A MEMETIC ALGORITHM FOR CLUSTERING WITH CLUSTER BASED FEATURE SELECTION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

İLYAS ALPER ŞENER

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
OPERATIONAL RESEARCH

AUGUST 2022

Approval of the thesis:

A MEMETIC ALGORITHM FOR CLUSTERING WITH CLUSTER BASED FEATURE SELECTION

submitted by **İLYAS ALPER ŞENER** in partial fulfillment of the requirements for the degree of **Master of Science in Operational Research Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Sinan Gürel
Head of Department, **Operational Research** _____

Prof. Dr. Cem İyigün
Supervisor, **Industrial Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Sinan Gürel
Industrial Engineering, METU _____

Prof. Dr. Cem İyigün
Industrial Engineering, METU _____

Prof. Dr. Esra Karasakal
Industrial Engineering, METU _____

Assoc. Prof. Dr. Yeşim Aydın Son
Health Informatics, METU _____

Assist. Prof. Dr. Kamyar Kargar Mohammadinezhad
Industrial Engineering, TED University _____

Date: 23.08.2022

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: İlyas Alper Şener

Signature :

ABSTRACT

A MEMETIC ALGORITHM FOR CLUSTERING WITH CLUSTER BASED FEATURE SELECTION

Şener, İlyas Alper
M.S., Department of Operational Research
Supervisor: Prof. Dr. Cem İyigün

August 2022, 111 pages

Clustering is a well known unsupervised learning method which aims to group the similar data points and separate the dissimilar ones. Data sets that are subject to clustering are mostly high dimensional and these dimensions include relevant and redundant features. Therefore, selection of related features is a significant problem to obtain successful clusters. In this study, it is considered that relevant features for each cluster can be varied as each cluster in a data set is grouped by different set of features, so the problem is named as clustering with cluster based feature selection problem. We approach the problem as a center based clustering and three tasks which are selection of relevant features, decision of cluster centroid locations, and assignment of data points to the clusters are considered. The problem is combinatorially NP-Hard and mathematical models are not capable to solve large-size problems. Moreover, developed heuristics in the literature obtain results with high variance. Therefore, a metaheuristic framework which follows the problem characteristics is proposed. A memetic algorithm that integrates a genetic approach and neighborhood search is proposed to solve data sets with high number of data points. A modified version of this algorithm is also developed for high dimensional data sets. Proposed algorithms have

been tested on different problem instances with different size and dimensions. Both simulated and real data sets are utilized for the tests. Experimental results have shown that the proposed approach obtains stable results with high accuracy and outperforms the state of the art.

Keywords: Metaheuristic approach, Memetic algorithm, Center Based Clustering, Feature selection, Rectilinear distance

ÖZ

KÜME ÖZGÜ ÖZNİTELİK SEÇİMİ İLE KÜMELEME PROBLEMİNE MEMETİK ALGORİTMA YAKLAŞIMI

Şener, İlyas Alper
Yüksek Lisans, Yöneylem Araştırması Bölümü
Tez Yöneticisi: Prof. Dr. Cem İyigün

Ağustos 2022 , 111 sayfa

Kümeleme metodu benzer veri noktalarını aynı gruba, birbirine benzemeyen veri noktalarını ise ayrı gruplara yerleştirmeyi hedefleyen yaygın kullanılan bir güdümsüz öğrenme metodudur. Kümeleme metoduna bağlı olan veri noktaları çoğunlukla yüksek boyutlu verilerden oluşmaktadır ve bu veriler ayırt edici ve alakasız öznitelikler içermektedir. Bu nedenle, başarılı bir kümeleme yapabilmek adına ayırt edici özniteliklerin seçilmesi de önemli bir problem arz etmektedir. Bu çalışmada ayırt edici özelliklerin her bir küme için değişken olabileceği ve her bir küme gruplara ayrılırken farklı öznitelikleri kullanacağı öngörülmüştür. Dolayısıyla odaklamılan problem küme özgü öznitelik seçimi ile kümeleme problemi olarak adlandırılmıştır. Problemde kümelemerin bir merkez etrafında oluştuğu yaklaşımıyla, ayırt edici özniteliklerin seçilmesi, küme merkezlerinin belilenmesi ve veri noktalarının kümelere atanması görevleri bir arada değerlendirilmektedir. Bu problem kombinatoryal NP-hard bir problemdir ve matematiksel modeller büyük ölçekli problemleri çözmekte yetersiz kalmaktadır. Ayrıca, literatürde geliştirilmiş olan sezgisel metodlar yüksek değişkenliğe sahip sonuçlar elde etmektedir. Bu nedenle, problemin çözülmesi adına

probleme odaklı geliştirilen bir metasezgisel yöntem izlenmiştir. Yüksek sayıda veri noktasına sahip veri setlerini kümelemek adına genetik algoritma ve komşuluk arama metodlarını içeren bir memetik algoritma önerilmiştir. Ayrıca, yüksek boyutlu veri setleri üzerinde başarılı sonuçlar elde etmek adına güncellenmiş bir memetik algoritma yaklaşımı geliştirilmiştir. Önerilen algoritmalar farklı problem büyüklüklerine ve boyutlarına sahip olan farklı veri örnekleri üzerinde test edilmiştir. Bu testler hem gerçek hem de yapay olarak oluşturulmuş veri setleri üzerinde uygulanmıştır. Yapılan testler sonucunda geliştirilen algoritmaların mevcut çalışmalardan üstün performans gösterdiği, ayrıca yüksek başarı ve düşük verimliliğe sahip sonuçlara ulaştığı gözlenmiştir.

Anahtar Kelimeler: Metasezgisel yaklaşım, Memetik algoritma, Merkeze bağlı kümeleme, Öznitelik seçimi, Rektilineer uzaklık

To my dear family

ACKNOWLEDGMENTS

First of all, I want to express my sincere thanks to my thesis advisor, Cem İyigun. I have learned a lot from him during my graduate studies about academia and life. It is certainly an honor to work with him and I am grateful to him for his support and guidance throughout this study.

I would also like to thank the members of the examining committee Prof. Dr. Sinan Gürel, Prof. Dr. Esra Karasakal, Assoc. Prof. Dr. Yeşim Aydın Son and Assist. Prof. Dr. Kamyar Kargar for their time in reviewing this work and valuable comments.

I want to thank my colleague Dilay Özkan for her support and friendship throughout my graduate studies in METU.

No words can be enough to thank my better half Dilan, who always shows her support to me. I always feel her endless love and patience when I was stuck.

I thank to my mother Fatma, and father Necdet for always believing in me and expressing their support every time whatever I choose to do in my life.

Finally, I would like to thank TÜBİTAK for financially supporting me through the 2210-A scholarship program during my graduate education.

TABLE OF CONTENTS

APPENDICES

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ALGORITHMS

ALGORITHMS

CHAPTER 1

INTRODUCTION

Massive quantities of data is generated in the recent years in line with developments in different technologies. Moreover, data generation is still in an increasing trend so that more data is created in each year. Higher number of data might be considered as a useful resource. However, handling of these data can cause problems as well since it creates difficult problems to solve, and it might be costly in terms of duration. In this environment, data mining methods have been developed in order to extract valuable information from a data set within shorter durations.

Problems in data mining mainly divided into two categories as supervised learning and unsupervised learning. In supervised learning, objects are labeled a priori, and these labels are utilized to train algorithms. Regression and classification algorithms are the most known ones in supervised learning.

On the other hand, unsupervised learning methods deal with data sets including unlabeled objects and aim to deduce meaningful insights from these data sets. Clustering is the most known unsupervised learning method in the literature which aims to group similar objects in clusters and separate dissimilar objects to different clusters.

In the literature, clustering methods are mainly divided into two categories which are hierarchical and partitional clustering. In hierarchical clustering, data set is organized in an hierarchical structure through iterations. It enables to create different numbers of clusters due to its dendrogram structure. However, merging or splitting of an hierarchical structure is not possible so that an incorrect decision in previous iterations cannot be fixed and it may converge to an undesirable clustering result. Partitional clustering methods aim to find cluster centroids that groups similar objects in the

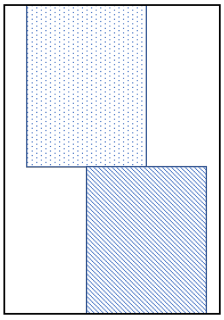
same groups. Assignments in partitional clustering can be done through either soft or hard partitioning. In soft clustering, data points are assigned to the clusters with a probability. That is, a data point can belong to more than one cluster. However, in hard clustering an object becomes a member of one cluster only. The main drawback of partitional clustering is the number of clusters that defines the data set is not known.

Clustering problem has some characteristics that are crucial to consider. Similarity measure is one of them since it directly affects the results of objective function. So that, all the procedures are identified according to selected similarity measure. One of the similarity measures is the distance between data points. Objects closer to each other can be considered as similar. That is, the larger the distance between objects, the more dissimilar they are. To calculate the distance between data points, Rectilinear and Euclidean distances are commonly used metrics in literature.

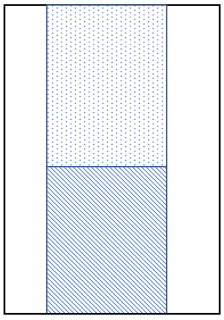
Determination of the number of clusters is another important decision to make, especially in partitional clustering methods. Since clustering aims to group similar objects and to separate dissimilar ones, number of clusters has notable effects on the results.

Another characteristic that should be considered is to decide on the features to be utilized. Some of the features may not contribute to describe the data at all. In fact, some features may even adversely affect the cluster quality. These features are defined as *masking variables*, and differentiating them from *true variables* is substantial [5]. In such cases, these attributes should be eliminated so that the clustering algorithm yields better results. Thus, when clustering is applied, relevant features should be selected. In other words, dimensionality can be reduced.

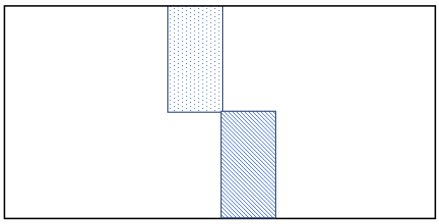
In this study, we have worked three types of clustering which are visualized in Figure 1.1. Figure 1.1 (a) represents the clustering with cluster based feature selection problem. In this problem hard partitioning is applied so that each data point is assigned to exactly one cluster. Moreover, selected features might be different for clusters and some of them may not be selected at all. Figure 1.1 (b) represents a subset of the clustering with cluster based feature selection problem. Here, all clusters must select the same features. Lastly, Figure 1.1 (c) represents a variation of (a). In this structure, we have faced with an high dimensional data set which includes more features than



(a) Clustering with Cluster Based Feature Selection



(b) Clustering with Feature Selection



(c) Clustering with Cluster Based Feature Selection in High Dimensional Data

Figure 1.1: Clustering Problems with Feature Selection

the data points.

