The Pennsylvania State University
The Graduate School

CLUSTERING USING DIFFERENTIAL EVOLUTION

A Thesis in
Computer Science
by
Navdha Sah

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December 2016

The thesis of Navdha Sah was reviewed and approved* by the following:

Thang N. Bui
Associate Professor of Computer Science
Chair, Mathematics and Computer Science Programs
Thesis Advisor

Omar El Ariss
Assistant Professor of Computer Science

Jeremy J. Blum
Associate Professor of Computer Science

Sukmoon Chang
Associate Professor of Computer Science
Graduate Coordinator

Linda M. Null
Associate Professor of Computer Science

*Signatures are on file in the Graduate School.

Abstract

In the past few decades, clustering or cluster analysis has emerged as an essential tool in several fields to help find conceptually meaningful groups of objects that share similar characteristics. It has proven useful in a wide variety of domains such as engineering, computer science, medical science, social science, etc. Clustering is defined as the problem of partitioning an unlabeled data set into groups (or clusters) of similar objects such that objects in the same cluster are more similar to each other than objects in other clusters. The clustering problem is known to be NP-hard. Over the years, much research effort has been made to design efficient clustering algorithms to provide high quality solutions. In this thesis, we give a clustering algorithm, which is based on differential evolution but has additional features, such as local optimization. Experimental results on a total of 17 real-world data sets of various sizes and dimensions show that the algorithm is competitive, fast, efficient, and provides good clustering solutions.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to start by thanking my thesis advisor, Dr. Thang Bui, for his endless support, guidance, brilliance and patience throughout the thesis process. If not for his expert advice and continuous motivation, I wouldn't have learned as much as I did in the field of combinatorial optimization. I will always be grateful to him for helping me cross all hurdles that came along the way. I would also like to thank my thesis committee for their support, encouragement and feedback. Finally, a big thank you to my family and friends for believing in me when I didn't.

Dedicated to my parents,
C M Sah and Jyoti Sah.

Chapter 1

Introduction

The clustering problem is the problem of partitioning a collection of objects into groups called "clusters" such that objects in a cluster are more similar to each other than objects in another cluster. If there is no external information provided with the objects, such as class labels, the problem of clustering becomes an instance of *unsupervised learning*.

Clustering, or cluster analysis, plays an important role in the analysis, description and utilization of valuable information hidden within groups. In general, cluster analysis helps find conceptually meaningful groups of objects that share similar characteristics automatically [42]. Cluster analysis has proven helpful in a wide variety of domains and has been applied in the fields ranging from engineering, computer science, medical science to earth science, life science, and economics. Some interesting applications of clustering are given below [44].

- **Market research:** Cluster analysis is widely used for large-scale segmentation and profiling in marketing, to locate targeted customers and design strategies accordingly for effective customer relationship management [4].

- **Recommender systems:** In recent years, recommender system for online product recommendation or location based services has gained a lot of popularity. To build recommender systems, one of the most popular techniques is collaborative filtering that makes use of the known preferences of a group of

users to make recommendations or predictions of unknown preferences for other users [41]. One of the fundamental tools for collaborative filtering is choosing the right clustering technique to be able to group users/locations with similarities together.

- **Community detection:** Another application of clustering that has witnessed an increasing interest in recent years is community detection in real-world graphs such as social networks (e.g., LinkedIn, Facebook, Twitter, etc.), biological networks, and web graphs [25].

In the past few decades, cluster analysis has also been applied to problems in machine learning, artificial intelligence, pattern recognition, web mining, spatial database analysis, image segmentation, software evolution, anomaly detection, etc. All of these applications clearly illustrate the vital role that clustering plays in the exploration and analysis of unknown data introduced with real world applications.

That being said, the problem of clustering is not easy, and as a consequence, tremendous research efforts have been devoted to designing new clustering algorithms since Karl Pearson's earliest research in 1894 [35]. Effectively, the clustering problem is the problem of finding a clustering that maximizes the similarity of items within the same clusters and at the same time, minimizes the similarity between items in different clusters. The reason why clustering is a difficult problem is due to the following aspects [30]: (1) clustering is NP-Hard, (2) there are no widely accepted theories for clustering that could be applied to every type of data set, and (3) the definition of a cluster is determined by the characteristics of the data set and the understanding of the user and is therefore, quite arbitrary. Over the years, researchers have categorized clustering algorithms into various types such as prototype-based algorithms (K-means) [26], density-based algorithms [12], graph-based algorithms (hierarchical agglomerative clustering) [42], hybrid algorithms [20], etc.

However, for the scope of this thesis, we discuss only the classification of the clustering algorithms based on crisp and fuzzy clustering algorithms. In crisp clustering, an item can only be assigned to one cluster, but in fuzzy clustering, an item may be assigned to more than one clusters with varying degrees of membership in each cluster. We focus on crisp clustering for the rest of the thesis.

As stated earlier, clustering is an optimization problem. The optimization could

focus on different characteristics of a cluster such as its compactness, separation or connectivity. Ideally, the aim of a clustering algorithm would be to optimize the cluster quality based on all of these attributes. Nevertheless, the evaluation of a cluster quality, or cluster validity, can be formally defined as giving *objective* evaluations to clustering results in a *quantitative* way [17]. Basically, a cluster validity index could be used to indicate the quality of a clustering solution. It has been observed that the performance of a clustering algorithm is heavily dependent on the choice of the cluster validity index.

Some of the traditional clustering algorithms such as K-means, employ a greedy search technique to find the optimal clustering. However, such techniques, even though computationally effective, suffer major drawbacks [29], e.g.,

- The number of clusters needs to be specified *a priori* for these algorithms.
- Quality of clustering for these algorithms is evaluated without considering all characteristics of either the data set or the clustering.
- These algorithms are not able to effectively explore the search space, which is the set of all possible clustering, and tend to get stuck in local optima. As such, they fail to find the globally optimal solution.

The algorithm given in this thesis tries to overcome some of these problems, such as prior specification of number of clusters and limited exploration of the search space. Some global optimization tools like Genetic Algorithms (GAs) [2], Simulated Annealing (SA) [38], and Particle Swarm Optimization (PSO) [21] have also been implemented in the past for the same reasons. The issue of not optimizing all characteristics of data set or clustering could be handled by multi-objective clustering algorithms.

In this thesis, we give a clustering algorithm that is based on differential evolution with a few additional functionalities, such as local optimization, to cluster unlabeled data sets. The algorithm is tested on 17 real-life data sets of various sizes and dimensions. Experimental results show that our algorithm is competitive, fast, and efficient, and provides good clustering solutions for all data sets used in the thesis.

The rest of the thesis is organized as follows. Chapter 2 provides a formal definition of the clustering problem, description of the measures used to analyze the

clustering, brief summary of genetic algorithms, differential evolutionary algorithms, and other related works. The Differential Evolution Clustering (DEC) algorithm is discussed in details in Chapter 3. Chapter 4 provides the setup, data sets and the experimental results obtained. The conclusion and future work are given in Chapter 5.

Chapter 2

Preliminaries

In this chapter, we briefly go through the formal definition of clustering and the various similarity measures and clustering validity indexes that are used in the DEC algorithm. A brief overview of genetic algorithms and differential evolution algorithms is also presented in this chapter. In addition, we discuss the existing related work in the literature in the field of clustering.

2.1 Definitions

2.1.1 Clustering problem

Let $X = \{X_1, X_2, \ldots, X_n\}$ be a set of n data items, each having f real-valued features, i.e., $X_i \in \mathbb{R}^f, i = 1, \ldots, n$, and, $X_i = (X_{i1}, \ldots, X_{if})$, where X_{ij}'s are called *features* of X_i.

A *clustering* of X is a partition $\mathcal{C} = (C_1, \ldots, C_k)$ of X for some $k > 1$ such that:

1. $\bigcup_{i=1}^{k} C_i = X$,
2. $C_i \cap C_j = \phi \quad \forall i, j \in \{1, 2, \ldots, k\}, i \neq j$,
3. $C_i \neq \phi \quad \forall i \in \{1, 2, \ldots, k\}$.

If C_i is a cluster in the clustering $\mathcal{C}$, we define the centroid of C_i, denoted by ω_i, as

$$\omega_i = \frac{1}{|C_i|} \sum_{X_j \in C_i} X_j.$$

We also define $\Omega = \{\omega_1, \ldots, \omega_k\}$ as the collection of centroids of the clustering $\mathcal{C}$. To indicate the quality of a clustering $\mathcal{C}$, a function $\mathcal{F}$, called the *validity index*, is used to capture the notion of what a good clustering is. Intuitively, a clustering is considered to be good if items within the same cluster are more similar than items in different clusters. Examples of validity indexes are given in the next few subsections. The clustering problem can now be defined as follows:

Clustering Problem

Input: Data set $X = \{X_1, \ldots, X_n\}$ where $X_i \in \mathbb{R}^f$, a distance function $\mathcal{D}$ and a validity index $\mathcal{F}$.

Output: A clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$ of X such that $\mathcal{F}(\mathcal{C}, \mathcal{D})$ is optimized over all possible clusterings of X.

It is to note that the number of clusters is not passed as an input. This problem is known to be NP-hard [27]. In the following sections, we discuss the validity indexes used to determine quality of a clustering solution. These validity indexes are dependent on the similarity measures used to determine the similarity and dissimilarity of items in clusters. Therefore, we also discuss the different techniques to find similarity between items.

2.1.2 Similarity measure

To obtain a clustering and calculate its fitness, we need to be able to recognize natural groupings of similar items in a data set. Hence, defining an appropriate distance measure plays a fundamental role in finding clusters. The most widely used distance measure to compute similarity between two items is the Euclidean distance. It can be expressed as follows for two f-dimensional items X_i and X_j

$$d(X_i, X_j) = \sqrt{\sum_{p=1}^{f} (X_{ip} - X_{jp})^2}\,.$$

The Euclidean distance measure is a special case of the Minkowsky metric ($\alpha = 2$) [18], which is defined as:

$$d(X_i, X_j) = \left(\sum_{p=1}^{f} (X_{ip} - X_{jp})^{\alpha} \right)^{1/\alpha}.$$

However, Euclidean distance is preferred over Minkowsky metric for clustering data with high dimensionality because under Minkowsky metric, the distance between items becomes larger with growth in dimensionality. Even in Euclidean distance, features with higher magnitudes dominate the distance function and influence the clustering. To avoid this, a commonly used approach is normalization. If the features are normalized over a common range, the problem of large scale features dominating the distance metric is not an issue. In this thesis, we are using Euclidean distance and a normalized distance measure, which we refer to as *MinMax* distance, to calculate the similarity between two data items.

The *MinMax* distance measure makes use of the minimum and maximum range for the features in the data set. It is defined as:

$$d(X_i, X_j) = \sum_{p=1}^{f} \frac{|X_{ip} - X_{jp}|}{\mathtt{MAX}_p - \mathtt{MIN}_p},$$

where X_{ip} and X_{jp} are the p^{th} feature values for items X_i and X_j, and $\mathtt{MAX}_p$ and $\mathtt{MIN}_p$ are the maximum and minimum values for the p^{th} feature across the data set, that is,

$$\mathtt{MAX}_p = \max\{X_{ip} \mid i = 1, \ldots, n\},$$
$$\mathtt{MIN}_p = \min\{X_{ip} \mid i = 1, \ldots, n\}.$$

This measure can be helpful for data sets that have features with significant variations in magnitudes as it prevents a feature from dominating the distance calculation.

2.1.3 Clustering validity indexes

In this section, we discuss the various clustering validity indexes that are used across the literature to analyze the quality of a clustering solution on a quantitative basis.

Generally, a clustering validity index serves two purposes: (1) helps determine the number of clusters, and (2) helps find the best partitioning [8]. The two main aspects to clustering that a validity index should consider are:

1. *Cohesion:* The items in a cluster should be as similar as possible. A low variance of the items in a cluster indicates a good cohesion or compactness of a cluster.

2. *Separation:* Any two clusters should be as distinct as possible in terms of similarity of items. Basically we want the clusters to be well separated.

Apart from that, a clustering validity index should also possess the following attributes [9]:

1. It should require little or no user interaction or specification of parameters.

2. It should be computationally feasible for large data sets as well.

3. It should yield valid results for data of arbitrary dimensionality too.

To obtain a crisp clustering, some of the well known indexes are the DB index [9], the CS index [7], and the PBM index [32]. Since these indexes are optimizing in nature, i.e., maximizing or minimizing the indexes helps us find the optimal clustering, the validity indexes are best suited with optimization algorithms like Genetic Algorithms, Particle Swarm Optimization, etc.

We define $\mathcal{F}$ as the function that calculates the validity index. Given a clustering $\mathcal{C}$ and a similarity measure $\mathcal{D}$, $\mathcal{F}(\mathcal{C}, \mathcal{D})$ returns a real number to express the validity index, or the fitness of $\mathcal{C}$. In what follows, we discuss in detail the validity indices that are used in the thesis.

2.1.3.1 DB Index

DB index [9] evaluates the quality of a clustering $\mathcal{C} = \{C_1, \ldots, C_K\}$ by calculating the largest intra-to-inter cluster spread in $\mathcal{C}$. To do that, intra cluster distance (average distance of all items in a cluster from the centroid) of a cluster and inter cluster distance (distance between two centroids) between two clusters are calculated. We

define S_i as the average distance of all items in a cluster C_i to its centroid ω_i and it can be calculated as follows:

$$S_i = \left[\frac{1}{|C_i|} \sum_{X \in C_i} \mathcal{D}(X, \omega_i)^q \right]^{\frac{1}{q}},$$

where $\mathcal{D}(X, \omega_i)$ is the distance between an item X in C_i and its centroid ω_i, and $q \geq 1$ is an integer that can be independently selected. When $q = 1, S_i$ becomes the average Euclidean distance of vectors in cluster. Similarly, when $q = 2, S_i$ is the standard deviation of the distance of the items in a cluster to their centroid.

We denote by M_{ij} the inter-cluster distance between two centroids ω_i and ω_j, i.e,

$$M_{ij} = \mathcal{D}(\omega_i, \omega_j), \quad i \neq j\,,$$

Let D_i be:

$$D_i = \max \left\{ \frac{S_i + S_j}{M_{ij}} \mid 1 \leq i, j \leq K, i \neq j \right\}.$$

Then, the DB index of a clustering $\mathcal{C}$ is defined as:

$$\mathcal{F}_{DB}(\mathcal{C}, \mathcal{D}) = \frac{1}{K} \sum_{i=1}^{K} D_i\,,$$

where K is the number of clusters. It is to note that smaller value of the DB index indicates better clustering composed of compact and separated clusters. Therefore, the clustering problem turns into a minimization problem when the DB index is used as the validity index.

