will increase to converge to global value. So we need to study the optimum value of K_{max}.

2) In chapter 4 we used a new point symmetry-based distance measure instead of Euclidean type for improving the performance of clustering. It is interesting to investigate other kinds of distance measures.
3) We used *k*d-tree for improving the performance of classification. Many optimizing search strategies in *k*d-trees are developed in literature. We can use these strategies for improving the time complexity of our algorithms and study their performance.
4) Our proposed algorithms depend on *k*d-tree for improving the performance of clustering. It is interesting to study some other kinds of trees like R+_tree and Bkd-tree: A Dynamic Scalable kd-tree.

Bibliography

[1] I. Osman, and G. Laporte, "Metaheuristics: A bibliography," Annals of Operations Research Journal, Springer Netherlands, vol. 63, no. 5 , pp. 513–623, October 1996.

[2] C. Blum, and A.Roli, "Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison," ACM Computing Surveys, vol. 35, Number 3, pp. 268–308, September 2003.

[3] G. Gan, C. Ma, and J. Wu, *Data Clustering: Theory, Algorithms, and Applications.* ASA-SIAM Series on Statistics and Applied Probability, SIAM, Philadelphia, ASA, Alexandria, VA, 2007.

[4] S. K. Halgamuge, and L. Wang, *Classification and Clustering for Knowledge Discover.* Springer-Verlag Berlin Heidelberg, New York, 2005.

[5] S. Theodoridis, and K. Koutroumbas, *Pattern Recognition.* 2nd ed. Elsevier Academic Press, Amsterdam, 2003.

[6] P. Arabie, L. J. Hubert, and G. De Soete, *Clustering and Classification.* River Edge. World Scientific Publishing, Singapore, 1996.

[7] M. Halkidi, and M. Vazirgiannis, "Clustering Validity Assessment: Finding the optimal partitioning of a data set," presented at the IEEE International Conference on Data Mining, San Jose, California, USA, pp. 187–194, 29 November - 2 December, 2001.

[8] J. C. Dunn "Well Separated Clusters and Optimal Fuzzy Partitions," Journal of Cybernetics, vol. 4, pp.95–104, 1974.

[9] R. B. Calinski, and J. Harabasz, "Adendrite Method for Cluster Analysis," Commun. Statistics - Theory and Methods, vol. 3, no. 1, pp. 1–27, 1974.

[10] D. L. Davies, and D.W. Bouldin, "A cluster separation measure," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol 1, no. 4, pp. 224–227, 1979.

[11] M. K. Pakhira, S. Bandyopadhyay, and U. Maulik, "Validity index for crisp and fuzzy clusters," Pattern Recognition, vol. 37, no. 3, pp. 487–501, March 2004.

[12] C. H. Chou, M. C. Su, and E. Lai, "A new cluster validity measure and its application to image compression," Pattern Analysis and Applications, vol. 7, no. 2, pp. 205–220, July 2004.

[13] T. Zhang, R. Ramakrishnman, and M. Linvy, "BIRCH: An efficient method for very large databases," Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996.

[14] G. Hammerly, and C. Elkan, “Alternatives to the k-means algorithm that find better clusterings,” Proc. ACM on Information and Knowledge Management, pp. 600–607, November 2002.

[15] P.S. Bradley, and U.M. Fayyad, “Refining Initial Points for K-Means Clustering,” ICML 1998, pp. 91–99, January 1998.

[16] W. Wang, J. Yang, and R. Muntz, “STING: A statistical Information grid approach to spatial data mining,” VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, Athens, Greece, August 25-29, 1997.

[17] P. Berkhin, “Survey of Clustering Data Mining Techniques,” Accrue Software, Technical Report, 2002.

[18] J. Kogan, *Introduction to Clustering Large and High-Dimensional Data.* Cambridge University Press, New York, 2007.

[19] J. Kogan, M. Teboulle, and C. Nicholas, *Grouping Multidimensional Data*. Springer-Verlag Berlin Heidelberg, New York, 2006.

[20] J. Valente de Oliveira, and W. Pedrycz, *Advances in Fuzzy Clustering and its Applications*. John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England, 2007.

[21] M. Sato-Ilic, and L. C. Jain, *Innovations in Fuzzy Clustering*. Springer-Verlag Berlin Heidelberg, New York, 2006.

[22] W. Pedrycz, *Knowledge-Based Clustering From Data to Information Granules*. John Wiley & Sons, Inc., Hoboken, New Jersey, 2005.

[23] J. MacQueen, “Some methods for classification and analysis of multivariate bservations,” Proceedings of the Fifth Berkely Symposium on Mathematical Statistics and Probability, vol. 1, pp. 281–297, 1967.

[24] A. Jain, M. Murty, and P. Flynn, “Data clustering: A Review,” ACM Computing Surveys, vol. 31, no. 3, pp. 264–323, September 1999.