Clustering with feature selection problem can be modeled as a nonlinear mixed integer model. However, the model is classified as NP-hard so that optimization models cannot provide convenient clusters within reasonable durations. Thus, various heuristic methods have developed for clustering problems in the literature.

We propose a metaheuristic framework in order to obtain successful clusters from a data set within reasonable durations. The proposed algorithm aims to minimize within-cluster distance by choosing cluster centroids, assignment of data points and selecting relevant features simultaneously.

Two memetic algorithms are developed in this study which focus on clustering with cluster based feature selection problem. The first memetic algorithm is developed to generate clusters on data sets that include high number of objects. In this algorithm,

each feasible solution is represented by a chromosome which includes the information of cluster centroids and feature selection. When the centroids and selected features are known, data assignment becomes a trivial problem that includes calculation of distances to centroids. Therefore, chromosome representation reduces the problem space significantly. Although genetic algorithm is able to find proper solutions, it has a deficiency on finding local minima. Hence, a neighborhood search algorithm is added in order to increase exploitation capabilities.

Although initial algorithm obtains successful clustering results for data sets with high number of objects, it has problem space limitations on high dimensional data sets. Therefore, a novel algorithm is developed considering high dimensional data sets. The most fundamental alteration in this algorithm is the chromosome representation in which chromosomes stores the information of cluster centroids and assignments. When the centroid and assignment information for each cluster is known, feature selection becomes an easy problem. In consideration of created clusters and their centroids, all features can be sorted by distance and the ones that gives the least distances will be selected as relevant features. Since information of selected features is neither stored in chromosomes nor added into search process, increasing number of features does not have a major effect on problem space. Thus, the updated algorithm is able obtain successful results on high dimensional data sets.

Finally, benefits of cluster based feature selection structure is analyzed. Data sets in bio-informatics domain generally consists of high number of features, and clusters may be grouped by different relevant feature sets. Therefore, we obtain the results of clustering with cluster based feature selection on the Wisconsin Diagnostic Breast Cancer data set. Then, these results are compared with the clustering problem, which selects same feature set for all clusters.

In this thesis, our contributions are as follows:

- We propose a unified metaheuristic framework for three different clustering problems mentioned above.
- We propose a memetic algorithm for the clustering with cluster based feature selection problem. The metaheuristic algorithm solves both clustering and clus-

ter based feature selection problems simultaneously.

- We also develop another memetic algorithm that obtains compelling results on high dimensional data sets.
- We compare the clustering with cluster based feature selection, and clustering with feature selection problems. We show that choosing features for each cluster separately obtains better results.

The rest of this study is organized as follows. A background information on clustering techniques is given and literature related with proposed algorithm is reviewed in Chapter 2. In Chapter 3, we give the problem definition and introduce a two mathematical models to represent the problem. The proposed memetic algorithm is explained in detail in Chapter 4. Subsequently, experimental result on proposed CCBFS algorithm is reported in Chapter 5. In Chapter 6, memetic algorithm is updated to work well with high dimensional data sets and details of the CCBFS-H algorithm is explained. Chapter 7 presents and discusses the experimental results of CCBFS-H algorithm. Finally, main findings of this study and future research directions are given in Chapter 8.

CHAPTER 2

LITERATURE REVIEW

Clustering is a well-known problem for various objectives in the literature, since grouping of objects is required to deduce meaningful information from a data set. The main purpose of clustering is to group unlabeled data into distinct clusters such that objects in each cluster should possess similar attributes. At the same time, objects that are dissimilar to each other should be separated into different groups [14].

In this chapter, initially clustering problem is defined and two solution approaches of clustering is introduced in Section 2.1. Subsequently, we briefly explain feature selection methods in clustering problems in Section 2.2. In 2.3, we discuss metaheuristic algorithms that are proposed in the literature. Then, developed metaheuristic methods for feature selection problems is introduced in Section 2.4. Finally, literature on clustering of data points with feature selection will be presented.

2.1 Clustering Problem

Unsupervised learning methods aim to group similar data points in different clusters. However, in the unsupervised learning, data has not been labeled, so that it is hard to group data without prior knowledge. Therefore, these methods aim to find meaningful relationships among objects or features from a data set. In brief, unsupervised learning is utilized to deduce information about the data and reveal hidden patterns among them in order to group similar objects [7].

Clustering is the most known unsupervised learning method which studies to organize and classify the data. It can be utilized in various domains such as customer and

image segmentation, biological data analysis, market research, fraud detection [14]. Data in these domains include different types of data like continuous, binary or categorical, and clustering methods had to deal with each type of data. Since a clustering algorithm should be compatible with different types of data, distance metrics gain significant importance.

Distance metrics can be grouped by two as quantitative and qualitative features. For quantitative features, Minkowski distance metric is the most common one to calculate the distance of two objects. The most common distance metrics -Euclidean and Rectilinear Distance metrics- are subsets of the Minkowski distance metric. On the other hand, for quantitative features, similarity of two objects should be considered to calculate a distance value. Cosine, Pearson Correlation, Extended Jaccard, and Dice Coefficient measures are the most common similarity measures for quantitative features [39].

Determining the objective function is another significant issue in clustering algorithms. The objective function should be utilized to maximize similarity among the objects in the same cluster, and minimize the similarity among the objects in different clusters. There are several metrics for the evaluation criteria of clustering problem and these metrics can be grouped into two as internal and external metrics [42].

Internal quality measures mostly aims to measure compactness of the clustering results. They evaluate the similarity of intra-cluster distances or separability of inter cluster distances, or both. Some of the internal quality measures are; Sum of squared error, scatter criteria matrix [39] and category utility metric [12].

On the other hand, external measures compares the clustering results with predefined clusters to gain information from an external clustering. Some of the external quality measures are; confusion matrix (or contingency table), normalized mutual information (NMI) [28], [49], rand index (RI) [36], adjusted rand index (ARI) [23], [46], Fowlkes–Mallows index [16], and Jaccard index [24].

Since various domains are in need of clustering, there are substantial amount of clustering approaches. In the literature, those approached are categorized under two main clustering approaches; Hierarchical Clustering and Partitional Clustering [52].

2.1.1 Hierarchical Clustering

Hierarchical clustering handles the clustering problem with tree-based structure which is called as dendrogram. The dendrogram structure includes nested series of partitions and each layer provides different clusters. Since the number of clusters should be determined in advance, hierarchical clustering has an advantage to determine number of clusters, because number of clusters can be obtained in each level of the tree-based structure [38]. In the literature, hierarchical clustering methods are mainly divided into two which are *agglomerative* and *divisive* methods.

In agglomerative clustering, a bottom-up approach is followed. In these methods, clustering starts with the clusters that include single data points. Subsequently, clusters are merged in each iteration and this process continues until an all-inclusive cluster is obtained. On the other hand, top-down approach is utilized in divisive methods. These methods starts with one all-inclusive cluster and subsequently data points are split into different clusters in each layer to build a dendrogram.

In both algorithms, defining the distance values of clusters are similar, since these values are utilized in either merge or split operation. Widely studied algorithms in both metrics are single link, complete link and average link. In single link metric, similarity of two clusters are determined by their nearest neighbor objects. Contrarily, similarity of the most distant objects are evaluated in complete link clustering metric. In average link metric, average of all pair-wise distances in two clusters are utilized. Although it provides more discrete results, this measure is the most expensive one especially in large data sets [38].

The most prominent advantage of hierarchical clustering is that the number of clusters are determined by the tree-based structure. However, it also carries some weaknesses. In hierarchical clustering, each label is created via greedy algorithms and a misleading step is not reversible. Therefore, errors in the previous steps can not be corrected. Furthermore, since each data is handled, hierarchical clustering are computationally expensive. To overcome these defectives, several algorithms are developed. CURE [18], BIRCH [54], COBWEB [15], CHAMELEON [26] and SOM [29] are well-known algorithms in this area.

2.1.2 Partitional Clustering

In the partitional clustering method, clustering problem is considered as an optimization problem with respect to predetermined objective function. These methods are advantageous to handle large data sets, and needs less memory and computational time with respect to hierarchic methods. In partitional clustering, a two layered problem is studied which includes selection of cluster centroids and assignment of objects. Unlike hierarchical methods, number of clusters should be defined in advance which is a challenge of partitional clustering. Moreover, since the methods aims to optimize objective function, partitional clustering works well with quantitative data sets.

Partitional clustering methods can be divided into two as *hard partitioning* and *soft partitioning* with respect to the assignment type. In hard partitioning, each object must be assigned to exactly one cluster. On the other hand, objects have membership probabilities to clusters in soft partitioning, so that each object might be assigned to more than one cluster.

In hard clustering, k-means [33] and partitioning around medoids (PAM) [37] are two prominent algorithms in literature. In both of the algorithms, similar iterative structure is following. It starts with choosing k representative cluster centroids and assignment of data points to their nearest centroids. Subsequently, cluster centers are updated with respect to the assignments, and these two steps are repeated in iterative basis until stopping criteria are met. There are two main differences in these algorithms. In PAM, which is a type of k-medoid clustering, objects are selected as centroids while the point that is the average of the assignments is selected as centroid in k-means. Another difference is that PAM utilizes the sum of the distances in clusters, and k-means aims to minimize sum of square distances.

In soft clustering, memberships of objects take values between 0 and 1, since one object can be assigned to more than one cluster. Soft clustering approach was introduced in 1984 namely as the fuzzy c-means algorithm [8]. Probabilistic distance clustering [4] and K-Harmonic means [53] are other prominent algorithms in soft clustering. These methods are able to provide successful results especially when the data points are not separated well to form distant clusters.

2.2 Feature Selection in Clustering Problems

A data point in a data set is defined through different attributes which are named as features. Number of features determines the dimensions of a data set, and increasing number of features hides the meaningful information. Therefore, high number of features makes the clustering problem complicated and this is known in the literature as *curse of dimensionality* [22].

Redundant features may be included in a data set which degrades the relationship between data points and it causes to spend more computational time while applying clustering algorithms. In order to obtain relevant features and increase their effect two methods are utilized which are *feature extraction* and *feature selection*.

Feature extraction methods combines the features to obtain low number of features. After that, clustering is studied on the combined new features. However, this transformation adds noise to the data set since combination of the features are utilized [48]. Principal Component Analysis (PCA) is the most known method for feature extraction which aims to create new features as linear functions of the initial ones and maximize the variance among uncorrelated data. After that, eigenvalue (or eigenvector) problem is solved through created variables [25].

The motivation behind the feature selection methods is to eliminate redundant features from the data set. Therefore, relevant features will be kept in the data set, and this will improve the quality of the clustering results. There are three main groups for feature selection methods, which are *filter methods*, *wrapper methods*, and *hybrid methods*[2].