2.1.3.2 CS index

This index is a measure of the ratio of the sum of within-cluster scatter to between cluster separation and has the same idea as the DB index. The CS index is considered more efficient in tackling clusters of different densities and/or sizes. However, it is computationally more expensive as compared to the DB index.
Let the distance between two items X_i and X_j be denoted by $\mathcal{D}(X_i, X_j)$, then the CS index for a clustering $\mathcal{C}$ can be defined as:

$$\mathcal{F}_{CS}(\mathcal{C},\mathcal{D}) = \frac{\frac{1}{K}\sum_{i=1}^{K}\left[\frac{1}{|C_i|}\sum_{X_i\in C_i}\max_{X_j\in C_i}\{\mathcal{D}(X_i,X_j)\}\right]}{\frac{1}{K}\sum_{i=1}^{K}\left[\min_{j\in K, j\neq i}\{\mathcal{D}(\omega_i,\omega_j)\}\right]} = \frac{\sum_{i=1}^{K}\left[\frac{1}{|C_i|}\sum_{X_i\in C_i}\max_{X_j\in C_i}\{\mathcal{D}(X_i,X_j)\}\right]}{\sum_{i=1}^{K}\left[\min_{j\in K, j\neq i}\{\mathcal{D}(\omega_i,\omega_j)\}\right]},$$

where K is the number of clusters in $\mathcal{C}$.

A larger CS index would mean that either the clusters are not compact or not well separated. Therefore, smaller values of CS index indicate better clustering.

2.1.3.3 PBM index

The PBM validity index is also based on within-cluster and between-cluster distances. It is a maximization criterion, unlike the DB and the CS indexes, and can be defined as follows [32]:

$$\mathcal{F}_{PBM}(\mathcal{C},\mathcal{D}) = \left(\frac{1}{K}\times\frac{E}{E_K}\times D_K\right)^2,$$

where $K = |C|$. Let

$$\omega_X = \frac{1}{n}\sum_{i=1}^{n} X_i\,,$$

where $X = \{X_1,\ldots,X_n\}$ is the data set. Then,

$$E = \sum_{i=1}^{n}\mathcal{D}(X_i,\omega_X)$$

$$E_K = \sum_{k=1}^{K}\sum_{x\in C_k}\mathcal{D}(x,\omega_k)\,,$$

and

$$D_K = \max\{\mathcal{D}(\omega_i,\omega_j)\mid 1\leq i,j\leq K\}\,.$$

Algorithm: SimpleGA(P)

Initialize population P with random candidate solutions
Evaluate each candidate
repeat
    Select parents from P
    Crossover pairs of parents to produce offspring
    Mutate offspring
    Evaluate the offspring
    Replace an individual in P by comparing an existing individual with the new offspring and form a new population, P' for the next generations
until *Termination Condition* is satisfied;

Figure 2.1: A simple genetic algorithm

E is the distance of all items in the data set from ω_X. E_K is the total intra cluster sum for all clusters present and D_K is the maximum distance between the farthest centroids in the clustering.

2.2 Background

In this section, we provide a general understanding of the working of Genetic Algorithms (GA). This is followed by a discussion of Differential Evolutionary (DE) algorithms that inspired the DEC algorithm in the thesis.

2.2.1 Genetic algorithms

Genetic Algorithms (GA) were first introduced by John Holland at the University of Michigan in 1975 [16]. A GA consists of problem encoding, selection methods, and genetic operators such as crossover and mutation. There are typically four stages in a GA: (1) selection, (2) recombination (crossover), (3) mutation, and (4) replacement. Algorithm 2.1 is an example of a general genetic algorithm [11]. The first step is to generate a population of random solutions. The next step is to select two parents to perform recombination to generate an offspring. The selection function could vary from being a random selection process to a fitness proportional scheme or a ranking selection or tournament selection scheme [11]. There are various crossover methods

that can be used. Some common crossover methods are binomial crossover, k-point crossover, and uniform crossover. Once one or more offspring are generated, they are mutated using one of the various mutation operations. Finally, a mutated offspring is compared with its parent in the population. The offspring can replace its parent if it has a better fitness and be a part of the population for the next generations. This whole process is repeated until the stopping criterion is met. In the end, the best individual of the population is returned. This individual represents the best solution discovered by the algorithm.

The GA discussed above is a very simple algorithm that can be improved further to find better solutions. This could be achieved by introducing a local optimization method before replacement in the algorithm. The local optimization method would ensure that the offspring being introduced in the population is as good as it could be. It is also to note that the GA in Fig. 2.1 is referred to as a *steady-state* GA, as only one individual gets replaced in an iteration.

Differential evolution (DE) algorithms are a type of GA. The major difference between GAs and DEs is in terms of the offspring generation. The DE algorithm is discussed in detail in the next section.

2.2.2 Differential evolution algorithms

Differential evolution (DE) is a fast and robust optimization strategy that uses a stochastic, population-based search methodology. It uses the same idea as a genetic algorithm of maintaining a population of candidate solutions that is evolved to obtain better solutions. However, DE differs from traditional genetic algorithms mainly in the way new offspring solutions are generated. A candidate in the population, which we will refer to as individual I from here on, uses a floating-point representation and can be represented as an n-dimensional vector:

$$I_i = (I_{i1}, I_{i2}, \ldots, I_{in}),$$

where I_i is the i^{th} member of the population at a given generation.

In DE offspring solutions are linear combinations of existing solutions. To generate an offspring I_c for an individual I_p, three random unique individuals from the same generation are sampled by DE, namely, I_x, I_y and I_z. DE then computes a

component of the offspring by calculating the difference of I_x and I_y. Next, it scales the difference by a factor σ (usually $\sigma \in [0, 1]$) and finally adds it to I_z.

Generally, traditional DE uses binomial crossover to construct the offspring, that is, for each component of the offspring, there is a coin toss that determines whether the component would be computed or copied over from the parent individual I_p based on a crossover probability. Thus, I_c is computed as follows:

$$I_c = \begin{cases} I_n + \sigma(I_l - I_m) & \text{if } \mathcal{U}\text{Random}(0,1) < \eta \\ I_p & \text{otherwise,} \end{cases}$$

where $\eta \in [0, 1]$ is the crossover probability and $\mathcal{U}\text{Random}(0, 1)$ denotes a random number selected from $[0, 1]$.

Once an offspring is successfully generated, its fitness is calculated and then it is compared against its parent. Normally, the replacement scheme of the algorithm is simple, meaning if the offspring is better than the parent, the parent is replaced by the offspring. However, there could be different ways to handle the replacement scheme for a DE algorithm, for instance, a worse offspring could replace the parent with a certain probability, thereby allowing more exploration of the search space.

2.3 Related Work

In the past few years, tremendous research has been done to cluster complex data sets using evolutionary computing techniques. Most of the existing techniques take in the number of classes as an input instead of determining it through the algorithm. This is a major drawback, since in a new data set, the number of classes may be unknown or impossible to determine. Fortunately, there has been significant work done in this area to determine the clustering for an unlabeled data set. In this section, we review a small portion of the clustering literature by focusing on the evolutionary computing approaches and their application in the clustering problem. The approaches discussed in the beginning of the section require the number of clusters as an input. Later, we talk about algorithms that either tried to find the optimal number of clusters or did not require it to be provided as an input. Finally, we describe our methodology and provide a short overview of the work done in this

thesis.

We start our discussion with one of the most widely used algorithms in clustering: the K-means algorithm. This algorithm takes as input a number K, and starts with K centroids that are either randomly selected or derived using some *a priori* information provided. Each item is clustered with the closest centroid. After all items are clustered, the centroids get updated. Based on these updated centroids, all items are clustered again until some convergence criteria is met.

Even though the K-means algorithm is easy to implement, it has the following drawbacks:

- The number of clusters, K, has to be provided by the user as part of the input.
- Performance of the algorithm is greatly influenced by the data set.
- The algorithm is greedy in nature and therefore dependent on the initial selection of centroids.

Due to these disadvantages, there have been other computing techniques explored for the clustering problem. For the remaining section, we provide a summary of the most important approaches studied over time for clustering using evolutionary computation algorithms.

One of the first applications of GAs to obtain clustering was introduced by Raghavan and Birchand in [36]. The main idea in their approach was to use a genetic encoding that directly allocated n patterns to K clusters, such that each candidate solution consisted of n genes. These genes were represented as a range of integer values in the interval $[1, K]$, where the integer values represented the cluster the item was supposed to belong to. For example, for $n = 5$ and $K = 3$, the encoding "11322" would allocate the first and second objects to cluster 1, the third object to cluster 3, and the fourth and fifth objects to cluster 2. Based on this problem representation, the GA tried to find the optimal partition according to a fitness function that measured the quality of the partition [8]. This algorithm outperformed K-means in the analysis of simulated and real data sets. The main drawback of the algorithm was the redundancy in the representation scheme; for instance, "11322" and "22311" would represent the same grouping solution $(\{1, 2\}, \{3\})$. This problem was tackled by Falkenauer [13] by introducing the representation of the group labels as additional genes in the encoding and by applying *ad hoc* evolutionary operators on them.

Another GA approach to obtain clustering was proposed by Bandyopadhyay *et al.* [2]. The idea of the algorithm was to encode cluster-separating boundaries. GAs were used to separate the clusters by determining hyperplanes as decision boundaries for dividing the attribute feature space. Srikanth *et al.* [39] proposed another approach that encoded the center, extend, and orientation of an ellipsoid for every cluster to be able to study the shape of the variance for each cluster.

Some researchers have used hybrid clustering algorithms to determine the clustering for a given data set. Krishna and Murty [22] introduced a GA with direct encoding of object cluster association like [39] but used K-means to determine quality of the encodings. Similarly, Kuo *et al.* [23] used ART2 neural network to determine the initial solution and then used genetic K-means to obtain the final solution.

The problem of finding the optimal number of clusters in large data sets has also been investigated by many researchers. Lee and Antonsson [24] used an Evolutionary Strategy (ES)-based method to dynamically cluster a data set. The proposed ES implemented variable-length individuals to search for both centroids and optimal number of clusters. Another approach to dynamically classify a data set using evolutionary algorithm [14] was implemented by [37], where two fitness functions were simultaneously optimized: one to give the optimal number of clusters and the other to give a proper cluster centroid. Bandyopadhyay *et al.* [3] devised a variable string-length genetic algorithm to tackle the dynamic clustering problem using a single fitness function [8]. More recently a hybrid evolutionary model has been proposed for K-means clustering that uses meta-heuristic methods to identify and select good initial centroids to improve the cluster quality in [19].

In more recent years, two promising approaches emerged for numerical optimization problems, namely, Particle Swarm Optimization (PSO) and DE. Paterlinia and Krink [34] used a DE algorithm and compared its performance with a PSO and a GA algorithm over the partitional clustering problem. Their work focused on semi-supervised clustering with a preassigned number of clusters. In [31], Omran *et al.* proposed an image segmentation algorithm based on the PSO. Das *et al.* proposed a DE-based algorithm for automatic clustering of real-life data sets and images in [8]. Another recent DE-based algorithm with heterogeneous influence for solving complex optimization problems was proposed by Ali *et al.* in [1].

For this thesis, we have implemented a DE algorithm to find clustering for a given

data set. The algorithm is not provided with the number of clusters for the data set as an input. We take the idea of introducing K number of centroids to get an initial clustering from the algorithm given in [8]. Furthermore, we introduce techniques to explore all possible clusterings for the data set and provide local optimization techniques to improve the clustering. Finally, we have compared our algorithm with the algorithm in [8] and, in turn, the algorithms referred to in their paper. The following performance metrics have been used in the comparative analysis: (1) accuracy of clustering results, (2) speed of convergence, and (3) robustness. We conducted the tests on the same data sets used in [8] and some other larger data sets that were not covered by their paper. In addition, we have also experimented and compared results for different validity indexes. In the next chapter, we discuss our algorithm in depth.

Chapter 3

DE Clustering Algorithm

In this chapter we describe our algorithm, called Differential Evolution Clustering (DEC) algorithm, for the clustering problem. Our algorithm is based on the classical differential evolution algorithm but with a number of additional features including local optimization. DEC can be divided into two main stages. The first stage is the initialization stage. This is the entry point of the algorithm, where initialization of the individuals in a population is done. The next stage is the evolution stage. In this stage, the population goes through a number of cycles. Within each cycle, there are two phases: an exploration phase and an exploitation phase, spanning over a fixed number of generations. The first 70% of the generations in a cycle make up the exploration phase. The remaining 30% of the generations make the exploitation phase. In the exploration phase, the algorithm tries to explore all possible clusterings for the data set, whereas in the exploitation phase, the algorithm exploits its neighboring solutions in search of a better solution. Additionally, before another cycle of exploration and exploitation starts, the individuals in the population are perturbed to prevent early convergence of the algorithm. Finally, DEC returns the best solution in the population. These stages are discussed at length in the following sections but the overall DEC algorithm is given in Fig. 3.1.

3.1 Initialization

The DEC algorithm maintains a population of solutions. More precisely, the population consists of encodings of solutions. Each solution, i.e., the clustering of the data

Algorithm: **DEC**$(X, \mathcal{F}, K_{max}, K_{min})$
Input: Data set X, validity index $\mathcal{F}$, and K_{max},K_{min} are maximum and minimum allowed clusters in a clustering, respectively
Output: A clustering $\mathcal{C}$ of X

```
1  P ← φ      // current generation population
2  g ← 0      // current generation
3  cycle ← 0      // current cycle
4  P_new ← φ      // next generation population
5  D ← D_E      // Distance function used in F set to Euclidean distance
6  stopDE ← false
7  noImprovementCtr ← 0
8  Setup(X, P, K_max, K_min, D)    // Setting up population
9  while cycle ≤ numCycles and stopDE is false do
10     noImprovementCtr ← 0
11     while g ≤ numGenerations and stopDE is false do
12         foreach Individual I_p in P do
13             Randomly select I_x, I_y, I_z unique individuals from P
14             I_c ← Crossover(I_p, I_x, I_y, I_z, g, D, X)    // Generate offspring
15             I_c ← LocalOpt(I_c, g, F, D)
16             I' ← Replacement(I_p, I_c, g, F, D)
17             P_new ← P_new ∪ {I'}
18         P ← P_new
19         P_new ← φ
20         g++
21         if cycle > 0.8× numCycles and g > 0.7× numGenerations then
22             if fitness of best individual in P and P_new are the same then
23                 noImprovementCtr++
24                 if noImprovementCtr ≥ noImprovThreshold then
25                     stopDE ← true
26                     break
27     Perturb(P, cycle)
28     cycle++
29     if cycle > 0.7× numCycles then
30         D ← D_MM // Change distance from Euclidean to MinMax distance
31 Find best individual I_best in the population P
32 return Clustering represented by I_best
```

Figure 3.1: The Differential Evolution Clustering (DEC) algorithm

set, is encoded by a set of centroids. Each cluster in the clustering has one of these centroids at its center. We think of the encoding, i.e., the set of centroids, as the genotype and the corresponding clustering, i.e., the solution, as the phenotype. This is analogous to the genotype to phenotype mapping in natural evolution. The population is then initialized with a set of such encodings or individuals. Moreover, the fitness of each individual is determined based on the clustering solution it provides. In this section, we discuss the representation of an individual in the population and the mapping of the encoding of an individual to its clustering solution.