[25] S. J. Redmond and C. Heneghan, “A method for initialising the K-means clustering algorithm using kd-trees,” Pattern Recognition Letters, vol. 28, no. 16, pp. 965-973, 2007.

[26] K. Mumtaz and K. Duraiswamy, “A Novel Density based improved k-means Clustering Algorithm – Dbkmeans,” International Journal on Computer Science and Engineering, vol. 2, no. 2, pp. 213-218, 2010.

[27] M. Ester, H.-P Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise,” Proceedings of the

Second International Conference on Knowledge Discovery and Data Mining, AAAI Press, Menlo Park, pp. 226–231, 1996.

[28] R. Storn, and K. Price, "Differential Evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces," Technical Report TR-95-012, ICSI, March 1995.

[29] R. Storn, and K. Price, "Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces," Journal of Global Optimization, vol. 11, no. 4, pp. 341–359, 1997.

[30] J. A. Nelder, and R. Mead, "A simplex method for function minimization," Computer Journal, vol. 7, no. 4, pp. 308–313, 1 January 1965.

[31] M. Avriel, *Nonlinear Programming: Analysis and Methods*. Dover Publishing, New York, 2003.

[32] W. L. Price, "Global optimization by controlled random search," Computer Journal, vol. 20, no. 4, pp. 367–370, 1977.

[33] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*. John Wiley, Chichester , 1966.

[34] A. E. Eiben, and J.E. Smith, *Introduction to Evolutionary Computing*. Springer, Heidelberg, 2003.

[35] K. Price, R. Storn, and J. Lampinen, *Differential Evolution - A Practical Approach to Global Optimization*. Springer, Berlin, 2005.

[36] T. Back, D. B. Fogel, and Z. Michalewicz, *Handbook of Evolutionary Computation*. IOP and Oxford University Press, Bristol, 1997.

[37] S. Das, A. Abraham and A. Konar, *Metaheuristic Clustering*. Springer-Verlag Berlin Heidelberg, 2009.

[38] S. Das, A. Abraham, and A. Konar, "Automatic clustering using an improved differential evolution algorithm," IEEE Transactions on Systems, Man, and Cybernetics, Part A, vol. 38, no. 1, pp. 218–237, 2008.

[39] J. C. Bezdek, "Numerical taxonomy with fuzzy sets," Journal of Math. Biol., pp. 157–171, 1974.

[40] J. C. Bezdek, "Cluster validity with fuzzy sets," Journal of Cybernetics, vol. 3, pp. 58–72, 1974.

[41] X. Xie, and G. Beni, "Validity measure for fuzzy clustering," IEEE Trans. Pattern Anal. Machine Learning, vol. 3, pp. 841–846, 1991.

[42] D. Gustafson, and W. Kessel, ”Fuzzy clustering with a fuzzy covariance matrix,” Proc. IEEE CDC, San Diego, CA, USA, pp. 761–766, 1979.

[43] F. Attneave, “Symmetry information and memory for pattern,” The American Journal of Psychology, vol. 68, pp. 209–222, 1995.

[44] M. C. Su, and C. H. Chou, “A modified version of the k-means algorithm with a distance based on cluster symmetry,” IEEE Transactions Pattern Analysis and Machine Intelligence, vol. 23, no. 6, pp. 674–680, 2001.

[45] S. Bandyopadhyay and S. Saha, “GAPS: A Clustering Method Using a New Point Symmetry Based Distance Measure,” Pattern Recognition, vol. 40, pp. 3430-3451, 2007.

[46] S. Bandyopadhyay, and S. Saha, “A Point Symmetry Based Clustering Technique for Automatic Evolution of Clusters,” IEEE Transactions on Knowledge and Data Engineering, vol. 20, no. 11, pp. 1–17, 2008.

[47] C. H. Chou, M. C. Su, and E. Lai, “Symmetry as a new measure for cluster validity,” Second WSEAS International Conference on Scientific Computation and Soft Computing, pp. 209–213, 2002.

[48] A. W. Moore, “An introductory tutorial on kd-trees,” PhD. dissertation, Technical Report No. 209, Computer Laboratory, University of Cambridge, 1991.

[49] D. M. Mount and S. Arya, “ANN: A Library for Approximate Nearest Neighbor Searching,” http://www.cs.umd.edu/~mount/ANN, 2005.

[50] W. Williams, and J. Lambert, “Multivariate methods in plant ecology: V. Similarity analyses and information-analysis,” Journal of Ecology, vol. 54, no. 2, pp. 427–445, 1966.

[51] C. Blake, E. Keough, and C. J. Merz, “UCI Repository of Machine Learning Database,” http://www.ics.uci.edu/~mlearn/MLrepository.html, 1998.

[52] R. Bouckaert, E. Frank, M. Hall, R. Kirkby, P. Reutemann , A. Seewald, and D. Scuse, “WEKA Manual for Version 3-6-2,” University of Waikato, Hamilton, New Zealand, http://www.cs.waikato.ac.nz/~ml/weka, 2010.