Filter Methods

Filter methods applied to eliminate features before the onset of clustering algorithm. Thus, filter methods initially process the data set and evaluate each feature with a score. Then, clustering algorithm works with the features that have scores above a pre-determined threshold value [19].

In filter methods, features are evaluated through univariate or multivariate methods. In univariate filter methods, each feature is assessed separately. On the contrary, mul-

tivariate filter methods takes the correlation among features into consideration while scoring the features. Univariate method is a faster method since it evaluates features one by one, but multivariate method is promising to discard redundant features from the data set. [2]

Both of the filter methods are simpler and faster methods, since they are applied only once at the beginning. Therefore, these methods are applicable to the data sets with high dimensions. However, due to the lack of communication between clustering model, it might be disadvantageous in some data sets.

Wrapper Methods

Wrapper methods are working together with the clustering algorithm and utilize the results of clustering to evaluate the convenience of selected features. In wrapper methods, clustering starts with the subset selection of features. Subsequently, clustering results are evaluated for the features in selected subset. According to evaluation, a new subset is selected and this operation continues in iterative basis until a stopping criterion is met. As all subsets cannot be evaluated in high dimensional data, generally heuristic methods are applied in wrapper methods.

Feature selection by wrapper methods can be grouped into three categories, which are *sequential* [13], *bio-inspired* [27] and iterative methods [30]. In sequential search methods selected features are changed sequentially. In order not to stuck in local optima, bio-inspired methods are developed. Iterative methods considers the feature selection as an estimation problem to not apply a combinatorial search [45].

Wrapper methods are much expensive than the filter methods, since they are working iteratively with clustering algorithm, which is the main disadvantage of these methods. However, this interaction enables wrapper models to provide better clustering results.

Hybrid Methods

Considering the advantages and disadvantages of both methods, hybrid feature selection methods have been developed in the literature. Hybrid methods initially apply a filtering method to select several feature sets. Subsequently, results of each selected

subsets are evaluated and the subset that providing best results is selected. Thanks to utilizing advantageous sides of both methods, hybrid models provide better clustering results with respect to filter methods, and it is less computationally expensive than wrapper methods.

In this study, two evolutionary algorithms are proposed for clustering with cluster based feature selection problem. Therefore, a literature review on metaheuristic approaches for clustering and feature selection problems is provided in the following sections.

2.3 Metaheuristic Algorithms for Clustering

The clustering problem is an NP-Hard problem so that heuristic approaches are essential to obtain promising clustering solutions. Tabu Search, Variable Neighborhood Search, Genetic Algorithm (GA), Simulated Annealing and Ant Colony Optimization are the common metaheuristic methods in the literature.

Rolland et al. [40] develop a tabu search heuristic for the p-median problem. Since swap moves are computationally expensive, they utilize add and drop moves in their TS. Then, they apply strategic oscillation method in order not to stuck in a local optima. In line with strategic oscillation, feasibility may not be maintained so that algorithm is able to move from one local optima to another one.

Shi and Ólafsson [44] propose a randomized method named as *nested partitions*. Initially, they divide the feasible region into subregions. Then, they select the most promising subregion by using random sampling of the entire feasible region. Moreover, they adapt a local search method to find better clusters in the selected subregion. It is shown that the proposed algorithm converges to global optima in finite time and also stopping rules are defined to obtain a solution within a predefined time interval.

Variable Neighborhood Search (VNS) is introduced by Hansen and Mladenovic [20]. VNS aims to explore increasingly distant neighborhoods to find an improvement. Apart from other local search methods, VNS is capable to quit from a local optima in most of the cases. Since promising clustering results are found via VNS method, sig-

nificant improvements are developed through years. According to Hansen et al. [21] a VNS heuristic includes three phases that are worked iteratively. In the first phase, shaking procedure is applied to resolve local minima. Then, improvement procedure is utilized to improve the solution. After that, neighborhood change procedure is applied. In this procedure a decision for the exploration of the neighborhood is done.

Chiyoshi and Galvao [11] combine simulated annealing methods with vertex substitution. They utilize a cooling schedule that incorporate with the notion of temperature adjustments for simulated annealing. In order not to choose pairs of vertices randomly, the authors utilize vertex substitution method.

Levanova and Loresh [31] develop an implementation for simulated annealing and ant colony optimization for p-median problem. Authors utilize SA to search new solution spaces by leaving local optimum points. They apply ant system algorithms to find shortest paths between two feasible solutions. In this method, shortest paths provide highest level of pheromones so that ants probably move to the nearest clusters over some time.

Sheng and Liu [43] propose a hybrid genetic algorithm for k-medoids clustering problem. They follow a two-fold parent selection method and transfer the best parents and offsprings to the next generations. To decrease the time of convergence, they develop a modified David-Bouldin index for fitness calculations. Moreover, a priori assumptions for number of clusters are not taken and number of clusters are tested between the range of 2 and k_{max} (which equals to $\sqrt{n}$). The experiments have illustrated the effectiveness of the proposed algorithm by comparing it with related clustering algorithms.

Beg and Islam [3] argue that a poor-quality initial population may cause to poor quality genetic algorithm results. Therefore, they propose an algorithm that creates successful offsprings in the initial generation. In this algorithm, 50% of the chromosomes are created by deterministic approach while the remaining 50% are created via random selection phase. In the deterministic approach, first, a set of chromosomes are generated through k-means where each set includes a predefined number of chromosomes. Subsequently, chromosomes are sorted in accordance with their fitness value and chromosomes with best fitness values are selected to the initial generation.

Akay et al. [1] define a new fitness function which can be utilized in genetic algorithms. The developed fitness function includes three different components which are between cluster distance, within cluster distance and silhouette width. The aim of the new function is minimizing the ratio of intra to inter cluster distances. The results have shown that the proposed algorithm provides better results than some other clustering algorithms.

2.4 Metaheuristic Algorithms for Feature Selection in Clustering

As mentioned in Section 2.2, heuristic methods are widely utilized in feature selection problems, since these problems are too complicated to handle. Brusca (2004) [9] develops a heuristic approach to eliminate masking variables in a k-means clustering problem. It is assumed that clusters are known a priori and the data points includes solely binary values. The proposed algorithm initially selects the subset of features. Then, one feature from unselected ones are tested and added into the subset in iterative basis. In each step, success of selected features are recorded and the best subset is utilized as the result of the proposed algorithm.

Chen et al. [10] utilize support vector machines and genetic algorithm for features selection problem in clustering. Two heuristic approaches which are GA and an hybrid GA/SVM approach is developed to distinguish representative genes of a cancer from irrelevant ones. Genetic algorithm is utilized to optimize locations of centroids in consideration of feature selection. Selected features in each clustering were then ranked according to a distance metric, and top ranked features are taken as representative genes in the clustering problem.

Rostami and Moradi [41] come up with a three-step feature selection algorithm in clustering, where k-means and genetic algorithm are both used. Chromosomes are binary, and they represent the selected features. They run the k-means algorithm to cluster the features into a predefined number of clusters which indicates the total number of features to be chosen. They applied one point crossover as crossover operator.

Wu et al. [51] study feature selection for clustering by using genetic algorithms. In

order to speed up the convergence process in GA, two methods are utilized. First, Taguchi method is followed to eliminate some of the poor offsprings without testing them. As a second method, similar offsprings are grouped and better offsprings are selected for retrieval accuracy computations. Hubert's Γ statistics is utilized in this method to calculate cluster validity as a measure for fitness.

Benati et al. [6] proposed two mixed integer linear programming formulations for feature selection in clustering problems. However, it is observed that if clusters overlap, even small problems may not be solved. Therefore, two heuristics algorithms are developed. On the first algorithm they divide the problem into two parts as best assignment and best feature problems. Then, calculated distances for selected best features are sorted. In the second heuristic method, features are added to or dropped from a selected feature set.

Sun et al. [47] proposed a genetic algorithm for feature selection for high-dimensional data clustering (GA-FSF clustering). They came up with a different fitness function that considers the features in the chromosomes. Chromosome representation contains both the features that are used and data points which are chosen as cluster centers. They select 10% of the best individuals from the previous population, and randomly choose other individuals from the current population. During the process, it is ensured that each chromosome possesses different number of features.

Apart from the clustering problems, we have utilized several methods developed for metaheuristics in the literature. Wang et al. (2019) [50] present a multi-offspring genetic algorithm with two-point crossover in consideration of biologic theory. In order to obtain best offsprings, two-point crossover is applied and multiple number of offsprings are generated in each crossover operation. They also study the relationship between the number offprings in each iteration and computational speed.

Li et al. [32] propose a localized feature selection method instead of a global feature selection. They develop an algorithm that computes scatter separability of each cluster, and find feature subsets for these clusters. The proposed algorithm is not only capable of eliminating redundant features, but also provides better understanding on creation of the data set. In their experiments they emphasize the benefits of local feature selection.

Frigui and Nasraoui [17] develop an algorithm that perform clustering and feature weighting simultaneously. They utilize different subset of features for each cluster. Different to our problem, they do not select certain features for a cluster, but they assign weights to the features.

Öz [35] works on clustering with cluster based feature selection problem. She studies hard partitioning and determines the number clusters and number of selected features a priori. Initially, a nonlinear mixed integer mathematical model is proposed and linearized. Since the problem is highly complex, proposed mathematical models are not able to provide convenient results within shorter durations. Thus, two different heuristic algorithms have developed and experimented on synthetic data sets.

In this study, we work on clustering with cluster based feature selection problem. Our work differs from the existing literature based on applying a metaheuristic algorithm with a problem specific approach. Details of the problem and a unified metaheuristic framework for this problem are provided in the following chapters.

CHAPTER 3

PROBLEM DEFINITION

In Chapter 2, background information for clustering and several solution methods including metaheuristics and local search algorithms are given. In this chapter, we first define the problem in Section 3.1. Subsequently, a nonlinear mixed integer mathematical model and a linearized model are provided in Section 3.2 which represents the studied problem.

3.1 Problem Statement

The aim of this study is to cluster the data points based on relevant features of each cluster. In the problem setting, each cluster includes one data point as a cluster centroid and each data point must be assigned to exactly one cluster. Moreover, one data point can be selected as centroid for at most one cluster. In this problem, it is assumed that each data point possesses values for several features. However, different subsets of features are meaningful for clusters separately. Therefore, relevant features should be selected considering each cluster individually.

In this clustering problem, dimensions of a data set determines the set of data points, N, and set of features, M. Furthermore, parameters of number of clusters, p, and number of selected features, q, should be defined in advance. For instance, if number of clusters are not given in advance, and the decision is left to a model, the model would define $|N| = n$ clusters to minimize the total distance. But, it is an undesirable result since it will create clusters with only one data point so that any meaningful information about the data set cannot be obtained. Moreover, if a model determines the number of selected features, it would select only one of them, since an additional

feature selection increases the distance value in all cases.