3.1.1 Encoding

It is important to have a good encoding of the solution for the individuals that comprise the population. In DEC, an individual's encoding is a triple,

$$I = (\Omega, \mathcal{T}, \mathcal{C}),$$

where $\Omega = \{\omega_1, \ldots, \omega_K\}$ is the set of centroids, $\mathcal{T} = \{t_1, \ldots, t_K\}, 0 \leq t_i \leq 1$, is the set of thresholds, and $\mathcal{C} = \{C_1, \ldots, C_K\}$ is the clustering, i.e., a collection of clusters.

Every individual has a maximum of K centroids and K threshold values. The threshold values determine whether a centroid will be considered for finding the clustering of data set X. If the threshold for a centroid falls between $[0, 0.5)$, it is considered inactive, otherwise it is active. Basically, an *active* centroid is a centroid that takes part in clustering of X. Items that are closest to the active centroids are then grouped together to form clusters. Therefore, each active centroid ω_i has a non-empty cluster C_i that consists of a collection of similar items. However, if ω_i is inactive, then C_i is an empty cluster. The clustering $\mathcal{C}$ is the solution corresponding to individual I.

In the DEC algorithm, all offspring are created using only Ω, that is, using only the genotype as in a traditional DE algorithm. However, our algorithm differs slightly from the traditional algorithms in that the phenotype, or the clustering solution, is also a part of the encoding. The reason behind including the phenotype in the encoding is that DEC algorithm also manipulates the clustering or the phenotype along with the genotype to create new individuals. In the clustering problem, if we only manipulate the genotype, we might not see the desired change in the phenotype.

For instance, even though the genotype, i.e., the centroids for a new offspring could be very different from the parent, the equivalent phenotype or clustering could be very similar. Alternatively, a small change in the genotype could result in the phenotype changing significantly. Both of these cases could be problematic and undesirable when trying to explore or exploit all possible solutions. Therefore, we have decided to work with both the genotype and the phenotype together.

In addition to the notations defined in earlier chapters, the following notations are used throughout the thesis.

- $\Omega(I)$ represents the set of centroids for an individual I.
- $\mathcal{C}(I)$ represents the clustering solution for I.
- $\mathcal{T}(I)$ represents the threshold values for centroids in I.

3.1.2 Fitness calculation

The algorithm for mapping the genotype to phenotype is shown in Fig. 3.2. After the data has been partitioned and a clustering is obtained, the next step is to calculate the fitness of an individual. However, before doing that we need to determine that an individual is valid and ready for fitness calculation. We define a valid individual as an individual that has: (1) at least the minimum number of required clusters in the clustering solution, and (2) each cluster must have a prescribed minimum number of items. A clustering with an insufficient number of clusters is considered invalid in our algorithm as it does not promote exploration of the search space. The prescribed minimum has an effect on the performance of the algorithm as some validity indexes have a tendency to direct the algorithm towards clusterings that have the smallest possible number of clusters. Having a minimum prescribed number of clusters larger than the obvious choice of 2 allows the algorithm to explore more feasible solutions. Furthermore, a clustering with very tiny or empty clusters is also considered invalid. The presence of such a cluster indicates that there is an active centroid in the genotype that is not close to sufficiently many of the items in the data set. Since DEC is an optimization algorithm, it would be inappropriate to let such a bad centroid be propagated over the generations. In this thesis, a *tiny cluster* is defined as a cluster that has less than ten items.

Algorithm: ComputeClustering$(I, X, \mathcal{D})$

Input: Individual I, the data set X and a distance measure $\mathcal{D}$

Result: A modified individual I for which the clustering $\mathcal{C}$ has been computed or recomputed

```
1  foreach ω_i ∈ Ω(I) do
2      C_i ← φ
3  foreach item x in X do
4      Find an active ω_i ∈ Ω(I) that is closest to x
5      Add x to cluster C_i
6      Recompute ω_i when |C_i| = 0 (mod ξ)    // where z is an integer
```

Figure 3.2: Compute clustering

There are a number of ways to deal with an empty or tiny cluster. We could either deactivate the centroid of the tiny cluster after redistributing its items, or we could move the bad centroid to a better position such that it attracts more items, or we could forcefully add items to the tiny cluster till it is no longer tiny. In the DEC algorithm, we use a hybrid approach to deal with tiny clusters. We always deactivate an empty cluster and either redistribute items from a tiny cluster or add new items to the tiny cluster. The algorithm for cleaning up an individual before calculating its fitness is given in Fig. 3.3. An individual's fitness is defined in terms of the quality of the clustering it represents. We have already discussed the different cluster validity indexes that can be used to measure a cluster quality in Chapter 2. Thus, the fitness of an individual in the population is the computed cluster validity index for the clustering it represents. On that account, the DEC algorithm tries to optimize the partitioning by minimizing or maximizing the validity index.

We note that when items are moved from one cluster to another, it is a good practice to recompute the centroids for the clusters involved. By doing that, we ensure that a centroid accurately represents its cluster's center. Nevertheless, in the *GrowCluster* algorithm given in Fig. 3.3, we do not recompute centroids of clusters from which the items are removed. This is an implementation detail to save some computation time without affecting the quality of the solution adversely. Since only a small number of items are being moved from the cluster, it is safe to assume that

Algorithm: CleanIndividual$(I, \mathcal{D})$

Input: An individual I and a distance measure $\mathcal{D}$
Result: Eliminate empty and tiny clusters in I

```
1  foreach cluster C_i in I do
2      if |C_i| = 0 then
3          Set t_i ∈ [0, 0.5)    // Deactivate centroid ω_i
4      else if C_i is tiny then
5          Redistribute(I, C_i, D) with probability μ or,
6          GrowCluster(I, C_i, m, D) with probability 1 − μ
           // where μ increases with time and m denotes the number of
              items to be added to C_i
```

Algorithm: Redistribute$(I, C_i, \mathcal{D})$

Input: An individual I, a tiny cluster C_i belonging to I and a distance measure $\mathcal{D}$
Output: Redistribute items in C_i to other clusters in I

```
1  foreach item x in cluster C_i do
2      Find centroid ω_j that is closest to x
3      Move x from C_i to C_j
4      Recompute ω_j
5  Set t_i ∈ [0, 0.5)    // Deactivate ω_i
```

Algorithm: GrowCluster$(I, C_i, m, \mathcal{D})$

Input: An individual I, a tiny cluster C_i, an integer m and a distance measure $\mathcal{D}$
Output: Move m items from other clusters in I to C_i

```
1  Let x be the item in C_i that is closest to its centroid ω_i
2  Find m items in other clusters that are closest to x
3  Move these items to C_i
4  Recompute centroid w_i
```

Figure 3.3: Cleaning up an individual

its centroid would not change significantly; unless the cluster is small in size and becomes tiny after items are moved out of it. However, the algorithm would handle this case, as this cluster would now become a tiny one and would need to be fixed. The algorithm to calculate the fitness of an individual is summarized in Fig. 3.4. It

---

Algorithm: Fitness$(I, \mathcal{F}, \mathcal{D})$

Input: An individual I, validity index $\mathcal{F}$, and distance measure $\mathcal{D}$
Output: Fitness of I based on its clustering $\mathcal{C}(I)$

```
1  ComputeClustering(I)
2  CleanIndividual(I)
3  return F(C(I), D)
```

Figure 3.4: Computing the fitness of an individual

involves computing the clustering for a set of active centroids and cleaning up the clusters, if needed.

3.1.3 Population setup

The population in DEC comprises of a fixed number of valid individuals. To setup the population, we use the algorithm outlined in Fig 3.5. Every individual has K centroids and threshold values denoted by:

$$\Omega = \{\omega_1, \ldots, \omega_K\},$$

where $\omega_i = \{\omega_{i1}, \ldots, \omega_{if}\}$, $\texttt{MIN}_j \leq \omega_{ij} \leq \texttt{MAX}_j$, are the centroids and $\texttt{MAX}_p$ and $\texttt{MIN}_p$ are the maximum and minimum values for the p^{th} feature across the data set, and

$$\mathcal{T} = \{t_1, \ldots, t_K\}, 0 \leq t_i \leq 1,$$

are the thresholds. Basically, we ensure that all centroids are initialized within the search space of the problem. As noted before, a centroid ω_i is considered *active* if the corresponding threshold is greater than 0.5; otherwise it is considered *inactive*. Using this scheme, an individual generated may not have enough active centroids to be considered valid. In that case, we randomly select inactive centroids and make

them active. The choice of minimum and maximum K is kept constant throughout the algorithm and limits the algorithm to look for solutions within the range specified, that is, any clustering obtained would not have more than K clusters or fewer than a specified number of clusters. The size of the population P is determined as follows:

$$N = \lambda \times f$$

where N is the population size, f is the number of features and λ is a constant positive integer that is chosen at the beginning of the algorithm.

While setting up the initial population, we make sure that all individuals in the population are valid. However, when the population is evolved, keeping an invalid solution in the population could make the algorithm more flexible in exploring the search space. Thus, an invalid individual or offspring is assigned a bad fitness so that it can participate in exploring the search space but it is not expected to survive for long.

The initialization stage ends after the setup is done. The next stage is the evolution stage where the population is evolved. We start by generating an offspring using crossover. The offspring then undergoes local optimization before it is compared against its parent to be considered as a member of the next generation population. All of these algorithms are discussed in detail in the remaining sections.

3.2 Crossover

The creation of an offspring in the DEC algorithm is done using binomial crossover. A parent I_p and three other unique individuals I_x, I_y, I_z, called donors, are chosen randomly from the population in order to generate a new individual. From these individuals, the genotype for an offspring I_c, that is Ω_c, and its threshold $\mathcal{T}_c$ is calculated component wise using the algorithm in Fig. 3.6. A crossover probability η, scaling factor σ_t for threshold, and scaling factor σ_ω for the centroids are defined in the algorithm to help with exploration and exploitation. In the DEC algorithm, we change the crossover probability based on the generation. The crossover probability is set at 0.8 at the beginning of the exploration phase and reduced to 0.2 in the exploitation phase. It can be clearly seen from the algorithm that when the

Algorithm: Setup$(X, P, K_{max}, K_{min}, \mathcal{D})$

Input: An empty population P, the data set X, maximum and minimum number of clusters allowed, and distance measure $\mathcal{D}$

Output: A filled population P

$N \leftarrow \lambda \times f$
Let R_1, R_2, R_3 be the intervals [0.9, 1], [0.3, 1] and [0,1], respectively
foreach *activation threshold* R *in* $\{R_1, R_2, R_3\}$ **do**
    Generate $N/3$ individuals using `GenerateIndividual(`$R, X, \mathcal{D}$`)`
    Add the individuals generated to P
return P

Algorithm: GenerateIndividual$(R, X, \mathcal{D})$

Input: A range R that determines the threshold values for $t \in \mathcal{T}$, the data set X, and distance measure $\mathcal{D}$

Output: A newly created individual I

`// Create` $\Omega = \{\omega_1, \ldots, \omega_K\}$`, where` $\omega_i = \{\omega_{i1}, \ldots, \omega_{if}\}$
for $i = 1$ *to* K **do**
    for $j = 1$ *to* f **do**
        $\omega_{ij} \leftarrow \mathcal{U}\text{Random}(\texttt{MIN}_j, \texttt{MAX}_j) < \eta$
`// Create` $T = \{t_1, \ldots, t_K\}$
for $i = 1$ *to* K **do**
    $t_i \leftarrow$ random number $\in R$
$I \leftarrow (\Omega, \mathcal{T}, \phi)$
`ComputeClustering(`$I, X, \mathcal{D}$`)` `// generates a clustering for` I
`CleanIndividual(`$I, \mathcal{D}$`)`
return I

Figure 3.5: Generating initial population

Algorithm: Crossover$(I_p, I_x, I_y, I_z, g, \mathcal{D}, X)$

Input: Parent individual I_p, donor individuals I_x, I_y, I_z, current generation g, distance measure $\mathcal{D}$ and data set X

Output: An offspring $I_c = (\Omega_c, \mathcal{T}_c, \mathcal{C}_c)$

```
// Create offspring I_c = (Ω_c, T_c, C_c)
// Create T_c = {t_1^c, ..., t_K^c}
for i = 1 to K do
    if URandom(0,1) < η then
        t_i^c ← t_i^x + σ_t(t_i^y − t_i^z)
    else
        t_i^c ← t_i^p
// Create Ω = (ω_1^c, ..., ω_K^c) and ω_i^c = {ω_i1^c, ..., ω_if^c}
for i = 1 to K do
    σ'_ω ← σ_ω(g)
    if URandom(0,1) < η then
        for j = 1 to f do
            if g > 0.7× total number of generations then
                σ ← σ'_ω
            else
                σ ← σ_ω(g)
            ω_ij^c ← ω_ij^x + σ(ω_ij^y − ω_ij^z)
    else
        ω_i^c ← ω_i^p
I_c ← (Ω_c, T_c, φ)
ComputeClustering(I_c, X, D) // generates a set of clusters for I_c
CleanIndividual(I_c, D)
return I_c
```

Figure 3.6: Crossover

crossover probability is high, the centroids as well as the threshold in the offspring would be computed from the donors rather than being copied over from the parent. This helps with the exploration of the search space as the offspring would be likely created further away from its parent. When the probability is reduced, the offspring would have more components from the parent and thus, would help in exploiting the neighboring solutions.

We have also defined two types of scaling factors to help with exploration and exploitation. The scaling factor σ_t for the threshold is kept constant throughout the generations, whereas the scaling factor for the centroid σ_ω is calculated as follows.

$$\sigma_\omega(g) = \sigma_{base}(g) + (\zeta(g) \times (\mathcal{U}\text{Random}(0,1) - 0.5)) \tag{3.1}$$

where, ζ and σ_{base} change based on the generation. σ_{base} is reduced in a step-wise fashion through the generations, whereas ζ is kept constant throughout the exploration phase and is reduced at the beginning of the exploitation phase. These values are then reset at the beginning of every cycle. Equation. 3.1 shows how stochastic variations are allowed in the amplification of the difference vector based on the generation. The scaling factor is either kept the same for all features of a centroid or varied across the features. The idea is that we want to allow more exploration in the earlier generations by amplifying all features in the centroids with the same magnitude. However, towards the exploitation phase, we want to focus on the effect of modifying a subset of features within a centroid. Therefore, in the algorithm given in Fig. 3.6, it can be seen that σ_ω is either calculated once for every centroid in I_c or calculated repeatedly for every feature of a centroid.

3.3 Local Optimization

After we get a new individual from crossover, we want to improve it before comparing it against the parent. On that account, we discuss some local optimization techniques introduced in the DEC algorithm to modify an offspring. There are three different techniques implemented in the thesis, namely, *BreakUp*, *Merge* and *Scatter* for local optimization. The *BreakUp* algorithm given in Fig. 3.8 involves breaking up a large cluster to form smaller, more compact clusters which are added back to the individual

in place of the large cluster. The *Merge* algorithm given in Fig. 3.9 combines the two closest clusters in an individual if merging them improves the fitness of the individual. The *Scatter* algorithm given in Fig. 3.10 redistributes items from the smallest clusters in an individual, if getting rid of them improves the fitness of the individual. The *BreakUp* algorithm helps with the exploration as it introduces more centroids in the individual. However, the small clusters created in this step might not be good simply because of the way *BreakUp* operates (or the order in which the centroids are created in *BreakUp*). Thus, *Merge* and *Scatter* techniques are introduced in the exploitation phase to get rid of these not so good small clusters. As mentioned earlier, we accept the individual after *Merge* or *Scatter* only if the fitness is improved. The algorithm is summarized in Fig. 3.7.

In DEC, a large cluster is a cluster that has more than 40% of the total number of items in the data set. Logically, if a cluster is that big, it could be either because of the characteristic of the data set, where a large number of items are similar, or due to the placement of the centroids in the individual. If only one active centroid in $\Omega(I)$ is close to the items in the data set, it is quite likely that most of the items would get grouped in that cluster. It is easy to see that this might not be the optimal partitioning for the data set. Therefore, we use the *BreakUp* scheme to find natural groupings within the large cluster. The term `avgDist`(C_{new}) defined in the algorithm given in Fig. 3.8 denotes the average of distances from the items in cluster C_{new} to its centroid, and `maxDist`(C_{new}) is the distance of the farthest item in C_{new} to its centroid.

We already know that *BreakUp* introduces new centroids in the individual. However, this could cause an item in one cluster to now be in another cluster based on its closeness to their respective centroids. Therefore, to capture this, we regroup all the items that were not part of the large cluster. Once the clustering has been recomputed after *BreakUp*, depending on the generation the algorithm is in, the offspring is either ready to compete against its parent or improved further by *Merge* and *Scatter*. As mentioned earlier, the main idea behind the *Merge* algorithm is to combine clusters that might be close to each other but partitioned only because of the presence of an extra centroid. If the quality of the solution is improved by combining together the closest two clusters, then the algorithm gets rid of the extra centroid and computes a new centroid for the merged cluster. This scheme can be

Algorithm: LocalOpt$(I, g, \mathcal{F}, \mathcal{D})$

Input: Individual I, current generation g, validity index $\mathcal{F}$ and distance measure $\mathcal{D}$
Output: Returns a modified individual I with improved fitness

```
1  C_tabu ← φ
2  I_dup ← I
3  while a large cluster C_large exists such that C_large ∈ C(I_dup) but C_large ∉ C_tabu do
4      X_pool ← φ
5      {C_tiny, C_big} ← BreakUp(I_dup, C_large, D)
6      if C_tiny or C_big ≠ φ then
7          Move all items in C_tiny to X_pool
8          foreach cluster C_i that is not large in I_dup do
9              Move all items in C_i to X_pool
10     else
11         C_tabu ← C_tabu ∪ {C_large}
12         continue;
13     foreach new cluster C_new in C_big do
14         Add ω_new to I_dup
15     foreach item x in X_pool do
16         Find closest centroid w_i for x in I_dup
17         Add x to C_i in I_dup
18     CleanIndividual(I_dup)
19     if g > 0.7× total number of generations then
20         Merge(I_dup, F, D)
21         Scatter(I_dup, F, D)
22     if Fitness(I_dup, F, D) is better than Fitness(I, F, D) then
23         return I_dup
24     else
25         return I
```

Figure 3.7: Local optimization

Algorithm: BreakUp$(I, C', \mathcal{D})$

Input: An individual I, the largest cluster C' and the distance measure $\mathcal{D}$
Output: A set of new clusters that are formed by breaking up the large cluster C'

```
1  C_new ← φ
2  C_set ← φ
3  ω_new ← ω'   // ω' is the centroid for C'
4  while C' is not empty do
5      Let x be the item in C' that is closest to ω_new
6      if d(x, ω_new) < β × max{avgDist(C_new), maxDist(C_new)} then
7          Move x from C' to C_new
8          Recompute ω_new
9      else
10         C_set ← C_set ∪ {C_new}
11         C_new ← φ
12         ω_new ← x   // the item becomes the current centroid
13 C_tiny, C_big ← φ
14 foreach C ∈ C_set do
15     if C is tiny then
16         C_tiny ← C_tiny ∪ {C}
17     else
18         C_big ← C_big ∪ {C}
19 return {C_tiny, C_big}
```

Figure 3.8: Break up a large cluster into smaller ones

Algorithm: Merge$(I, \mathcal{F}, \mathcal{D})$

Input: An individual I, validity index $\mathcal{F}$ and distance measure $\mathcal{D}$

Result: Clusters that are closest to each other in I are combined if the fitness is improved

```
1  successCounter ← 0
2  foreach C ∈ C(I) do
3      Mark C as unused
4  while successCounter < ψ do
5      Let C_i and C_j be the two closest clusters in I such that C_i or C_j is unused
       // D(ω_i, ω_j) is minimum among all pairs of centroids in I
6      Mark C_i and C_j as used
7      Let C_new ← C_i ∪ C_j and I' ← I − (C_i ∪ C_j) ∪ C_new
8      if Fitness(I', F, D) is better than Fitness(I, F, D) then
9          I ← I'
10         successCounter++
```

Figure 3.9: Merging closest clusters together

helpful in scenarios when the *BreakUp* in the algorithm causes too much division and ends up partitioning items that might still be more similar to each other as compared to the rest.

Similarly, the *Scatter* algorithm helps in disposing of the extra clusters that *BreakUp* might end up creating. The idea is to find the smallest cluster in an individual and redistribute its items to other larger clusters to see if the fitness can be improved. This scheme would be helpful when the presence of an additional centroid is affecting the quality of a solution. However, if the centroid is an isolated one that is grouping together a small bunch of outliers in the data set, the fitness would not be improved by removing this centroid. Our algorithm takes care of this situation and allows deactivation of a small centroid only when the overall fitness is improved. Furthermore, the *Merge* algorithm allows at most ψ pair of clusters to be merged. Similarly, the *Scatter* algorithm redistributes items of at most ψ small clusters. This scheme helps prevent the individual from getting pushed into a local optima. Finally, depending on whether these techniques improve the fitness or not, the unmodified or modified offspring is returned.

Algorithm: **Scatter**$(I, \mathcal{F}, \mathcal{D})$

Input: An individual I, validity index $\mathcal{F}$ and distance measure $\mathcal{D}$
Result: Items in smallest clusters are redistributed in I if fitness is improved.

```
1  C_tabu ← φ
2  successCounter ← 0
3  while successCounter < ψ and |C_tabu| < |C(I)| do
4      I_dup ← I
5      Find the smallest cluster C_min in C(I) − C_tabu
6      Redistribute(I_dup, C_min, D)
7      if Fitness(I_dup, F, D) is better than Fitness(I, F, D) then
8          I ← I_dup
9          successCounter++
10     else
11         C_tabu ← C_tabu ∪ {C_min}
```

Figure 3.10: Redistribute items in small clusters if that improves fitness

3.4 Replacement

After crossover and local optimization, an offspring is ready to be compared against the parent. DEC uses a simple scheme for replacing a member of the population. The algorithm is outlined in Fig. 3.11. If the offspring generated has a better fitness than its parent, the parent gets replaced in the population. However, even when an offspring has worse fitness, it replaces the parent in the next generation with a small probability. This scheme helps maintain the diversity of the population and also grants more exploration of the search space. The *AcceptWorse* predicate used in the *Replacement* algorithm returns true or false based on this probability and the current generation number. In the exploration phase, we can allow a worse solution to replace the parent more frequently. The probability of accepting a worse offspring is set to 30% in the beginning generations. However, it would not be wise to keep the probability high in the exploitation phase as well, since we expect the algorithm to have explored the search space enough by the end of the exploration phase. Therefore, we decrease the probability of accepting the worse offspring down to 10%.

Algorithm: Replacement$(I_p, I_c, g, \mathcal{F}, \mathcal{D})$

Input: Parent individual I_p, offspring I_c, generation g, validity index $\mathcal{F}$, and distance measure $\mathcal{D}$

Output: Better individual that gets added to new population

```
1 if Fitness(I_p, F, D) is better than Fitness(I_c, F, D) then
2     if AcceptWorse(g) then
3         return I_c
4     else
5         return I_p
6 else
7     return I_c
```

Figure 3.11: Replacement scheme

3.5 Population Perturbation

So far we have discussed what takes place within a cycle in the DEC algorithm. In this section, we discuss what happens in between cycles. A good clustering for a data set can be found fairly quickly in some cases and may take a very long time in other cases. Depending on the data set, if the search space is small to begin with, we could find a good solution quickly. However, in general, it would take many generations and cycles for the algorithm to be able to find a good solution. Usually, these type of search algorithms have a tendency to get stuck in a basin of attraction, an area of local optima in the search space from which it is hard to escape. For that reason, we introduce perturbation in the population in between cycles. The perturbation scheme modifies a percentage of individuals in the population making the population more diverse. In turn this gives the algorithm a chance to move away from its current local optima and explore the search space further. The algorithm is given in Fig. 3.12. The percentage of the number of individuals to be modified is decided based on the current cycle. In the earlier cycles, when the algorithm is still exploring the search space, we allow more individuals to be modified. However, as the algorithm progresses, the individuals in the population are more reliable and therefore, the percentage of individuals to be perturbed is reduced. To modify an individual, the centroids and their corresponding activation thresholds are changed

Algorithm: Perturb$(P, cycle)$

Input: Population $P, cycle$
Result: Modified population P

```
1  if cycle ≥ 0 and cycle < 0.4× numCycle then
2      Modify(P, 0.5, cycle)   // Modify 50% of population P
3  else if cycle ≥ 0.4× numCycle and cycle < 0.8× numCycle then
4      Modify(P, 0.3, cycle)   // Modify 30% of population P
5  else if cycle ≥ 0.8× numCycle and cycle < × numCycle −1 then
6      Modify(P, 0.1, cycle)   // Modify 10% of population P
```

Algorithm: Modify$(P, \gamma, cycle)$

Input: Population P, γ% of individuals to be modified
Result: A modified population P with perturbed individuals such that the resulting clustering is changed.

```
7  Randomly select a subset A ⊆ P − {I_best} such that |A| = γ × |P|
8  foreach individual I = (Ω, T, C) ∈ A do
9      foreach t ∈ T do
           // modify threshold
10         t ← δ_t(t, cycle)
11     foreach ω = {ω_1, ..., ω_f} ∈ Ω do
           // modify centroids
12         for i = 1 to f do
13             ω_i ← δ_ω(ω_i, cycle)
14     Compute the fitness of clustering C(I)
       // clustering C of I is also recomputed here
```

Figure 3.12: Perturbation of population after every cycle

according to function δ_t that is defined as follows:

$$\delta_t(t, cycle) = \begin{cases} 1 + (\alpha(cycle) \times 0.5)) \times t, & \mathcal{U}\text{Random}(0,1) < 0.5 \\ 1 - (\alpha(cycle) \times 0.5)) \times t, & \text{otherwise} \end{cases} \tag{3.2}$$

where α is a function of *cycle*. Similarly, δ_ω can be defined as

$$\delta_\omega(\omega, cycle) = \begin{cases} (1 + \alpha(cycle)) \times \omega, & \mathcal{U}\text{Random}(0,1) < 0.5 \\ (1 - \alpha(cycle)) \times \omega, & \text{otherwise} \end{cases} \tag{3.3}$$

These functions help determine whether a threshold or feature in a centroid has to be modified or not and the magnitude of change in the threshold or feature if it is to be modified. The reason behind introducing δ_t and δ_ω is to allow exploration and exploitation in the algorithm. The magnitude and probability of perturbing the centroids and the threshold are higher in the beginning stages of the search as compared to the later stages.

3.6 Interchanging Distance Measure

Other than perturbing the population in between cycles, we also change the distance function after a number of cycles have passed. In the DEC algorithm given in Fig. 3.1, the distance measure is changed from Euclidean distance to *MinMax* distance after we are done with 80% of the cycles. The reason for changing the distance measure is that the Euclidean distance does not give equal preference to all features in an item. Due to this, the search would be biased by the features with larger magnitude in the algorithm. Therefore, to ameliorate the effect of this bias, we change the distance function to the *MinMax* distance. *MinMax* distance gives equal priority to all features in the items by normalizing the distance.

When the distance measure is changed, the fitness of all individuals in the current population are recomputed using the new distance measure. The clustering of offspring created in the remaining cycles are computed using the *MinMax* distance. The fitness comparison is also done using the new distance measure. However, it should be noted that the best individual in the final population, at the end of the

algorithm, is returned by recomputing the validity index for all clustering solutions in the population using the Euclidean distance. This is done to provide a fair comparison of our results with other algorithms discussed in the next chapter.

3.7 Stopping Criteria

As discussed in previous sections, the population in the DEC algorithm might converge early. Thus, it is more efficient to terminate the search rather than making the algorithm run for the total number of generations. To determine the convergence of the algorithm, we observe the fitness of the best individual in two consecutive generations in the exploitation phase. If the fitness does not improve for a fixed number of consecutive generations, we can safely assume that the population has become stagnant and the algorithm has converged.