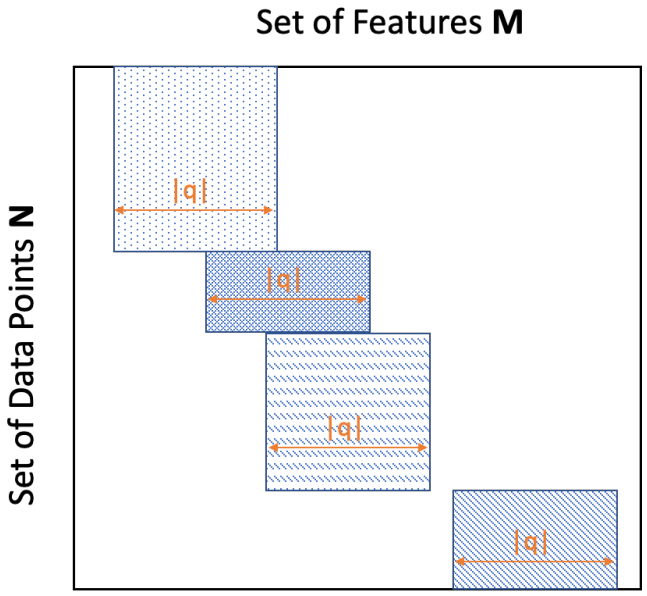


Figure 3.1: Representation of the Clustering with Cluster Based Feature Selection Problem

Figure 3.1 provides an example for a data set which has $|N| = n$ data points and $|M| = m$ features, and it is divided into $p = 4$ clusters. Each cluster utilizes exactly q features, and inequality of $q \leq m$ is always valid.

As shown in Figure 3.1, selected features in each cluster may vary. One feature might be utilized in more than one cluster, so that there may be common features to be used by each cluster, but it is not a requirement. Moreover, a feature may not be utilized in any cluster at all.

In the problem setting, hard partitioning is applied by assigning each data point to exactly one cluster, so that disjoint clusters are constructed. Therefore, sum of the number of data points in clusters must be equal to total number of data points in the set, $|p_1| + |p_2| + ... = |N|$.

We utilize rectilinear ($L_1 - Norm$) distance as a similarity measure between data points. Since all features are not selected, distance of a data point to its cluster centroid is calculated with respect to selected ones. Let v_{ik} denotes the value of i^{th} data point's k^{th} feature, and there are a total of m features. Then, rectilinear distance d_{ij}

between two data points i and j can be calculated through following formula.

$$d_{ij} = \sum_{k=1}^{m} d_{ijk} \tag{3.1}$$

$$where \quad d_{ijk} = \mid v_{ik} - v_{jk} \mid \tag{3.2}$$

Distance between the centroid and all data points is calculated and sum of these distances represents the value of within-cluster distance. Minimizing the sum of within-cluster distances is the aim of this study so that developed methods select the relevant features and assignments accordingly.

In brief, the problem is named as clustering with cluster based feature selection (CCBFS). In this problem, we create center-based clusters and each cluster's center is one of its points. Moreover, predetermined number of features are selected from the feature set for each cluster separately. To obtain logical results from a model, we determine the number of clusters p and number of selected features q in advance. The objective of the problem is to minimize the total within-cluster distances.

Defined problem is highly similar to well-known *p-median problem*, as data points are assigned to several clusters, and a cluster's center is selected from one of the assigned data points. Regarding the *p-median problem*, cluster centroids can be considered as facility locations, and data points can be considered as customers. On the other hand, *p-median problem* has two features for data points, which are x and y coordinates on the plane, and these two features have to be considered. However, in our problem, only relevant features should be taken into account to calculate distances and redundant features should be disregarded for each cluster.

We can define our problem through three tasks, which should be carried simultaneously. In the first task (i), data points should be assigned to one of the p clusters. In the second task (ii), one data point should be located as centroid for each cluster. The last task (iii) aims to select relevant features on clustering and these features may vary for each cluster.

This combinatorial problem can be represented through a mathematical model which is given in the Section 3.2. Objective of the model is to minimize the total distance between data points and cluster centroids, within-cluster distance, and determine the

selected features for each cluster on this purpose.

3.2 Mathematical Model of the Problem

In this section, nonlinear mixed integer model for clustering with cluster based feature selection problem is provided. Subsequently, linearized version of the model, and its complexity is reported.

3.2.1 Sets and Model Parameters

Since we aim to cluster data points in a data set by selecting relevant features, the mathematical model will include two sets; set of data points (rows), N, and set of features (columns), M.

In the above mentioned problem, number of selected features, q, and number of clusters, p should be determined in advance. Otherwise, the model will define n clusters and/or select only one feature. Furthermore, distance values between data points for each feature should be provided to the model. Thus, d_{ijk} is a parameter which represents the distance between the k^{th} feature of object i and j.

In the Table 3.1, the notation used for sets and parameters are summarized.

3.2.2 Decision Variables

Three binary decision variables are defined to model the problem. First variable, x_{ij}, is introduced to determine assignments of data points to cluster centroids. Since hard partitioning is utilized, a data point can be assigned exactly one cluster. Aim of the second variable, y_j, is to determine cluster centroids. Finally, third variable, z_{jk}, is introduced to determine the selected features in j. It enables model to select subsets of features in each cluster separately.

In the Table 3.1, the notation used for decision variables are also summarized.

Table 3.1: Model Parameters

Sets:	
N	Set of data points
M	Set of features
Parameters:	
d_{ijk}	Distance between the data point i to j on feature k, $i \in N$, $j \in N$, $k \in m$
q	Number of selected features
p	Number of clusters
Decision Variables:	
x_{ij}	Binary allocation variable takes value of 1 if data point i is assigned to the data point j, 0 otherwise, $i \in N,\ j \in N$
y_j	Binary variable takes value of 1 if data point j is selected as a cluster centroid, 0 otherwise, $j \in N$
z_{jk}	Binary variable takes value of 1 if the feature k is selected for cluster centroid j, 0 otherwise, $j \in N,\ k \in m$

3.2.3 Nonlinear Mixed Integer Model

Öz (2019) [35] provides a nonlinear mixed integer model for clustering with cluster based feature selection problem. The objective of the mathematical model is to minimize the sum of distances between objects and their corresponding cluster centroids. It also includes the selected features for each cluster. The nonlinear mixed integer model is represented as;

$$\text{min} \quad \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{M} d_{ijk} z_{jk} x_{ij} \tag{3.3}$$

$$\text{s.t.} \quad x_{ij} \leq y_j \qquad \forall i, j \in N \tag{3.4}$$

$$\sum_{j \in N} x_{ij} = 1 \qquad \forall i \in N \tag{3.5}$$

$$\sum_{j \in N} y_j = p \tag{3.6}$$

$$\sum_{k \in M} z_{jk} = q\, y_j \qquad \forall j \in N \tag{3.7}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in N \tag{3.8}$$

$$y_j \in \{0, 1\} \qquad \forall j \in N \tag{3.9}$$

$$z_{jk} \in \{0, 1\} \qquad \forall j \in N, \forall k \in M \tag{3.10}$$

The objective function (3.3) sums up the distance between data points and their cluster centroids considering the features to be used in each cluster. (3.4) assures that if object j is a cluster centroid, then object i can be assigned to the cluster with center j. Otherwise, any data point cannot be assigned to this cluster. (3.5) makes an object belongs to exactly one cluster. Since p is the total number of clusters, (3.6) ensures that there are exactly p cluster centers. (3.7) guarantees that the cluster center j has exactly q features. (3.8), (3.9), and (3.10) show that the decision variables are binary.

3.2.4 Linearized Mixed Integer Model

In the nonlinear model, objective function (3.3) includes two decision variables. Öz (2019) [35] has applied several linearization methods in order to obtain a linearized model. She defines an additional decision variable w_{ijk} which defined as
$w_{ijk} = z_{jk}\, x_{ij}$.

The linearized model is formulated as;

$$\min \quad \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{M} d_{ijk}\, w_{ijk} \tag{3.11}$$

$$\text{s.t.} \quad \sum_{i\in N} x_{ij} \leq n\, y_j \qquad \forall j \in N \tag{3.12}$$

$$\sum_{j\in N} x_{ij} = 1 \qquad \forall i \in N \tag{3.13}$$

$$\sum_{j\in N} y_j = p \tag{3.14}$$

$$\sum_{k\in M} z_{jk} = q\, y_j \qquad \forall j \in N \tag{3.15}$$

$$\sum_{k\in M} w_{ijk} = q\, x_{ij} \qquad \forall i, j \in N \tag{3.16}$$

$$w_{ijk} \leq z_{jk} \qquad \forall i, j \in N, \forall k \in M \tag{3.17}$$

$$w_{ijk} \geq 0 \qquad \forall i, j \in N, \forall k \in M \tag{3.18}$$

$$x_{ij} \in \{0, 1\} \qquad \forall j \in N \tag{3.19}$$

$$y_j \in [0, 1] \qquad \forall j \in N \tag{3.20}$$

$$z_{jk} \in \{0, 1\} \qquad \forall j \in N, \forall k \in M \tag{3.21}$$

Updated objective function (3.11) includes parameter d_{ijk} and decision variable of w_{ijk}, therefore it is converted to a linear function. Constraints (3.13), (3.14), (3.15), (3.19), and (3.21) are the same with the ones in nonlinear model. (3.12) ensures that if a data point is selected as centroid, at most n data points can be assigned to that cluster. (3.16) guarantees that if i^{th} data point is assigned to j^{th} cluster, exactly q number of w_{ijk} variable equal to 1. In (3.17) z_{jk} takes values of 0 and 1, so that w_{ijk} is bounded. Since values of d_{ijk} is always nonnegative in the objective function and it is a minimization problem, w_{ijk} will always be equal to 0 or 1. Finally, y_j is defined as continuous variable in (3.20), since it always takes values of 0 or 1 in a minimization problem.

Table 3.2: Number of Constraints and Variables of the Linearized Model

	Number of Constraints		**Number of Variables**	
	Equality	*Inequality*	*Binary*	*Continuous*
Linearized Model	n (2 + n) + 1	n (1 + nm)	n (n + m)	n (1 + nm)

In Table 3.2 the number of constraints and number of variables of the linearized model is reported. Although the model is converted to a linearized one, the model is complex for higher values of n, since both constraints and variables are increased by the square of n. Moreover, number of features, m, has also significant effects on both number of variables and inequalities.