It can be seen from the algorithm given in Fig. 3.1 that the distance function is changed after 80% of the cycles have passed. Thus, we do not check for stagnancy in the population till a number of generations have passed after changing the distance function. This allows the algorithm a chance to adapt and utilize the new distance function introduced. If the fitness of the best individual improves before the $noImprovThreshold$ is met within a cycle, we restart the $noImprovementCounter$. This is done because the population is perturbed between cycles and we want to give the algorithm time to explore the perturbed solutions before we check for stagnancy again.

3.8 DEC Parameters

Table 3.1 presents a summary of the parameters used in the DEC algorithm for this thesis. The population size, N, depends on the number of features and λ, which is set to 10 or 20 depending on how many individuals we want in the population. Generally, a very small or very large population does not help with the search process. Therefore, we require at least 80 individuals in the population and allow at most 200 individuals for a data set. The maximum number of generations per cycle used for all data sets in DEC is 250 and the maximum number of cycles allowed is 10.

We allow a maximum of 20 clusters in an individual (denoted by K_{max}), and

require at least 2 clusters in an individual (denoted by K_{min}). It should be noted that K_{min} can be chosen to be any value greater than 2. Additionally, we require each cluster in a clustering to have at least a minimum number of items for it to be considered valid. Logically, a cluster should have at least one item in it but in DEC, any cluster that has less than 10 items is considered tiny and either marked up for redistribution or size increment. As for the cluster centers, we want them to be an accurate representation of the cluster they represent. Therefore, DEC re-computes the centroids of a cluster on various occasions. For instance, when a cluster is built from scratch, its centroid is recomputed every time ξ items are added to it. We set ξ to be 5 for the entirety of the DEC algorithm. In other cases, a centroid is either recomputed from scratch for all items in its cluster or after the addition of a single item, depending on how accurate we want the representation to be.

Table 3.1: Summary of the DEC parameters

Parameter	Value	Comments
N	[80, 200]	size of the population
$numGenerations$	250	total number of generations
$numCycles$	10	total number of cycles
λ	10,20	factor used to determine N
K_{max}	20	maximum number of clusters allowed
K_{min}	2	minimum number of clusters required
μ	0.5	probability of calling either *Redistribute* or *GrowCluster*
ξ	$\mathcal{F}(s)$	number of items added to cluster before its center is recomputed
η	0.8,0.2	crossover probability
σ_t	0.5	scaling factor for threshold values
σ_{base}	0.8	base scaling factor for centroids
ζ	0.5,0.001	parameter that determines change for σ_ω
β	5	threshold value to be considered in *BreakUp*
ψ	2	successful mutation counter in local optimization
α	0.5,0.3,0.2	parameter that determines perturbation for centroids and thresholds
$noImprovThreshold$	100	number of consecutive non-improvements of population before we quit

After the initial population has been generated and evaluated, the algorithm chooses random donors to create an offspring. The crossover probability, η, for either selecting a parent component or calculating a new component using the donors is dependent on the generation. Initially, η is set to 0.8. It is reduced to 0.2 after 70% of the generation have passed. Apart from the crossover probability, the scaling factor

also contributes to the crossover scheme. We have two different scaling factors for the two components of an individual I, namely, σ_ω for the centroids $\Omega(I)$ and σ_t for the threshold values $\mathcal{T}$. The scaling factor for threshold is fixed at 0.5 throughout the algorithm. However, the scaling factor for the centroid is initially set to 0.8. It is reduced by 0.1 after some generations have passed, such that the scaling factor does not go below 0.2 till the end of a cycle. In addition to that, the value of ζ in Equation. 3.1 is set to 0.5 for the first 70% generations and then reduced to 0.001 for the remaining 30% generations.

Once crossover is done and an offspring is generated, an initial clustering for the offspring is obtained. During the initial clustering, the centroids get recomputed based on how many items get added to a cluster. This number ξ is determined by $\mathcal{F}(s)$, where:

$$\mathcal{F}(s) = \begin{cases} 5 & s \leq 500 \\ 5\left(\left\lceil \frac{s-500}{200} \right\rceil + 1\right) & 500 < s \leq 1500 \\ 5\left(\left\lceil \frac{s-1500}{500} \right\rceil + 1\right) & s > 1500 \end{cases}$$

and s represents the size of the data set .

Next, the offspring undergoes local optimization based on its clustering. Only individuals that contain a large cluster go through the *BreakUp* scheme. We consider a cluster to be large if its size is more than 40% of the total number of items n in the data set X. If a large cluster exists, we break it up based on a threshold value depending on β. The value of β is kept constant throughout the algorithm. Other local optimization techniques that an individual may go through are *Merge* and *Scatter*. The maximum number of pairs of clusters that can be combined (ψ) is set to 2. Similarly, the maximum number of small clusters that can be redistributed in the local optimization of an individual is set to 2. After local optimization, the offspring is ready to be compared against its parent. In DEC, a good offspring always replaces its parent. However, the algorithm also allows a worse offspring to replace its parent with a small probability. The probability of accepting a worse solution is set to 30% for the first 70% of the generations and 10% for the remaining generations. At the end of a cycle, the entire population goes through a perturbation scheme. The parameter α in Equations. 3.2 and 3.3 changes according to the cycle number. It is set to 0.5 for first one third of the cycles. It is reduced to 0.3 for the next one

third and finally set to 0.2 for the remaining one third of the cycles. The role of this parameter is to get the algorithm out of possible local optima if required. The parameter α is varied such that it allows more exploration in terms of perturbation for the earlier cycles as compared to the later cycles. It should also be noted that towards the end of the cycles, the stagnancy of the population is also checked. The parameter $noImprovThreshold$ is set based on the total number of generations in the exploitation phase. Since we have a fixed number of generations for every cycle, we set this value to be 100.

3.9 DEC Running Time

In this section we discuss the worst case running time complexity of the DEC algorithm. We first list the assumptions made in our analysis of the running time. We consider a number of parameters in the algorithm to be of fixed size, i.e., a constant value independent of the input size. The list below enumerates the fixed parameters in the DEC algorithm:

- The population size N
- The number of features, f, for an item
- The maximum number of centroids in an individual K
- The thresholds for the centroids
- The total number of generations, $numGenerations$, per cycle
- The total number of cycles, $numCycles$, in the algorithm
- The number of items being redistributed from a tiny cluster
- The number of items added to a cluster at any point

It should be noted that the recomputation of a centroid can utilize the previous centroid value, thereby taking only $O(1)$ time. However, in some cases, the centroid has to be recomputed from scratch, thereby taking $O(n)$ time. The following table gives the worst case running time for each function implemented in the DEC algorithm.

Table 3.2: Summary of run time complexity for algorithms used in DEC

Function	**Worst Case Running Time**
ComputeClustering	$O(n)$
Redistribute	$O(1)$
GrowCluster	$O(n)$
CleanIndividual	$O(n)$
$\mathcal{F}_{DB}$	$O(n)$
$\mathcal{F}_{CS}$	$O(n^2)$
$\mathcal{F}_{PBM}$	$O(n)$
Fitness	$O(n)$ or $O(n^2)$
GenerateIndividual	$O(n)$ or $O(n^2)$
Setup	$O(n)$ or $O(n^2)$
Crossover	$O(n)$ or $O(n^2)$
BreakUp	$O(n)$
Merge	$O(n)$ or $O(n^2)$
Scatter	$O(n)$ or $O(n^2)$
LocalOpt	$O(n)$ or $O(n^2)$
Replacement	$O(1)$
Modify	$O(n)$ or $O(n^2)$
Perturb	$O(n)$ or $O(n^2)$

All the algorithms that have running times given as $O(n)$ or $O(n^2)$ indicate the complexity based on the validity index used in the algorithm. If DEC uses the DB index or PBM index, the complexity is $O(n)$. However, if DEC uses the CS index, the run time complexity is $O(n^2)$. Thus, we can conclude that the running time of our algorithm is $O(n)$ or $O(n^2)$ depending on the validity index used.

Chapter 4

Experimental Results

In this chapter, we study the effect of using different validity indexes and distance measures in the Differential Evolution Clustering (DEC) algorithm for a number of real-life data sets. Furthermore, we also compare performance of the DEC algorithm with the ACDE algorithm implemented by Das *et al.* in [8] and other clustering algorithms referred in their paper.

4.1 Experimental Setup

The DEC algorithm was tested with various type of data sets to observe the algorithm's behavior with respect to distinct characteristics of different data sets. The data sets used in this thesis were selected based on:

- number of items,
- number of classifications,
- number of features

in the data set [5],[33]. Table 4.1 provides a summary of the data sets used in this thesis. We have tested our algorithm with a total of 17 data sets. These data sets have size ranging from 150 to 3500 items, number of features ranging from 3 to 90 and number of classification ranging from 2 to 15. We refer to the classification given in these data sets as the ground truth. It should be noted that even though the ground truth classification is given, the DEC algorithm does not use this information.

Table 4.1: Summary of data sets

Data sets	**Size**	**# of features**	**# of clusters**
Cancer	683	9	2
Cleveland	297	13	5
Dermatology	358	34	6
E.coli	336	7	8
Glass	214	9	6
Haberman	306	3	2
Heart	270	13	2
Ionosphere	351	33	2
Iris	150	4	3
M.Libras	360	90	15
Pendigits	3497	16	10
Pima	768	8	2
Sonar	208	60	2
Vehicle	846	18	4
Vowel	990	13	11
Wine	178	13	3
Yeast	1484	8	10

We calculate the validity index for the ground truth in two different ways. First, we compute the validity index using the ground truth classification. These values are under the columns labeled GT in the following tables. In the second method, we first compute the centroids of the clustering in the ground truth classification. We then apply one iteration of the K-means algorithm to obtain a new clustering, from which the validity index is computed. These validity indexes are under the columns labeled GT_K. Table 4.2 summarizes the different validity indexes calculated for all data sets.

The DEC algorithm is implemented in C++ and compiled using g++ and the -O3 optimization flag. The algorithm is run 30 times for each data set independently to account for its stochastic nature. All of these runs were conducted on machines running Intel Core i5-3570 CPU at 3.40 GHz and 32 GB of RAM with Ubuntu 14.04

Table 4.2: Validity index values for the ground truth

Data sets	DB		CS		PBM	
	GT	GT_K	GT	GT_K	GT	GT_K
Cancer	0.79	0.78	1.19	1.13	11.35	11.65
Cleveland	8.46	6.78	15.28	12.72	7.12	3.51
Dermatology	5.89	3.85	4.72	2.43	6.03	7.28
E.coli	2.32	2.39	2.37	2.27	15.23	13.29
Glass	3.73	3.76	4.29	3.96	0.78	0.91
Haberman	4.72	4.81	9.59	7.66	2.51	2.64
Heart	4.15	3.91	10.41	8.51	12.44	13.30
Ionosphere	4.08	3.81	4.88	4.74	0.72	0.74
Iris	0.75	0.73	0.97	0.88	4.59	4.72
M.Libras	3.85	3.56	3.93	3.13	0.22	0.25
Pendigits	2.14	2.05	2.47	2.15	31.84	35.10
Pima	4.42	4.18	16.00	8.89	23.19	25.62
Sonar	5.68	5.58	6.25	5.71	0.22	0.23
Vehicle	12.62	8.11	7.72	3.73	64.65	94.76
Vowel	11.07	11.07	13.71	13.43	0.39	0.40
Wine	1.51	1.14	2.17	1.44	384.63	501.27
Yeast	2.92	2.49	3.41	2.66	0.09	0.11

operating system. The Mersenne Twister random number generator mt19937 [28] was used in the algorithm implementation for any random number generation.

4.2 Results and Discussion

In this section, we discuss the performance of the DEC algorithm on the data sets given in Table 4.1. In particular, we consider: (1) quality of the solution based on different validity indexes, (2) impact of distance measures and local optimization in finding solutions, and (3) time taken to find a good solution. In what follows, we discuss these points in detail.

4.2.1 Effect of validity indexes

We collect results by running DEC 30 times for each triple $(X, \mathcal{F}, K_{min})$, where X is a data set in Table 4.1, $\mathcal{F} \in \{\text{DB}, \text{CS}, \text{PB}\}$ is a validity index, $K_{min} \in \{2, K_{orig}\}$ is the minimum number of clusters, and K_{orig} is the number of classes in the ground truth classification of X. Tables 4.3 and 4.4 summarize the mean validity index values obtained. In Table 4.3 the minimum number of clusters (K_{min}) is set to 2, whereas in Table 4.4, K_{min} is set to K_{orig}.

It can be seen in Table 4.3, that when K_{min} is as low as 2, DEC converges to the minimum number of clusters for all data sets and is unable to return the ground truth classification. This is the case since all three validity indexes get the lowest or highest value when the number of clusters is small. Therefore, even when DEC finds a clustering solution similar to the ground truth, it would replace it with a clustering that has fewer clusters but a better fitness. Similarly, when K_{min} is set to the ground truth classification, labeled K_{orig} in Table 4.4, it can be seen that DEC again converges to the minimum number of clusters specified (with the exception of the *M.Libras* data set).

In order to further see the effect of K_{min} on these validity indexes, the algorithm was run 30 times for a range of values of K_{min} for a subset of these data sets. The results obtained are plotted in Fig. 4.1–4.3.

Table 4.3: Comparison of clustering validity indexes with $K_{min} = 2$

Data Set		**Mean # Clusters returned by DEC**			**Mean Validity Index**								
					DB			**CS**			**PBM**		
Name	**# Classes**	DB	CS	PBM	GT	GT_K	DEC	GT	GT_K	DEC	GT	GT_K	DEC
Cancer	2	2.00	2.82	2.00	0.79	0.78	0.58	1.19	1.13	1.13	11.35	11.65	16.14
Cleveland	5	2.00	2.00	2.83	8.46	6.78	1.12	15.28	12.72	1.29	7.12	3.51	42.76
Dermatology	6	2.00	2.00	3.00	5.89	3.85	1.00	4.72	2.43	1.12	6.03	7.28	14.61
E.coli	8	2.00	2.00	2.00	2.32	2.39	0.91	2.37	2.27	0.95	15.23	13.29	66.16
Glass	6	2.00	2.00	2.00	3.73	3.76	0.84	4.29	3.96	0.10	0.78	0.91	11.18
Haberman	2	2.03	2.00	2.00	4.72	4.81	0.88	9.59	7.66	0.53	2.51	2.64	25.42
Heart	2	2.00	2.00	2.40	4.15	3.91	1.12	10.41	8.51	1.31	12.44	13.30	40.94
Ionosphere	2	2.00	2.00	2.60	4.08	3.81	1.12	4.88	4.74	1.58	0.72	0.74	2.04
Iris	3	2.00	3.00	3.00	0.75	0.73	0.42	0.97	0.88	0.60	4.59	4.72	4.41
M.Libras	15	2.87	2.33	2.00	3.85	3.56	1.43	3.93	3.13	1.51	0.22	0.25	1.40
Pendigits	10	2.00	2.00	2.14	2.14	2.05	0.81	2.47	2.15	0.83	31.84	35.10	90.86
Pima	2	2.00	2.00	2.00	4.42	4.18	0.71	16.00	8.89	0.58	23.19	25.62	156.22
Sonar	2	2.00	2.00	2.00	5.68	5.58	1.12	6.25	5.71	1.11	0.22	0.23	1.18
Vehicle	4	2.00	2.00	2.07	12.62	8.11	0.58	7.72	3.73	0.31	64.65	94.76	422.01
Vowel	11	2.00	2.00	3.73	11.07	11.07	0.82	13.71	13.43	0.80	0.39	0.40	3.65
Wine	3	2.00	2.00	2.00	1.51	1.14	0.96	2.17	1.44	0.70	**384.63**	**501.27**	**158.88**
Yeast	10	2.00	2.00	2.05	2.92	2.49	0.71	3.41	2.66	0.32	0.09	0.11	2.05

Table 4.4: Comparison of clustering validity indexes with $K_{min} = K_{orig}$

Data Set		**Mean # Clusters returned by DEC**			**Mean Validity Index**								
					DB			**CS**			**PBM**		
Name	**# Classes**	DB	CS	PBM	GT	GT_K	DEC	GT	GT_K	DEC	GT	GT_K	DEC
Cancer	2	2.00	2.82	2.00	0.79	0.78	0.58	1.19	1.13	1.13	11.35	11.65	16.14
Cleveland	5	5.00	5.00	5.00	8.46	6.78	1.60	15.28	12.72	1.59	7.12	3.51	29.86
Dermatology	6	6.00	6.00	6.00	5.89	3.85	1.57	4.72	2.43	1.48	6.03	7.28	8.70
E.coli	8	8.23	8.40	8.00	2.32	2.39	1.29	2.37	2.27	1.68	15.23	13.29	27.41
Glass	6	6.00	6.00	6.00	3.73	3.76	1.02	4.29	3.96	0.30	0.78	0.91	5.05
Haberman	2	2.03	2.00	2.00	4.72	4.81	0.88	9.59	7.66	0.53	2.51	2.64	25.42
Heart	2	2.00	2.00	2.40	4.15	3.91	1.12	10.41	8.51	1.31	12.44	13.30	40.94
Ionosphere	2	2.00	2.00	2.60	4.08	3.81	1.12	4.88	4.74	1.58	0.72	0.74	2.04
Iris	3	3.00	3.00	3.00	0.75	0.73	0.56	0.97	0.88	0.60	4.59	4.72	4.42
M.Libras	15	2.87	2.33	2.00	3.85	3.56	1.43	3.93	3.13	1.51	0.22	0.25	2.14
Pendigits	10	10.47	11.00	10.03	2.14	2.05	1.07	2.47	2.15	1.33	31.84	35.10	53.06
Pima	2	2.00	2.00	2.00	4.42	4.18	0.71	16.00	8.89	0.58	23.19	25.62	156.22
Sonar	2	2.00	2.00	2.00	5.68	5.58	1.12	6.25	5.71	1.11	0.22	0.23	1.18
Vehicle	4	4.00	4.07	4.00	12.62	8.11	0.87	7.72	3.73	0.62	64.65	94.76	422.20
Vowel	11	11.33	11.17	11.03	11.07	11.07	1.25	13.71	13.43	1.34	0.39	0.40	2.17
Wine	3	3.00	3.00	3.03	1.51	1.14	1.12	2.17	1.44	0.94	**384.63**	**501.27**	**256.23**
Yeast	10	10.00	10.00	10.39	2.92	2.49	1.26	3.41	2.66	1.38	0.09	0.11	0.24

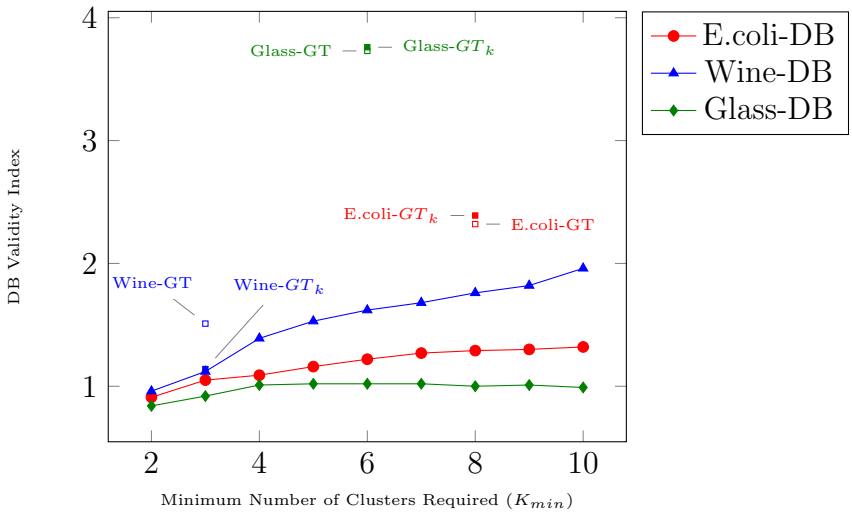

Figure 4.1: Effect of K_{min} on DEC using DB index

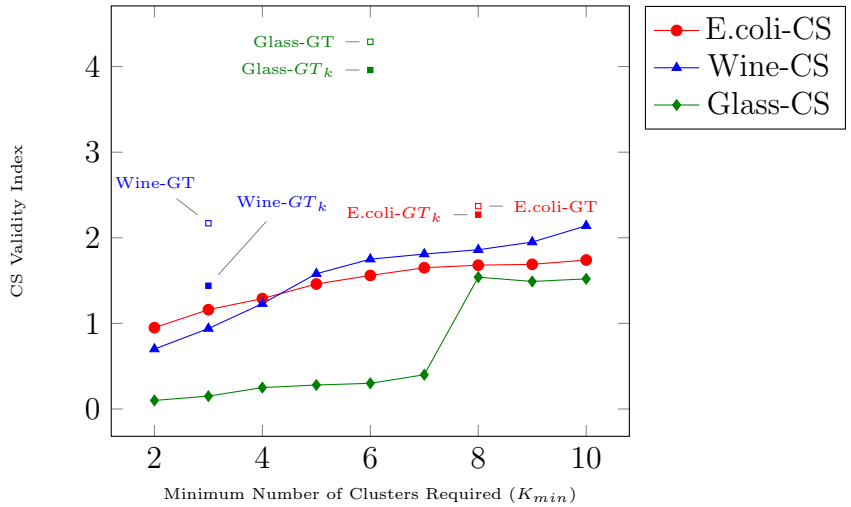

Figure 4.2: Effect of K_{min} on DEC using CS index

It can be clearly seen from Fig. 4.1 and 4.2 that both the DB and CS index values generally increase as the specified K_{min} increases. Similarly, in Fig. 4.3, the value of the PBM index decreases (i.e., fitness degrades) as the specified K_{min} increases (with the exception of the *Wine* data set). Due to this behavior, it is not possible for our optimization algorithm to be able to find a clustering closer to the ground truth with a K_{min} specified lower than the known number of clusters. However, it should be noted that the DEC algorithm is able to find a clustering with a better

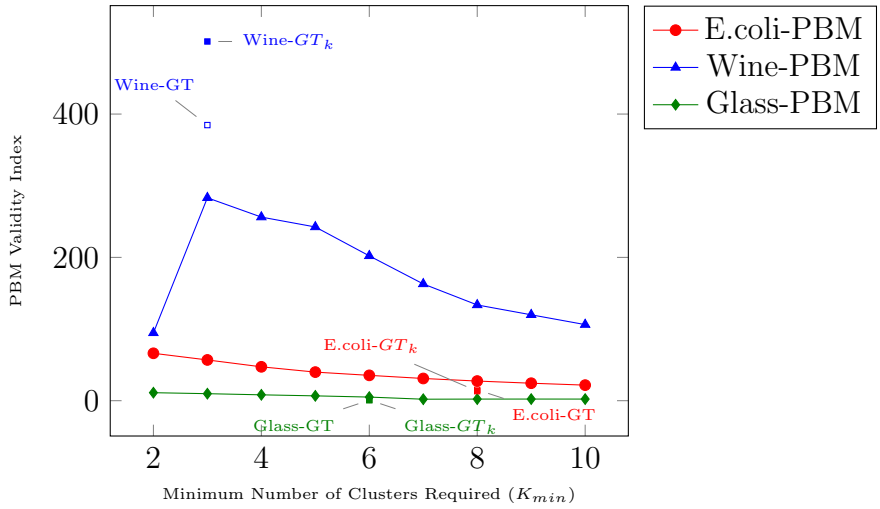

Figure 4.3: Effect of K_{min} on DEC using PBM index

fitness than the ground truth classification consistently for all data sets. The only exception is the *Wine* data for the PBM index (highlighted) for both Tables 4.3 and 4.4.

Table 4.5 compares the mean CPU time in seconds for different validity indexes on one run of the DEC algorithm. As discussed in Table 3.2, the DEC algorithm takes longer to finish for all data sets when CS index is used instead of DB or PBM index. Additionally, the number of features in a data set also impact the running time as the distance calculation takes longer. Moreover, when K_{min} is set to 2, the algorithm has a larger search space to explore and therefore, it takes longer for DEC to find a good clustering for almost all data sets.

Table 4.5: Mean CPU time in seconds used by DEC for different validity indexes

Data sets				DB		CS		PBM	
Name	**Size**	**# features**	**# clusters**	K_{min}	K_{orig}	K_{min}	K_{orig}	K_{min}	K_{orig}
Cancer	683	9	2	192.63	192.63	1209.07	1209.07	227.00	227.00
Cleveland	297	13	5	187.53	63.86	400.83	77.43	132.37	56.33
Dermatology	358	34	6	653.10	227.14	828.00	244.00	252.23	184.73
E.coli	336	7	8	166.10	30.86	432.43	34.56	129.20	29.10
Glass	214	9	6	86.00	3788.11	147.63	3407.07	79.23	66.66
Haberman	306	3	2	46.80	46.80	178.30	178.30	36.56	36.56
Heart	270	13	2	164.80	164.80	335.40	335.40	121.13	121.13
Ionosphere	351	33	2	271.03	271.03	483.93	483.93	201.96	201.96
Iris	150	4	3	30.30	19.07	62.03	36.06	21.46	16.13
M.Libras	360	90	15	488.97	488.97	688.06	688.06	518.73	518.73
Pendigits	3497	16	10	1693.17	216.23	51237.00	1003.00	1305.57	191.73
Pima	768	8	2	345.10	345.10	1997.87	1997.87	269.70	269.70
Sonar	208	60	2	325.70	325.70	434.86	434.86	204.63	204.63
Vehicle	846	18	4	332.90	282.10	1692.87	799.16	362.46	270.86
Vowel	990	13	11	382.23	74.00	2282.10	91.20	181.06	73.16
Wine	178	13	3	71.36	42.83	132.50	79.60	32.86	27.23
Yeast	1484	8	10	909.40	19760.70	15092.50	21209.80	569.30	447.00

To analyze how different validity indexes partition a data set, we have collected clustering solutions for the entire population at the end of DEC for all data sets. In what follows, we show the clustering solutions obtained for the best individual in the population for the *Wine* data set. In order to check the algorithm's ability to find a clustering similar to the ground truth, we collected the results with minimum number of clusters set to K_{orig}, i.e., 3.

The best clustering obtained by DEC using the DB index in one of the runs of DEC looks as follows.

Clustering using DB index	**Ground Truth**		
	Class 1 59 items	**Class 2** 71 items	**Class 3** 48 items
Cluster 1 55 items	51	4	0
Cluster 2 72 items	8	64	0
Cluster 3 51 items	0	3	48

The ground truth classification is labeled as Class 1, 2 and 3, and the DEC classification is labeled as Cluster 1, 2 and 3. It can be seen that DEC is able to get a clustering very similar to the ground truth using the DB index. The misclassification shows that a few items, as classified by the ground truth, are in reality closer to the centroid of another cluster. Since with the DB index data is partitioned based on the compactness of a cluster, this information tells us that these misclassified items might be outliers in their respective groups. On the other hand, the best clustering obtained by DEC using the CS index on one run of DEC looks as follows.

Clustering using CS index	**Ground Truth**		
	Class 1 59 items	**Class 2** 71 items	**Class 3** 48 items
Cluster 1 36 items	36	0	0
Cluster 2 86 items	23	63	0
Cluster 3 56 items	0	8	48

On the other hand, the CS index is not able to cluster all items in Class 1 together. Since the CS index forms clusters based on how similar the items are to each other, this shows that 23 items in Class 1 are more similar to items in Class 2 than the rest of the items in Class 1. Lastly, the best clustering obtained by DEC using the PBM index on one run of DEC looks as follows.

Clustering using PBM index	**Ground Truth**		
	Class 1 59 items	**Class 2** 71 items	**Class 3** 48 items
Cluster 1 55 items	49	6	0
Cluster 2 18 items	4	13	1
Cluster 3 105 items	6	52	47

The above classification shows that with the PBM index, DEC is not able to partition items in Class 2 and 3. Based on how the PBM index is defined, it is expected that a higher preference is given to the separation between clusters. After observing the centroids in the ground truth classification for Class 2 and 3, it was noted that these two classes are actually close together, whereas Class 1 is further apart from either

of them. Thus, the best individual returned by DEC using this index was one where items from Class 1 were separated and very far away from items in Class 2 and 3.

To summarize, in a clustering algorithm, a data set can be partitioned in several ways depending on the validity index used. As discussed in earlier chapters, the choice of a validity index significantly impacts the performance of an algorithm. The results collected attest to the fact that a validity index affects the quality of the solution, the time taken to find a good solution and the interpretation of the result. Basically, a clustering algorithm can be used to study various characteristics of a data set by exploiting the properties of different validity indexes.