If we consider the defined problem from a combinatorial perspective, clustering with cluster based feature selection is a NP-hard combinatorial problem, since this problem can be reduced to *p-median problem* [34]. When the number of clusters and distance measure are already given, the optimal clusters can be determined by complete enumeration. All combinations of the data points and attributes can be experimented. However, it would be costly even when there exist a few features and the data set is small. Öz (2019) [35] formulates the total number of possible solutions as in 3.22. In the formula, n, p, q and m represent number of data points, number of clusters, number of features and number of selected features, respectively.

$$\left[\frac{1}{P!}\sum_{k=0}^{P}(-1)^k \begin{pmatrix} P \\ k \end{pmatrix} (P-k)^n\right] \left[\begin{pmatrix} m \\ q \end{pmatrix}^P\right] \tag{3.22}$$

The first term in 3.22 calculates the total number of partitions of n data points into p clusters. This term is increased exponentially with increasing values of n. Therefore, problem size become immense on the data sets with high number of data points. On the other hand, second term illustrates the total number of ways to select q features from m features for all p clusters. This term indicates that the problem size will be enlarged by a factorial factor on increasing values of number of features, m.

To illustrate, consider there are only 20 objects, 2 clusters, 4 features, and only 2 of these features are selected. Even in this small data set, there are approximately $1.9 * 10^7$ possible solutions. For a bigger data set of $N = 200, p = 4, m = 12, q = 4$, which is analyzed in this study, total number of possible solutions has reached to $6.46 * 10^{129}$.

In this manner, mathematical models are able to solve small-sized problems and Öz (2019) [35] is able obtain optimal clustering for small-sized simulated data sets with CPLEX. However, since it is an NP-Hard problem, these models are not able to

provide clustering results in meaningful durations for bigger data sets. Therefore, a metaheuristic algorithm might provide better results for this problem. In Chapter 5, performances of the proposed algorithm and the models in [35] are tested. The details of the results are provided in Section 5.5.

CHAPTER 4

MEMETIC ALGORITHM FOR CLUSTERING WITH CLUSTER BASED FEATURE SELECTION (CCBFS)

In this chapter, our memetic algorithm for clustering with cluster based feature selection (CCBFS) is presented. In the Section 4.1, the algorithm is introduced and the process of the algorithm is explained. Subsequently, in the Section 4.2, each step of the algorithm is described in detail.

4.1 Overview of CCBFS

Main aim of the proposed algorithm is to cluster data points in a data set by minimizing the within-cluster distance values. There are $|N| = n$ data points in the data set and these data points are grouped into p clusters. Each cluster should have exactly one centroid from data point set, and a data point can be assigned as centroid of at most one cluster. Furthermore, each data point must be assigned to exactly one cluster. On the other hand, there are $|M| = m$ features in the data set which assumed to include relevant and redundant features. We also test the algorithm's ability to deduce the relevant features among all, and eliminate the non-informative features, by minimizing the total within-cluster distance.

As discussed in Chapter 3, studied problem is an NP-Hard problem, and a metaheuristic approach may provide successful results in shorter durations. That is why, we developed a memetic algorithm which includes two main components of genetic algorithm and neighborhood search. Thanks to our representation of chromosomes, which is explained in detail in Section 4.2.1, we are able to reduce the solution space significantly to apply a memetic algorithm. In the proposed method, chromosomes

include the information of selected features and cluster centroids. When the selected features and cluster centroids are known, a distance value can be calculated by assigning each data point to their nearest cluster centroid. Therefore, assignment task in our problem can be done with respect to selected features and centroids.

Although genetic algorithm decreases the solution space dramatically, it has a deficiency on finding local minima in our problem. Therefore, we add a neighborhood search into our algorithm, which increases exploitation capabilities of the algorithm. The local search method mainly checks the potential centroid candidates in the neighborhood, and in case a fitter centroid selection is found, it is added into list of possible solutions.

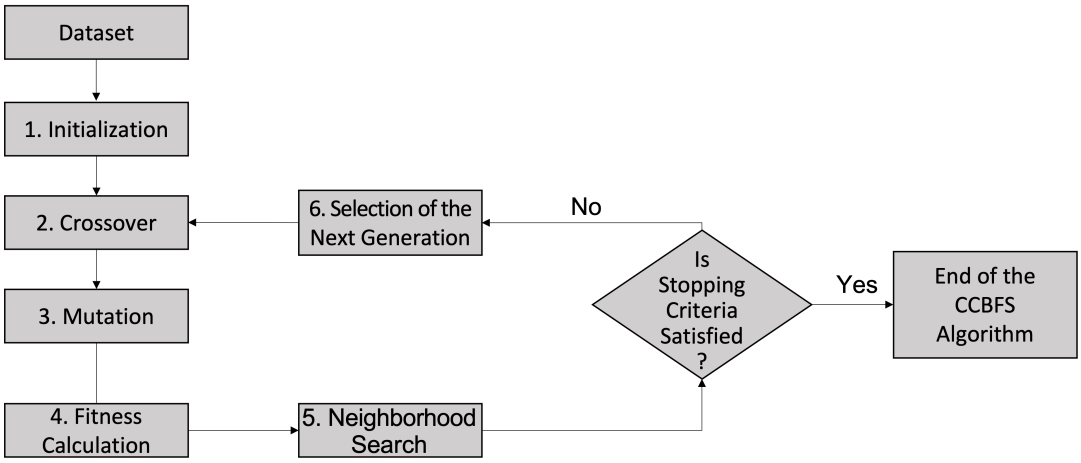


Figure 4.1: Flowchart of the CCBFS Algorithm

The flowchart of the memetic algorithm is shown in the Figure 4.1. As illustrated in the flowchart, the process starts with taking the data set and the parameter settings as inputs.

Since aim of the proposed algorithm is to minimize distance, range differences among features in the data set might be harmful for correct clustering. This situation is a common problem especially in real-life data sets. Therefore, standardization of the data set must be considered before running the CCBFS algorithm to avoid from scaling factor.

In Step 1, initial population is created with three different methods in order to increase exploration capability of the algorithm. In this step, unfit individuals are discarded

and only feasible individuals are allowed to create offsprings. Total number of individuals is equal to the population size (pop_{size}).

Crossover operation is conducted as Step 2. Firstly, a mating pool is obtained and four offsprings are created from two parents via uniform crossover. After that, mutation operation is applied to the offsprings with a small probability as Step 3. In this step, offsprings might be mutated with one of the three different types of mutation operator.

After the mutation operator, a fitness value is calculated for each individual in the population as Step 4. In this step, within-cluster $L_1 - Norm$ Distance values for each cluster is calculated and the total distance of an individual is taken as its fitness value.

In Step 5, neighborhood search algorithm is applied to the fittest individual in a population. Since each individual holds the selected feature and cluster centroid information, and transfer them via uniform crossover, potential centroids in the neighborhood are not considered through genetic algorithm. The purpose of the neighborhood search is testing the potential centroids that are close to the current ones which might provide fitter cluster centroids to a certain assignment. Then, the parents of the next generation are selected from the feasible individuals of the current generation according to selection criteria.

The creation of offspring process continues until the stopping criteria is satisfied which is the total number of generations (gen_n). At the end of the algorithm, the fittest offspring through all generations is taken as the best individual and its phenotype is reported as the best result of the CCBFS algorithm.

An overview of the CCBFS algorithm is summarized in Algorithm 1. In Section 4.2, pseudo-codes for each step in Algorithm 1 are also explained in detail.

4.2 Description of CCBFS Algorithm

In this section, each step of CCBFS algorithm is explained in detail.

Algorithm 1: CCBFS Algorithm

Input : Data set

Output: within-cluster distances, selected centroids and cluster specific features, assignments of data points

Step 1: Setting the parameters
Step 2: Initialization(Data set,q,p,initialization types) *# Algorithm 2*
for *i=1,...,*gen_n **do**
  for *j=1,...,*pop_{size}*/2* **do**
    Step 3: Uniform Crossover(Current population) *# Algorithm 3*
  end for
  for *k=1,...,*pop_{size} **do**
    Step 4: Mutation(Offsprings) *# Algorithm 4*
    Step 5: Fitness Calculation(Data Set, Offspring's Chromosome) *# Algorithm 5*
  end for
  Sort Offsprings by fitness value
  Step 6: Neighborhood Search(Chromosome of the fittest offspring) *# Algorithm 6*
  Step 7: Selection of the Next Generation(Current Population) *# Algorithm 7*
end for

4.2.1 Chromosome Representation

Appropriate chromosome representation has a crucial importance to obtain successful clusters within a reasonable time period. As an example, if the cluster numbers of each object are kept in the chromosomes, the algorithm will face with several difficulties. It is possible that one solution can be represented by more than one chromosome since cluster numbers are arbitrary. In fact, crossover or mutation operators may result in infeasible solutions, i.e., there is a possibility that total number of clusters may decrease. To avoid this problem, feasibility of the problem should be checked during the fitness calculation step. Thus, keeping the cluster numbers is not a good representation. Another chromosome representation could be designing a 0-1 matrix in

which objects in the same cluster take a value of 1. However, this matrix becomes a sparse matrix since there would be so many entries of 0 and it would require memory. Hence, choosing similarity matrix as a chromosome representation is not suitable. To handle crossover and memory requirement problems, the features that cluster centers use are kept in the chromosome. Since the cluster centers are chosen from the data points, a chromosome keeps the features of the data points chosen as cluster centers and their indices. Figure 4.2 illustrates the chromosome representation in the CCBFS algorithm.

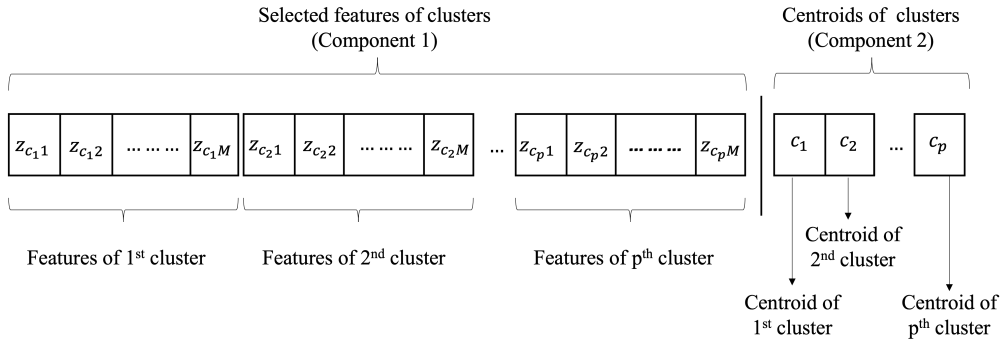


Figure 4.2: Chromosome representation

The chromosome representation consists of two components. Selected features for each cluster are recorded in the first component. In this component, features of each cluster centroid are added to the chromosome part by part. Each part that represents the selected features is m-dimensional because data sets includes a total of m features. Therefore, first m values characterize the selected and unselected features for the first cluster, second m values characterize the second cluster and so on. As an example $z_{c_1 k}$ refers to the k^{th} feature of the first cluster. It becomes 1 if the first cluster uses k^{th} feature, otherwise it becomes 0. Therefore:

$$\sum_{k \in M} z_{c_i k} = q \qquad \forall i \tag{4.1}$$

$$z_{c_i k} \in \{0, 1\} \qquad \forall i, k \tag{4.2}$$

Length of the first component is equal to the product of number of clusters and number of binary attributes, which is $(p * M)$.