4.2.2 Effect of distance measures

In the DEC algorithm, we use a combination of the Euclidean distance and the *MinMax* distance to search for a clustering solution. As mentioned previously, the Euclidean distance measure is influenced by the magnitude of the features in a data set. Thus, using this measure alone can influence the search drastically since larger features will dominate the distance affecting the direction of the search. To overcome this problem, we introduced another distance measure called the *MinMax* distance measure. This measure normalizes the feature values such that all features contribute equally to the distance. However, using this measure alone would not necessarily improve the search either. Depending on the data set, it is quite possible that the ground truth clustering is actually obtained using a subset of features rather than considering all features together. Therefore, we use a combination of the Euclidean distance and the *MinMax* distance in order to search for a good clustering. Tables 4.6–4.8 summarize the results collected over 30 independent runs for a subset of data sets using *MinMax* distance alone, labeled as DEC_{MinMax}, Euclidean distance alone, labeled as $\text{DEC}_{Euclidean}$, and our original algorithm which uses both as indicated in Fig. 3.1.

It can be observed that when DEC runs using Euclidean distance alone, the quality of the solution seems to be better. However, this does not necessarily indicate a good clustering solution. Based on different validity indexes, the distance measure can significantly affect the clustering for a data set. For instance, the *Wine* data set has features that vary considerably in magnitude. The clustering of the best solution

Table 4.6: Comparison of DB index based on different distance measures used in DEC

Data Set	$\mathbf{DEC}_{MinMax}$				$\mathbf{DEC}_{Euclidean}$				**DEC**			
	GT	GT_K	K_{min}	K_{orig}	GT	GT_K	K_{min}	K_{orig}	GT	GT_K	K_{min}	K_{orig}
E.coli	2.66	2.87	1.27	1.51	2.32	2.39	0.97	1.20	2.32	2.39	0.91	1.29
Glass	3.73	3.85	0.76	1.54	3.73	3.76	0.61	1.05	3.73	3.76	0.84	1.02
Wine	1.51	1.55	1.66	1.65	1.51	1.14	0.38	0.47	1.51	1.14	0.96	1.12

Table 4.7: Comparison of CS index based on different distance measures used in DEC

Data Set	$\mathbf{DEC}_{MinMax}$				$\mathbf{DEC}_{Euclidean}$				**DEC**			
	GT	GT_K	K_{min}	K_{orig}	GT	GT_K	K_{min}	K_{orig}	GT	GT_K	K_{min}	K_{orig}
E.coli	2.63	2.22	1.31	2.00	2.37	2.27	1.26	1.49	2.37	2.27	0.95	1.68
Glass	4.29	4.25	0.29	0.67	4.29	3.96	0.23	0.53	4.29	3.96	0.10	0.30
Wine	2.17	2.35	2.36	2.52	2.17	1.44	0.59	0.68	2.17	1.44	0.70	0.94

Table 4.8: Comparison of PBM index based on different distance measures used in DEC

Data Set	$\mathbf{DEC}_{MinMax}$				$\mathbf{DEC}_{Euclidean}$				**DEC**			
	GT	GT_K	K_{min}	K_{orig}	GT	GT_K	K_{min}	K_{orig}	GT	GT_K	K_{min}	K_{orig}
E.coli	12.84	13.87	43.46	31.49	15.23	13.29	37.17	23.36	15.23	13.29	66.16	27.41
Glass	0.78	0.88	4.78	1.68	0.78	0.91	8.80	4.28	0.78	0.91	11.18	5.05
Wine	384.64	366.61	249.37	311.63	384.63	501.27	910.24	897.07	384.63	501.27	158.88	256.23

given by $\text{DEC}_{Euclidean}$ for the *Wine* data set after one run of DEC is as follows.

Clustering using Euclidean	**Ground Truth**		
	Class 1 59 items	**Class 2** 71 items	**Class 3** 48 items
Cluster 1 18 items	17	1	0
Cluster 2 132 items	14	70	48
Cluster 3 28 items	28	0	0

The Euclidean distance causes the search to be guided based on the largest feature and thus, does not separate Classes 2 and 3 at all. On further analysis, it was observed that the items in Classes 2 and 3 differ mostly in features that have smaller magnitudes. Therefore, they appear to be very close to each other based on the Euclidean distance, which is dominated by the larger features. Thus, the algorithm is only able to separate items in Class 1 from Class 2 and 3. On the contrary, the clustering obtained by DEC_{MinMax} is as follows.

Clustering using MinMax	**Ground Truth**		
	Class 1 59 items	**Class 2** 71 items	**Class 3** 48 items
Cluster 1 47 items	47	0	0
Cluster 2 81 items	12	69	0
Cluster 3 50 items	0	2	48

It is interesting to see that DEC using *MinMax* distance is able to give a clustering similar to the ground truth for the *Wine* data set. However, this may not be the case for all data sets. If all features in a data set do not vary widely in magnitude, then

Euclidean and *MinMax* distance measures would not show a significant difference in the quality of the solutions. To summarize, different distance measures can be used in a clustering algorithm to study the relationship between a set of features if the ground truth classification is known. Moreover, even for data sets for which the classification is unknown, these distance measures can be used to study the characteristics of such data sets.

4.2.3 Effect of local optimization

In the classical DE algorithm implemented in [40], there is no local optimization after a clustering solution is obtained from crossover. To test the effect of local optimization methods implemented in our DEC algorithm, we collected results for two different algorithms: DEC_C with only crossover and no distance change or local optimization, and DEC_{C+D} with crossover and distance change but no local optimization as shown in Table 4.9 below.

Table 4.9: Tested algorithms

Algorithm	**Crossover**	**Distance Change**	**Local Optimization**
DEC_C	✓	✗	✗
DEC_{C+D}	✓	✓	✗
DEC	✓	✓	✓

Table 4.10 compares the DEC algorithm with the DEC_C and DEC_{C+D} on a number of data sets. We can observe that the fitness of the clustering solutions obtained by using DB index or CS index is better in case of DEC rather than DEC_{C+D} for almost all data sets except E.coli and Cancer. However, the values of the DB and CS index for the DEC_C algorithm are smaller than both DEC and DEC_{C+D} for multiple data sets. To understand why that is the case and study the results better, we show the clustering for the Wine data set for all three algorithms in Table 4.11.

We can observe from Table 4.11 that without distance change or local optimization, it is difficult for the algorithm to partition the items present in different classes. Moreover, even though the partition is closer to the ground truth after allowing

Table 4.10: DEC with no local optimization

Data Set	Algorithm	CS		DB	
		# of clusters	Index	# of clusters	Index
Cancer	DEC_C	2.00	1.07	2.00	0.74
	DEC_{C+D}	2.00	0.91	2.00	0.64
	DEC	2.00	1.13	2.82	0.58
Glass	DEC_C	2.00	0.89	2.00	0.61
	DEC_{C+D}	2.00	1.13	2.80	1.82
	DEC	6.00	0.30	6.00	1.02
Iris	DEC_C	3.00	0.73	3.00	0.61
	DEC_{C+D}	3.00	0.78	3.00	0.58
	DEC	3.00	0.60	3.00	0.56
Wine	DEC_C	3.00	0.82	3.00	0.48
	DEC_{C+D}	3.00	1.43	3.00	1.16
	DEC	3.00	0.94	3.00	1.12
E.coli	DEC_C	9.0	1.46	8.0	1.18
	DEC_{C+D}	8.0	1.47	9.2	1.24
	DEC	8.40	1.68	8.23	1.29

change in the distance measure, the DB index as well as the CS index show that DEC with local optimization has better fitness.

According to Table 4.10, the value of the validity indexes was lower for DEC_C algorithm in a few cases. However, after analyzing the individual cluster partitioning for other data sets such as Cleveland, Dermatology, Vehicle etc., a similar trend was observed, where items belonging to different classes were not partitioned by crossover alone. The reason that DEC_C algorithm could not partition items from different classes is due to the fact that the modification of the centroids done in the genotype might not impact the clustering in the phenotype at all. Furthermore, the DEC_{C+D} algorithm with crossover and distance change impacts the clustering only when the feature values vary over wide ranges. On the other hand, the local optimization introduced in our DEC algorithm considers both the genotype as well

Table 4.11: Clustering for Wine data set

Algorithm	Clusters	Ground Truth		
		Class 1 59 items	**Class 2** 71 items	**Class 3** 48 items
DEC$_C$	**Cluster 1** 44 items	41	3	0
	Cluster 2 124 items	8	68	48
	Cluster 3 10 items	10	0	0
DEC$_{C+D}$	**Cluster 1** 66 items	59	7	0
	Cluster 2 61 items	0	61	0
	Cluster 3 51 items	0	3	48
DEC	**Cluster 1** 55 items	51	4	0
	Cluster 2 72 items	8	64	0
	Cluster 3 51 items	0	3	48

as the phenotype when generating new individuals. This helps the algorithm find clustering solutions closer to the ground truth, which is not the case with DEC$_C$ or DEC$_{C+D}$ algorithms.

Similar behavior of these three algorithms was observed in other data sets as well. Thus, it seems that local optimization plays an important role in helping to find clustering solutions that are closer to the ground truth.

4.2.4 Effect of clustering computation

A clustering algorithm can be tailored to return a clustering solution either in the form of a set of clusters, where every cluster is a collection of items in the data set, or as a set of centroids with a scheme to add items to the centroids using some

other algorithm. In this section, we discuss different ways to obtain a clustering and analyze their effect on the quality of the solution.

4.2.4.1 Effect of clustering computation on ground truth

As mentioned in the previous sections, we have used two different clustering methods to calculate the fitness of the ground truth classification. In the first method, referred to as GT, the fitness is calculated using the known classification. In the second method, referred to as GT_K, we first compute the centroids for the known clusters and then use one iteration of the K-means algorithm to redistribute the items based on the computed centroids. This method could be considered a more accurate representation of the clustering since it is based on the actual distance of items from their respective centroids in the ground truth classification.

For some data sets, the clustering obtained by GT_K may differ significantly from the ground truth, GT, due to the distance measure not being able to capture the similarity between the items as defined by the ground truth. The clustering obtained for the *Wine* data set using GT_K is as follows.

Clustering using GT_K	**Ground Truth**		
	Class 1 59 items	**Class 2** 71 items	**Class 3** 48 items
Cluster 1 54 items	50	3	1
Cluster 2 66 items	0	49	17
Cluster 3 58 items	9	19	30

The fitness of the clustering obtained by GT_K is different than GT for all validity indexes, as seen in Tables 4.12–4.14.

4.2.4.2 Effect of Clustering Computation on DEC

The final solution given by DEC consists of a set of centroids, Ω, and a clustering $\mathcal{C}$. From Ω, one can obtain clusterings different from $\mathcal{C}$ by using different clustering computation methods. We use two different methods for computing clustering from Ω. The first method, called CA1 (Clustering Approach 1) has been discussed in the previous section in the computation of GT_K. The second method, called CA2 (Clustering Approach 2), is the same as CA1, except that the centroids are recomputed after every m items have been added to a cluster (for some constant m). Tables 4.12–4.14 below summarize the results obtained for a subset of the data sets considered in this thesis. The columns K_{min} and K_{orig} under CA1 and CA2 denote the performance of the algorithm using K_{min} and K_{orig} number of minimum clusters.

The NA marked in the table for *Glass* data set in the case of K_{min} set to two means that no partitioning was obtained. This shows that the centroids returned by DEC, when used to redistribute items using CA2, end up clustering all items together. This could happen when one of the centroids is far away from all items in the data set to begin with. This again highlights the nature of these validity indexes that try to increase the separation between centroids to improve the fitness, and encourage a partitioning where the clusters end up being on two extreme ends.

Table 4.12: Comparison of DB index values based on different types of result interpretation

Data Set	**GT**		**DEC**		**CA1**		**CA2**	
	GT	GT_K	K_{min}	K_{orig}	K_{min}	K_{orig}	K_{min}	K_{orig}
E.coli	2.32	2.39	0.91	1.29	1.80	1.06	1.00	1.25
Glass	3.73	3.76	0.84	1.02	1.41	1.44	NA	1.22
M.Libras	3.85	3.56	1.43	1.43	2.20	2.20	1.54	1.54
Pendigits	2.14	2.05	0.81	1.07	1.80	2.04	1.19	1.11
Wine	1.51	1.14	0.96	1.12	1.46	1.80	0.43	0.62

Table 4.13: Comparison of CS index values based on different types of result interpretation

Data Set	GT		DEC		CA1		CA2	
	GT	GT_K	K_{min}	K_{orig}	K_{min}	K_{orig}	K_{min}	K_{orig}
E.coli	2.37	2.27	0.95	1.68	3.17	1.23	0.28	1.28
Glass	4.29	3.96	0.10	0.30	2.22	0.91	NA	1.12
M.Libras	3.93	3.13	1.51	1.51	2.45	2.45	1.53	1.53
Pendigits	2.47	2.15	0.83	1.33	1.91	2.02	1.22	1.13
Wine	2.17	1.44	0.70	0.94	1.73	1.99	0.44	0.62

Table 4.14: Comparison of PBM index values based on different types of result interpretation

Data Set	GT		DEC		CA1		CA2	
	GT	GT_K	K_{min}	K_{orig}	K_{min}	K_{orig}	K_{min}	K_{orig}
E.coli	15.23	13.29	66.16	27.41	66.16	6.20	30.35	32.12
Glass	0.78	0.91	11.18	5.05	11.18	3.60	1.85	4.60
M.Libras	0.22	0.25	1.40	2.14	1.40	1.40	2.47	2.47
Pendigits	31.84	35.10	90.86	53.06	90.86	53.06	75.42	42.58
Wine	384.63	501.27	158.88	256.23	283.15	256.23	555.47	694.93

To analyze the difference in the clustering solutions obtained by these algorithms, we run DEC using the DB index on the *Wine* data set obtaining a solution $S^* = (\Omega^*, \mathcal{C}^*)$, where Ω^* is the set of centroids and $\mathcal{C}^*$ is the clustering for S^*. The clustering for S^* looks as follows.

Clustering using DEC	**Ground Truth**		
	Class 1 59 items	**Class 2** 71 items	**Class 3** 48 items
Cluster 1 55 items	51	4	0
Cluster 2 72 items	8	64	0
Cluster 3 51 items	0	3	48

CA1 changes the clustering $\mathcal{C}_1^*$ for Ω^* as follows.