The second component holds the indices of data points assigned as cluster centers.

For instance, c_1 gives the index of the data point that becomes the first cluster center, i.e., $y_{c_1} = 1$. Then $z_{c_1 1}$, $z_{c_1 2}$, ..., $z_{c_1 M}$ refers to the selected features of data point with index c_1. The length of second component is the number of clusters, p. Thus, the chromosome length is the summation of both component lengths, which is $(p*M+p)$.

To give an example about the chromosome representation, Figure 4.3 is provided. In this example, there are $p = 3$ clusters, and each cluster uses $q = 2$ features. Cluster centers are the data points with indexes $121, 23$ and 48. While the first cluster uses 1^{st} and 5^{th} features, the second cluster utilizes 1^{st} and 2^{nd} features, and 2^{nd} and 5^{th} features are selected in third cluster.

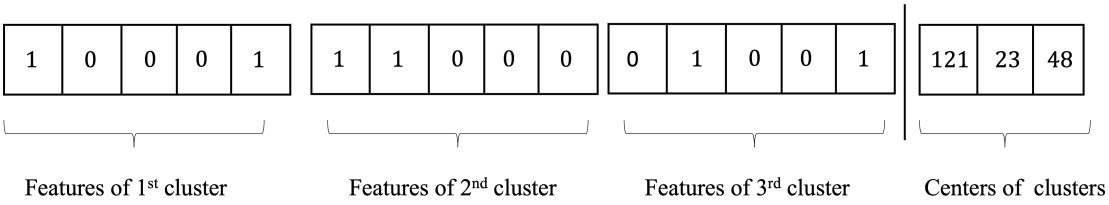


Figure 4.3: An example for chromosome representation

This representation keeps less memory, since it does not memorize the clusters that data points are assigned to. Therefore, to determine which object belongs to which cluster, data points should be assigned to the clusters based on their distances to cluster centroids kept in the chromosome.

Due to the structure of this representation, CCBFS algorithm is able to work in the instances with high number of data points. Since assignment can be done in consideration of selected features and cluster centroids, it is not added into exploration process. Therefore, solution space is squeezed significantly which can be represented as follows for the proposed memetic algorithm :

$$\left[\binom{m}{q}\right]^p \binom{n}{p} \tag{4.3}$$

In Equation 4.3, q features in M are selected for each cluster separately in the first term. In the second term, p centroids are selected from N. In Table 4.1, combinatorial problem sizes in the original problem and CCBFS algorithm are listed for some variations of p,n,m and q.

Table 4.1: Solution Space on Various Values of p, n, m and q

p	n	m	q	# of Solutions in the Original Problem	# of Solutions in CCBFS Algorithm
2	80	5	2	$6.04 * 10^{25}$	$3.16 * 10^{5}$
2	80	10	4	$2.67 * 10^{28}$	$1.39 * 10^{8}$
4	80	10	4	$1.18 * 10^{56}$	$3.08 * 10^{15}$
4	200	10	4	$2.09 * 10^{128}$	$1.26 * 10^{17}$

4.2.2 Initialization

Initialization step has a crucial importance in evolutionary algorithms, since it creates a basis for the whole generations during the algorithm. Therefore, we create initial population from three different aspects in order to attain an extensive foundation for the following generation. The process of initial population creation is explained in Algorithm 2 .

In the initialization type 1 ($InitType_1$), p different random data points are selected from the data set for each cluster in order to create cluster centroids. Hence, component 2 in the chromosome representation is satisfied by assigning centroids. After that, since the number of features to be used, q, is fixed, all attributes of these data points are not utilized and for each data point assigned as cluster centroid, q features are selected randomly. The random feature selection satisfies the component 1 on a chromosome (lines 3-7).

In the type 2 ($InitType_2$), we randomly select the features and assignment of the data points. As a first step, feature selection is done as explained in type 1. In the second step, each data point is assigned to one of the p clusters randomly.

In order to represent an individual, cluster centroids should be known. Therefore, all data points in a cluster are considered as cluster centroid and within-cluster distance values are calculated in accordance with randomly selected features. After that, the data points with smallest within-cluster distance values are assigned as cluster centroids for each cluster (lines 11-15).

For the initialization type 3, we first determine the assignment of data points via

random selection. Then, one data point randomly selected as cluster centroid for each cluster. However, we also need to determine selected features for each cluster to represent individuals' genotypes. Thus, we calculate the distance contribution of each feature on this cluster and select the q of them which has minimum addition on within-cluster distance values (lines 19-23).

As a last step, we control whether generated offspring is feasible or unique. If it is not, we discard the generated offspring since it is not add a value to the initial generation. Then, we start to create a new offspring from the beginning until a convenient individual is obtained.

4.2.3 Uniform Crossover Operator

Uniform crossover is an unbiased operator in terms of ordering of the chromosomes. Since it increases the exploration capabilities of the algorithm, uniform crossover is selected for crossover operations. We develop a crossover operator that works on two parents in the population, and the operator is applied to each component of the chromosomes by utilizing a crossover mask. The operator is applied to each component of the chromosomes distinctively. Therefore, it has two parts. In the first part, first components of the parents, feature component, are crossed. Second part crosses the cluster centroid indices. Each part yields two offsprings, and we obtain four offsprings in total out of two parents. Pseudo-code for the operation is provided in Algorithm 3.

For the first component, the crossover operator should separate the features of centers, i.e. it splits the first component into p. Then the parents exchange the feature sets of clusters according to a crossover mask, which is generated randomly. It is important to note that the crossover operator should divide the chromosome in between two cluster's tangent point in order to create feasible offsprings. If is is not divided properly, selection of q features in each cluster cannot be preserved. Figure 4.4 and 4.5 shows the undesirable and desirable splits of crossover operator for p equals to 2. Figure 4.4 destroys the chromosome structure because first cluster center is disrupted. However, in Figure 4.5 the cluster center structure is preserved.

Algorithm 2: Creation of the Initial Population

Input : Data set, q, p, Intervals for the initialization types

Output: Initial Population

```
k = 1
while k ≤ n do
    if k ∈ InitType_1 then
        for i=1,...,p do
            Select a centroid randomly
            Select q features randomly
        end for
    else if k ∈ InitType_2 then
        Assign each data point to one of the p clusters randomly
        for i=1,...,p do
            Select q features randomly
            Select the fittest centroid for cluster i with respect to assignments
              and selected features
        end for
    else
        Assign each data point to one of the p clusters randomly
        for i=1,...,p do
            Select a centroid among assigned data points randomly
            Select the fittest features for cluster i with respect to assignments
              and selected centroid
        end for
    end if
    if Created offspring is unique & feasible then
        k = k + 1
    end if
end while
```

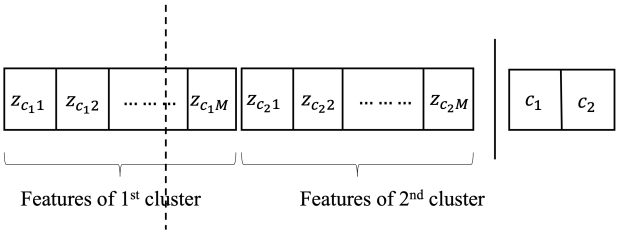


Figure 4.4: Undesirable chromosome split

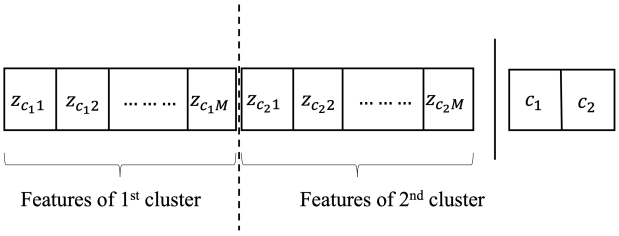


Figure 4.5: Desirable chromosome split

The creation structure of the first two offsprings is illustrated in the Figure 4.6. Here, two parents are selected for crossover as an initial step. Then, a crossover mask which is a $(1 * (m * p))$ boolean vector is created for each feature set of the chromosomes' first component, and feature sets of each cluster are crossed accordingly (lines 2-11). While creating the first two offsprings, cluster centers are directly transferred from their parents (lines 12-13).

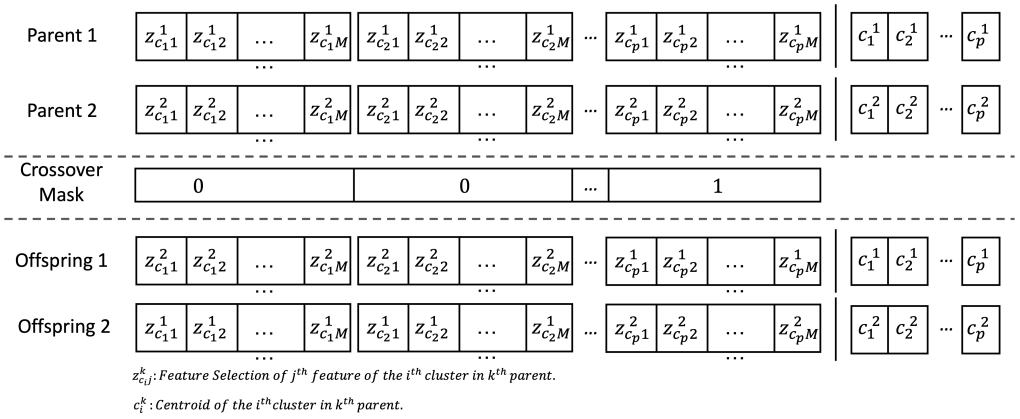


Figure 4.6: Uniform Crossover Operator to Create First Two Offsprings

Similar concept is utilized to create third and fourth offsprings which is shown in Figure 4.7. A crossover mask which is a $(1 * p)$ boolean vector is created for the

centroids. Offsprings three and four are generated from the crossover from second components of their parents (lines 14-25).

In both of the crossover operation, if randomly chosen value in the crossover mask is 1, then parent 1's genotype is transferred to 1^{st}/3^{rd} offspring. Similarly parent 2's genotype is transferred to offspring 2/4. On the other hand, if randomly chosen value is 0, then corresponding genotype of parent 1/3 is transferred to offspring 2/4.

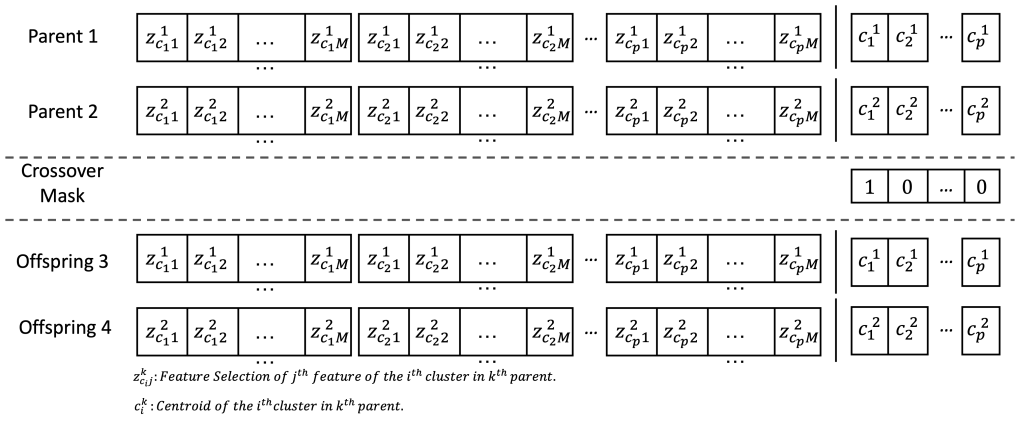


Figure 4.7: Uniform Crossover Operator to Create Last Two Offsprings

4.2.4 Mutation

Another operator to be designed in the genetic algorithm is mutation operator. With a predefined probability ($prob_{mut}$), the parents in the population are mutated. The mutation operator to be used in the proposed algorithm is called random mutation.

We utilize three different mutation operators in order to have stronger exploration capability. If an offspring is decided to be mutated, one of the following mutation types is applied to this offspring. A pseudo-code for this operation is provided in Algorithm 4.

4.2.4.1 Mutation - Type 1

The aim of this mutation operator is randomly searching for a different feature selection in one cluster. In this mutation type, a selected feature is turned into an unselected

Algorithm 3: Uniform Crossover Operation

Input : Current population

Output: Offsprings

for $k = 1, ..., pop_{size}/2$ **do**
  Create a $(1 * q)$ vector of boolean, $mask_1$
  for *a=1,...,q* **do**
    if $mask_1(a) = 1$ **then**
      Offspring1_feature(a)=Parent1_feature(a)
      Offspring2_feature(a)=Parent2_feature(a)
    else
      Offspring1_feature(a)=Parent2_feature(a)
      Offspring2_feature(a)=Parent1_feature(a)
    end if
  end for
  Offspring1_centroids = Parent1_centroids
  Offspring2_centroids = Parent2_centroids
  Create a $(1 * p)$ vector of boolean, $mask_2$
  for *b=1,...,p* **do**
    if $mask_2(b) = 1$ **then**
      Offspring3_centroid(b)=Parent1_centroid(b)
      Offspring4_centroid(b)=Parent2_centroid(b)
    else
      Offspring3_centroid(b)=Parent2_centroid(b)
      Offspring4_centroid(b)=Parent1_centroid(b)
    end if
  end for
  Offspring3_features = Parent1_features
  Offspring4_features = Parent2_features
end for

feature. Initially, one cluster is selected randomly among all. Subsequently, an unassigned feature is selected and a selected feature is cancelled at the same time (lines 2-5).

To illustrate, an example is given in Figure 4.8. In the example, the mutation operator is applied to the clustering problem where the number of clusters p, total number of features m, and total number of selected features q are equal to 3, 4, and 2, respectively. Red color represents the features selected, and blue color shows the unselected features. In this example, the center of third cluster is randomly selected. Before mutation, third cluster center uses $1^{st}$and 4^{th} attributes, and after the mutation operator, 4^{th} feature becomes unused, and 2^{nd} feature is selected to be used.

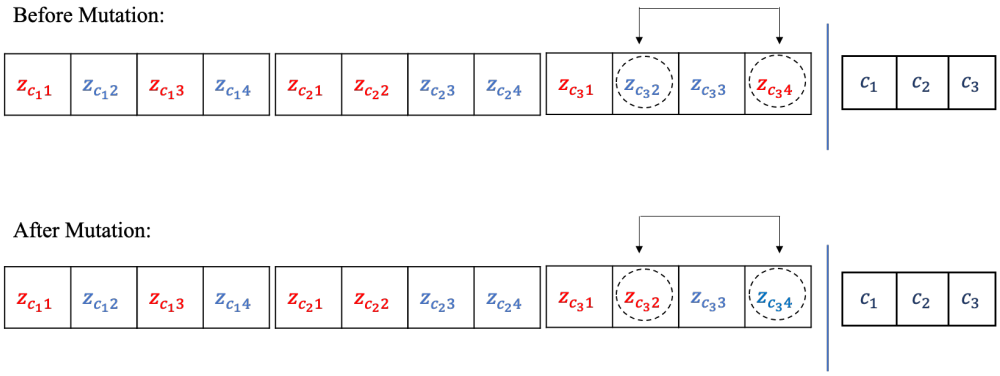


Figure 4.8: An example for Mutation - Type 1

4.2.4.2 Mutation - Type 2

The aim of the second mutation operator is also randomly searching for a different feature selection in one cluster. However, in this mutation type, all of the features of one cluster is selected randomly. In this mutation type, one cluster is selected randomly as in Type 1. After that, all of the selected features are cancelled and q of the features are selected randomly for this cluster (lines 6-8).

To illustrate, an example is given in Figure 4.9.

In the example, same structure of the example in Mutation - Type 1 is utilized. Here, 2^{nd} cluster is randomly selected for mutation. In this cluster 1^{st} and 2^{nd} features are selected before the mutation operation. After the mutation operator, both of features

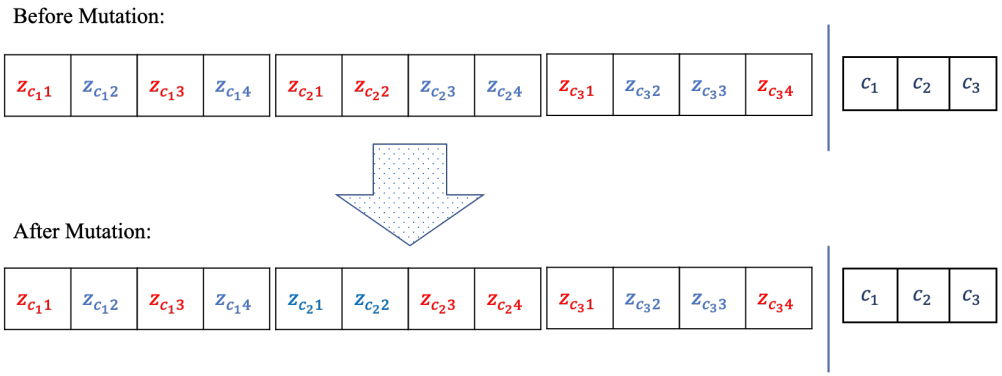


Figure 4.9: An example for Mutation - Type 2

becomes unused randomly, and 3^{rd} and 4^{th} features are selected to be used in this cluster.

4.2.4.3 Mutation - Type 3

The aim of the final mutation operator is searching possible feature selections in all of the clusters. This type is the most disruptive one among all mutation types, since all of the clusters affected in this mutation operator. In this mutation type, one selected and one unselected feature is picked randomly for all clusters. After that, their status is replaced in each cluster (lines 9-13).

To illustrate, an example is given in Figure 4.10.

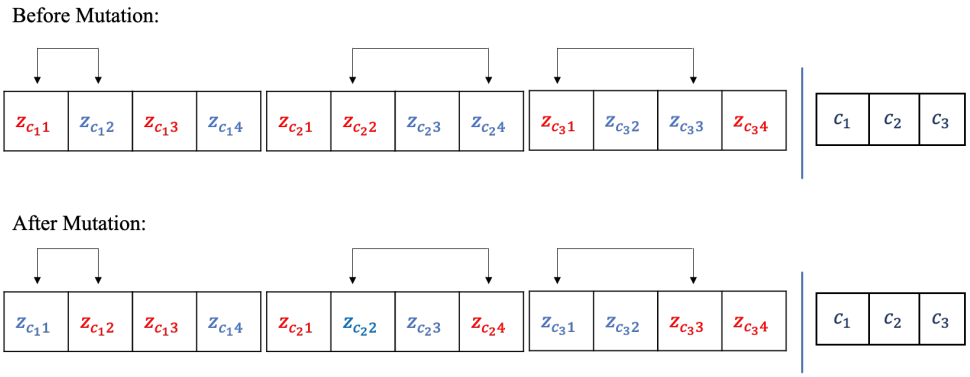


Figure 4.10: An example for Mutation - Type 3

In this example, same structure of the example in Mutation - Type 1 is utilized. Here,

1^{st} and 2^{nd}, $2^{nd}$and 4^{th}, 1^{st} and 3^{rd} features are selected for clusters 1, 2, and 3, respectively. After the mutation operator, status of pre-selected features are switched to unselected, and status of pre-unselected features are switched to selected.

Algorithm 4: Mutation Operation

Input : Offsprings

Output: Population to be Transferred

```
r=rand[0,1]
if r ≤ 1/3 * prob_mut then
    Select a cluster randomly
    Select one assigned and one assigned feature randomly
    Switch the assignment of selected features
else if r ≥ 1/3 * prob_mut & r ≤ 2/3 * prob_mut then
    Select a cluster randomly
    Switch the current selection with randomly selected q features
else
    for i=1,..,p do
        Select one assigned and one assigned feature randomly from i^th
         cluster
        Switch the assignment of selected features
    end for
end if
```

4.2.5 Fitness Function

After the mutation operator, assignment of data points to the clusters is done and fitness values for both parents and offsprings are calculated. We have selected the objective function of the clustering problem as the fitness function, which is the total sum of the rectilinear distances between data points and their cluster centers. In order to calculate the fitness of an offspring, each data point is assigned to its nearest cluster centroid as an initial step. Subsequently, within-cluster distance values for each cluster is calculated. Finally, sum of distance values of all p clusters is obtained and this value is assigned as the fitness value of the corresponding offspring. Pseudo-

code for fitness calculation is given in Algorithm 5.

Algorithm 5: Fitness Calculation

Input : Data set, Offspring's Chromosome

Output: Fitness Value, Cluster Assignment

for *i=1,...,n* **do**
    Calculate $L_1 - Norm$ distances of i^{th} data point to all centroids
    Assign each data point to its nearest cluster
end for
for *i=1,...,p* **do**
    Calculate within-cluster $L_1 - Norm$ distance of i^{th} cluster with respect to the assignments
end for
Calculate the sum of within-cluster distances as fitness value

4.2.6 Neighborhood Search

We obtain new cluster centroids through crossover operations. However, new offsprings are totally depend on their parents, and there might be better centroid selection in the neighborhood of the selected centroids. Therefore, we add a neighborhood search step on our CCBFS algorithm to increase its exploitation capabilities. The purpose of the neighborhood search is testing the potential centroids that are close to the current ones which might provide fitter cluster centroids. In this method, neighborhood is defined on the distances among data points. For instance, the data point that has lowest distance to its centroid considered as the closest neighbor in this cluster.