Clustering using CA1	**Ground Truth**		
	Class 1 59 items	**Class 2** 71 items	**Class 3** 48 items
Cluster 1 56 items	50	4	2
Cluster 2 51 items	0	43	8
Cluster 3 71 items	9	24	38

This clustering clearly differs from the clustering $\mathcal{C}^*$ returned by DEC. As discussed in the previous chapter, the algorithm works with both the genotype and the phenotype of the encoding, i.e., DEC modifies the centroids as well as the clustering of an individual to generate a new offspring. Even though DEC recomputes the centroids for an individual at several steps, it is expected that the final clustering may not look similar to the one provided by CA1. The fitness of the clustering returned by CA1 ($\mathcal{C}_1^*$) is 0.64; whereas, the fitness for $\mathcal{C}^*$ is 1.11. Furthermore, it can also be seen that the clustering obtained by CA1 is fairly similar to the clustering obtained by GT_K. However, CA1 again has a better fitness value of 0.64, whereas the ground truth clustering has a worse fitness of 1.14.

Finally, the clustering obtained using CA2 ($\mathcal{C}_2^*$) is as follows.

Clustering using CA2	**Ground Truth**		
	Class 1 59 items	**Class 2** 71 items	**Class 3** 48 items
Cluster 1 47 items	46	1	0
Cluster 2 67 items	0	50	17
Cluster 3 64 items	13	20	31

As expected, the clustering changes again when the centroids get recomputed along the way. The fitness of clustering for CA2 ($\mathcal{C}_2^*$) is 0.53 and better than DEC and CA1's fitness. However, depending on the data set, the distance measure used, as well as the size of the search space, the K-means algorithm might not always give a better clustering. Since the K-means algorithm tends to use a greedy approach, it is possible to get a bad solution as seen in the results for the *Glass* data set.

In summary, the clustering obtained by the DEC algorithm can be interpreted using different methods. These different methods can prove helpful in analyzing the results obtained by a clustering algorithm to study the misclassification and understand the characteristics of a data set.

4.2.5 Comparison of DEC against existing algorithms

Table 4.15 compares the performance of DEC with five existing clustering algorithms for a subset of data sets. The ACDE algorithm used in [8] is an automatic partitional clustering algorithm for unlabeled data sets that uses a similar chromosome representation scheme and fitness function as DEC, but without the clustering. DCPSO algorithm used in [31] and GCUK algorithm used in [3] are two well known genetic clustering algorithms that do not need to know the number of clusters *a priori*. Moreover, to investigate the effects of changes made in the classical DE algorithm, we also compare DEC with a simple DE-based clustering method used in [40]. Finally,

we have also compared DEC with a standard hierarchical agglomerative clustering algorithm based on the linkage metric of the average link [10]. Since we have experimented with different values of K_{min} for DEC, we show the results for K_{min} set to 2 as DEC(2) and results for K_{min} set to known number of clusters as DEC(k), where k is the known number of clusters.

It can be seen that DEC(2) surpasses all other algorithms in terms of the validity index but the number of clusters obtained is two and not close to the ground truth classification for most of these data sets. However, not getting a classification close to the ground truth is not necessarily a negative result. In fact, the difference in the clustering solutions based on the validity indexes as compared to the ground truth result can reveal interesting anomalies in the data set. Since we do not have any prior knowledge about how the ground truth was partitioned, it is possible that the misclassification in the clustering obtained by DEC is due to some underlying assumptions in the ground truth. The misclassification could also result due to presence of outliers in the data set, errors caused while collecting the data, or most commonly, human errors in the ground truth classification. Therefore, the DEC algorithm can be used as a helpful tool in detecting these errors in a given data set for pre-analysis purposes.

Table 4.15: Comparison of DEC with different algorithms

Data Set	Algorithm	CS		DB	
		# of clusters	Index	# of clusters	Index
Cancer	DEC(2)	2.00	1.13	2.82	0.58
	ACDE	2.00	**0.45**	2.05	**0.52**
	DCPSO	2.25	0.48	2.50	0.57
	GCUK	2.00	0.61	2.50	0.63
	Classical DE	2.25	0.89	2.10	0.51
	Average Link	2.00	0.90	2.00	0.76
Glass	DEC(6)	6.00	0.30	6.00	1.02
	DEC(2)	2.00	**0.10**	2.00	**0.84**
	ACDE	6.05	0.33	6.05	1.01
	DCPSO	5.95	0.76	5.95	1.51
	GCUK	5.85	1.47	5.85	1.83
	Classical DE	5.60	0.78	5.60	1.66
	Average Link	6.00	1.02	6.00	1.85
Iris	DEC(3)	3.00	0.60	3.00	0.56
	DEC(2)	3.00	**0.60**	2.00	**0.42**
	ACDE	3.25	0.66	3.05	0.46
	DCPSO	2.23	0.73	2.25	0.69
	GCUK	2.35	0.72	2.30	0.73
	Classical DE	2.50	0.76	2.50	0.58
	Average Link	3.00	0.78	3.00	0.84
Wine	DEC(3)	3.00	0.94	3.00	1.12
	DEC(2)	2.00	**0.70**	2.00	**0.96**
	ACDE	3.25	0.92	3.25	3.04
	DCPSO	3.05	1.87	3.05	4.34
	GCUK	2.95	1.58	2.95	5.34
	Classical DE	3.50	1.79	3.50	3.39
	Average Link	3.00	1.89	3.00	5.72

Chapter 5

Conclusion

In this thesis we have proposed a DE-based algorithm for crisp clustering of real world data sets with high dimensionality. An important feature of the algorithm used is that it does not require the number of clusters to be known in advance. We also include techniques such as breaking up a very large cluster into smaller compact clusters, merging two clusters that are not crisp enough into one cluster, and redistributing items from a small cluster to refine the existing clustering solution.

DEC was tested with 17 different real life data sets of varying size and dimensionality. It was observed that our algorithm was able to find clustering solutions with better validity index than the ground truth classification 98% of the time. Furthermore, DEC was able to outperform five other state-of-the-art clustering algorithms for all common benchmark data sets discussed in the thesis. However, this does not mean that the clustering solutions obtained at the end are "optimal" as the notion of a "good" clustering changes with different distance functions, validity indexes, and clustering methods, as discussed in the thesis.

Additionally, we also observed that features of a data set might contribute differently in finding the natural partitioning of the items. In cases where features are non-linearly correlated to each other, DEC would not be able to capture this relationship using the simplistic distance measures described in this thesis. Thus, it would not be able to find a good clustering for such data sets. Therefore, we believe that if the most relevant features of a data set are provided, it would be more feasible for the DEC algorithm to find the desired ground truth clustering. Hence, for future work, we hope to integrate the feature-subset selection scheme with the

DEC algorithm. The combined algorithm would automatically project the data to a lower dimension search space, and find appropriate centroids more efficiently. Since the choice of validity index affects a clustering algorithm significantly, we also plan on experimenting our algorithm with different validity indexes after integrating the feature selection algorithm with the DEC algorithm.

References

[1] Ali, M. Z., N. Awad, R. Duwairi, J. Albadarneh, R. G. Reynolds, and P. N. Suganthan, "Cluster-based Differential Evolution with Heterogeneous Influence for Numerical Optimization," IEEE Transactions on Evolutionary Computation, pp. 393–400, 2015.

[2] Bandyopadhyay, S., C. A. Murthy, and S. K. Pal, "Pattern Classification with Genetic Algorithms," Pattern Recognition Letters, 16(8), pp. 801–808, 1995.

[3] Bandyopadhyay, S. and U. Maulik, "Genetic Clustering for Automatic Evolution of Clusters and Application to Image Classification," Pattern Recognition, 35(6), pp. 1197–1208, 2002.

[4] Berry, M. J. and G. Linoff, *Data Mining Techniques for Marketing, Sales, and Customer Support,* Wiley, 1997.

[5] Blake, C., E. Keough, and C. J. Merz, UCI Repository of Machine Learning Database, 1998.
Available at: `http://www.ics.uci.edu/mlearn/MLrepository.html` (last accessed : March 31, 2016)

[6] Bui, T. N. and B. R. Moon, "Genetic Algorithm and Graph Partitioning," IEEE Transactions on Computers, 45(7), pp. 841–855, 1996.

[7] Chou, C. H., M. C. Su, and E. Lai, "A New Cluster Validity Measure and Its Application to Image Compression," Pattern Analysis Applications, pp. 205–220, 2004.

[8] Das, S., A. Abraham, and A. Konar, "Automatic Clustering Using an Improved Differential Evolution Algorithm," IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, 38(1), pp. 218–237, 2008.

[9] Davies, D. L. and D. W. Bouldin, "A Cluster Separation Measure," IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 224–227, 1979.

[10] Day, W.H., and Edelsbrunner, "Efficient Algorithms for Agglomerative Hierarchical Clustering Methods," Journal of Classification, 1(1), pp. 7–24, 1984.

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

[12] Ester, M., H. P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, Portland, OR, pp. 226–231, 1996.

[13] Falkenauer, E., *Genetic Algorithms and Grouping Problems,* Chichester, U.K. Wiley, 1998.

[14] Fogel, L. J., A. J. Owens, and M. J. Walsh, *Articial Intelligence Through Simulated Evolution,* Wiley, 1966.

[15] Halkidi, M. and M. Vazirgiannis, "Clustering Validity Assessment: Finding the Optimal Partitioning of a Data Set," Proceedings of IEEE International Conference on Data Mining, pp. 187–194, 2001.

[16] Holland, J. H., *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence,* University of Michigan Press, 1975.

[17] Jain, A. K. and R. C. Dubes, *Algorithms for Clustering Data,* Prentice-Hall, Inc., 1988.

[18] Jain, A. K., M. N. Murty, and P. J. Flynn, "Data Clustering : A Review," ACM Computing Surveys, 31(3), pp. 264–323, 1999.

[19] Karimov, J. and M. Ozbayoglu, "Clustering Quality improvement of K-means Using a Hybrid Evolutionary Model," Procedia Computer Science, 61, pp. 38–45, 2015.

[20] Karypis, G., E. H. Han, V. Kumar, "Chameleon: A Hierarchical Clustering Algorithm Using Dynamic Modeling," Computer, 32(8), pp. 68–75, 1999.

[21] Kennedy, J. and R. Eberhart, "Particle Swarm Optimization," Proceedings of the IEEE International Conference on Neural Networks, pp. 1942–1948, 1995.

[22] Krishna, K. and M. N. Murty, "Genetic k-Means Algorithm," IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 29(3), pp. 433–439, 1999.

[23] Kuo, R. J., L. Lioa, and C. Tu, "Integration of ART2 Neural Network and Genetic k-Means Algorithm for Analyzing Web Browsing Paths in Electronic Commerce," Decision Support Systems, 40(2), pp. 355–374, 2005.

[24] Lee, C.Y. and E. K. Antonsson, "Self Adapting Vertices for Mask Layout Synthesis," Proceedings of the International Conference on Modeling and Simulation of Microsystems, pp. 83–86, 2000.

[25] Leskovec, J., K. J. Lang, and M. Mahoney, "Empirical Comparison of Algorithms for Network Community Detection," Proceedings of the 19th International Conference on World Wide Web, ACM, pp. 631–640, 2009.

[26] MacQueen, J., "Some Methods for Classication and Analysis of Multivariate Observations," Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, pp. 281–297, 1967.

[27] Mahajan, M., P. Nimbhorkar, and K. Varadarajan, "The Planar k-Means Problem Is NP-hard," Proceedings of the Third International Workshop on Algorithms and Computation (WALCOM), pp. 274–285, 2009.

[28] Matsumoto, M. and T. Nishimura, Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1), pp. 3-30, ACM Press, 1998.

[29] Maulik, U., Bandyopadhyay, S. and Mukhopadhyay, A. *Multiobjective Genetic Algorithms for Clustering : Applications in Data Mining and Bioinformatics,* Springer, 2011.

[30] Milligan, G. W., "Clustering Validation: Results and Implications for Applied Analyses," in Clustering and Classification, edited by Arabie, P., L. J. Hubert, and G. De Soete, pp. 341–375, 1996.

[31] Omran, M., A. Engelbrecht, and A. Salman, "Particle Swarm Optimization Method for Image Clustering," International Journal of Pattern Recognition and Artificial Intelligence, 19(3), pp. 297–321, 2005.

[32] Pakhira, M. K., S. Bandyopadhyay, U. Maulik, "Validity Index for Crisp and Fuzzy Clusters," Pattern Recognition, 37(3), pp. 487–501, 2004.

[33] Pal, S. K. and D. D. Majumder, "Fuzzy Sets and Decision Making Approaches in Vowel and Speaker Recognition," IEEE Transactions on Systems, Man, and Cybernetics, 7(8), pp. 625–629, 1977.

[34] Paternilia, S. and T. Krink, "Differential Evolution and Particle Swarm Optimization in Partitional Clustering," Computational Statistics and Data Analysis, 50(5), pp. 1220–1247, 2006.

[35] Pearson, K., "Contributions to the Mathematical Theory of Evolution," Philosophical Transactions of the Royal Society A, Vol. 185, pp. 71–110, 1894.

[36] Raghavan, V. V. and K. Birchand, "A Clustering Strategy Based on a Formalism of the Reproductive Process in a Natural System," Proceedings of the 2nd Annual International Conference on Information Storage Retrieval, pp. 10–22, 1979.

[37] Sarkar, M., B. Yegnanarayana, and D. Khemani, "A Clustering Algorithm Using an Evolutionary Programming Based Approach," Pattern Recognition Letters, 18(10), pp. 975–986, 1997.

[38] Selim, S. Z. and K. Alsultan, "A Simulated Annealing Algorithm for the Clustering Problem," Pattern Recognition, 24(10), pp. 1003–1008, 1991.

[39] Srikanth, R., R. George, N. Warsi, D. Prabhu, F. E. Petri, and B. P. Buckles, "A Variable-Length Genetic Algorithm for Clustering and Classification," Pattern Recognition Letters, 16(8), pp. 789–800, 1995.

[40] Storn, R. and Price. K, "Differential Evolution – A Simple and Efficient Heuristic For Global Optimization Over Continuous Spaces," Journal of Global Optimization, 11(4), pp. 341–359, 1997.

[41] Su, X. and T. M. Khoshgoftaar, "A Survey of Collaborative Filtering Techniques," Advances in Artificial Intelligence, Article No. 4, 2009.

[42] Tan, P. N., M. Steinbach, and V. Kumar, Introduction to Data Mining. Pearson Addison Wesley, 2006.

[43] Vendramin, L., R. Campello, and E. R. Hruschka, "Relative Clustering Validity Criteria: A Comparative Overview," Statistical Analysis and Data Mining, 3(4), pp. 209–235, 2010.

[44] Wu, J., *Advances in K-means Clustering: A Data Mining Thinking,* Springer, 2012.