Search algorithm starts with assignment of all data points to their centroids. Then, pre-determined number of neighbors of these centroids (s) with respect to selected features are obtained. Subsequently, obtained data points are evaluated as if they are centroid separately, and a fitness value is calculated. Therefore, we test ($s*p$) possible solutions in this step.

Unless an alternative centroid selection in the neighborhood is found, neighborhood search step is terminated. However, in case a fitter offspring found, then the search

process is done all over again for $(s * p)$ neighbors. Note that, centroids of $(p - 1)$ clusters have not changed, but we still look their neighbors in the second step, because there still might be fitter centroid selection in the updated structure.

After the second step, neighborhood search is completed. If a fitter individual is found in the neighborhood search, it is attached to the population as a new member, otherwise, nothing is changed in the current population. In order to maintain evolutionary structure of the algorithm and not to increase computational time, we only apply neighborhood search to the fittest individual in the population. In Algorithm 6, pseudo-code of the proposed neighborhood search is provided.

Effects of the neighborhood search on the CCBFS algorithm is tested in this study. Moreover, a comparison of the two step and recursive neighborhood search is examined. Experimental results of these tests are presented in Section 5.4.

4.2.7 Selection of the Next Generation

Next generation is created based on the fitness values of parents and offsprings. Before creation, feasibility of all offsprings is checked and only feasible offsprings are taken into consideration to transfer.

Selection process starts with sorting the individuals in the current population. Parents and generated offsprings are sorted in descending order in terms of their fitness values. After that, first 5% of the population size are transferred directly to the next generation as parents. Remaining 95% of the next generation is chosen from the remaining feasible individuals in the current generation randomly. Purpose of 5%- 95% is to maintain both exploitation and exploration capabilities of the CCBFS Algorithm.

The aim of this selection rule is to improve poor parents and avoid premature convergence. Finally, properties of the best solution are saved into the best solution matrix. The pseudo-code is reported in Algorithm 7.

Algorithm 6: Neighborhood Search 1

Input : Chromosome of the Fittest Offspring

Output: Chromosome of the Fittest Offspring

Assign fitness value of the fittest offspring as F
Step = 1
for *i=1,...,p* **do**
    for *j=1,...,s* **do**
        Select the j^{th} neighbor of i^{th} cluster's centroid
        Assign the j^{th} neighbor as centroid,
        Calculate fitness value and denote as F_i^j
    end for
end for
if $min(F_i^j) < F$ **then**
    Switch i^{th} cluster's centroid with its j^{th} neighbor which gives minimum distance value
    if $Step == 1$ **then**
        Step = 2
        Go back to 3^{rd} line
    end if
end if

Algorithm 7: Selection of the Next Generation

Input : Current Population

Output: Population to be Transferred

Sort Individuals in the population based on fitness values
Select Population Size * 5% of the fittest offsprings
Select Population Size * 95% of the offsprings from the remaining feasible offsprings randomly

CHAPTER 5

COMPUTATIONAL RESULTS OF CCBFS

In this Chapter, computational results of the CCBFS algorithm is explained. Generation of the simulated sets that are utilized in this study is explained in 5.1. In Section 5.2, utilized performance measures for evaluation are provided. Subsequently, parameter settings of the algorithm is explained in 5.3. Then, computational results from the simulated data sets are presented in Section 5.4. Lastly, computational results on a benchmark study is reported in Section 5.5.

5.1 Simulated Data Sets

In this study, we utilize simulated data sets in order to examine the performances of the developed algorithm. Therefore, we generate data sets which include relevant and redundant features. Moreover, we create data sets in different sizes to test the CCBFS algorithm in different instances. Details of the data sets are summarized in Table 5.1. As shown in the Table 5.1, a data set includes n data points and m total number of features. Among m features, q of them represents related features while the remaining $(m - q)$ of them represents the irrelevant ones. Mainly, simulations are done by changing; number of clusters, number of data points, total number of features, and number of relevant features. The tests include five types of data point sets (80, 100, 200, 500 and 1000), five types of features (5, 6, 8, 10, 12), and three types of relevant features (2, 3, 4) for each cluster. Therefore, all tests are done for 75 instances for each value of p.

In order to create a problem instance, we first create an $(n * m)$ data set filled with redundant features. For each column, one redundant feature is generated from uniform

Table 5.1: Details of the Simulated Data Sets

Number of Clusters (p)	{2, 3, 4}
Number of Data Points (n)	{80, 100, 200, 500, 1000}
Number of Features (m)	{5, 6, 8, 10, 12}
Number of Relevant Features (q)	{2, 3, 4}

distribution and the entire column is filled with selected distribution. The distribution parameters for redundant features are listed in Table 5.2. It should be noted that selection of the uniform distribution is equally likely for each column.

Table 5.2: Distributions of Redundant Features

Number of Clusters (p)	Redundant Parameter Distribution
2	U(0, 20), U(0, 10) and U(0, 5)
3	U(0, 20), U(0, 10), U(0, 5) and U(-10, 0)
4	U(0, 20), U(0, 10), U(0, 5) and U(-10, 0)

After that, q of the m features for each cluster are selected for relevant features. For these features, we generate numbers from normal distribution in order to obtain denser values. To differentiate the clusters, they have different mean values, while standard deviation of the distribution is identical for all. Distribution of the relevant features for each cluster is shown in Table 5.3. While assigning the relevant features, it is assumed that each cluster has equal number of points, which is (N/p).

Table 5.3: Distributions of Redundant Features

Number of Clusters (p)	Relevant Parameter Distribution
2	N($\mu = 0, \sigma = 1$) and N($\mu = 5, \sigma = 1$)
3	N(0, 1), N(5, 1) and N(-7, 1)
4	N(0, 1), N(5, 1), N(-7, 1) and N(11, 1)

As a final step, each column is standardized separately. Both relevant and redundant

features are created via different distributions, and it causes a scale effect among the columns of all data instances. Therefore, we rescale each column one by one to an interval of $[0, 1]$.

5.2 Performance Measures

The proposed evolutionary algorithm is assessed with respect to various performance measures. Initially, performance measures according to within-cluster distance values (objective function) are reported in this section, which are percentage gaps, number of best and worst solutions and computational times. Subsequently, the clustering performance of the algorithm is tested by its capability on finding the pre-selected features. Utilized performance measures are summarized in the following sections.

5.2.1 Performance Measures on Objective Function

Percent gap of the best solution from ground truth objective (*Best vs.* Z^c):

After creation of a problem instance, we calculate a ground truth objective (denoted as Z^c) by selecting centroids for each cluster, and Z^c value represents the objective function of a created data instance. Therefore, we compare the best result of the proposed algorithm and the Z^c in all of the instances. We denote the within-cluster distance of the best solution as Z_{Best}, and the percentage gap of the best solution from ground truth objective is calculated as follows:

$$Best\ vs.\ Z^c = \left(\frac{Z_{Best} - Z^c}{Z^c} \right) * 100$$

It should be noted that Best vs. Z^c can take negative values. This means that CCBFS algorithm's best result finds denser clusters in terms of within-cluster distance value.

Percent gap between best and worst solution (*Worst vs. Best)*:

Although heuristic algorithms are beneficial to find feasible solutions for difficult problems, they may also provide objectionable results in some instances. Therefore,

we also compare the performances of the best and worst results of our tests. For this metric, the algorithm is run for 50 times for each problem instance and the best and worst results are compared according to following formula:

$$Worst\ vs.\ Best = \left(\frac{Z_{Worst} - Z_{Best}}{Z_{Best}} \right) * 100$$

Number of best solutions (*# of Best*):

We test the CCBFS algorithm 50 times for all of the problem instances. This measure reports us the replication of the best results. If the best result gap between Z^c value is non-positive, and the value for number of best solutions is high, it means that CCBFS algorithm provides relatively good results in most of the times.

Number of worst solutions (*# of Worst*):

This measure controls the replication of worst results which is quite similar with the previous one. However, the idea behind this metric is to test exploration capabilities of the CCBFS algorithm. If the worst result gap between best result value is high, and the value for number of worst solutions is high, it means that CCBFS algorithm is stuck in a local optima and is not able to explore better clusters in these instances.

Computational (CPU) Time:

CPU Time reports the average solution time of the proposed algorithm over 50 tests in the basis of seconds.

5.2.2 Performance Measures on Clustering

Correct selection of pre-selected features:

Since relevant and redundant features of each cluster are generated with different distributions, feature selection results of the proposed algorithm can be compared with the pre-selected ones.

For this measure, first we obtain the selected features of each cluster from the best solution among 50 trials. Then, compare the pre-selected features with selected ones

for each cluster. Since q features are selected in each cluster, there can be at most $p*q$ matches.

It should be noted that, correct clustering is a pre-requisite for this measure. If the clustering is not successful, correct selection of features would not be meaningful.

5.3 Parameter Setting for CCBFS Algorithm

Parameters of data sets are introduced in the Table 5.1. Here, number of clusters, data points and relevant features are listed. So we are able to test proposed algorithm in 225 instances total.

Table 5.4: Parameter Setting of the CCBFS Algorithm

Population Size	1000
Number of Generations	200
Ratio of Initial Generation by Type 1, 2 and 3	{50%, 25% , 25%}
Crossover Probability	100%
Mutation Probability by Type 1, 2 and 3	{1%, 1%, 1%}
Number of Candidate Centroids for Neighborhood Search	10

Furthermore, parameter setting related with the evolutionary algorithm is given in the Table 5.4. Population size is set at 1000 for all of the instances as it provides significant exploration capabilities to the algorithm. To determine number of generations in the algorithm, we have observed that the best results generally obtained after 100^{th} generation. That is why, this value is set as 200.

Another decision is the distribution of the initial generation with respect to initialization types. We tested seven possible distribution here which are, (i) all of the first generation is taken from either Type 1, Type 2 or Type 3. (ii) one of the types takes 50% of the initial population and others take 25% of it. (iii) Lastly, all initialization types create equal amount of individuals in the initial population. Then, we select the setting that 50% of the initial population is created by initialization type 1, since this setting obtains better results among others in harder problems while they have similar

Table 5.5: Comparison for Ratio of Initial Generation by Type 1, 2 and 3

Ratio (%)	***(50 - 25 - 25)***	*(25 - 50 - 25)*	*(25-25-50)*	*(33 - 33 - 33)*	*(100 - 0 - 0)*	*(0 - 100 - 0)*	*(0 - 0 - 100)*
Best vs. Z^c	**-1.36%**	-1.36%	-1.36%	-1.36%	-1.36%	-1.36%	-1.36%
Worst vs. Z^c	**0.57%**	0.85%	1.10%	0.78%	0.97%	0.78%	0.79%
# of Bests	**26.68**	22.88	23.64	22.48	23.12	23.6	24.64

results in smaller problem instances.

Since clustering is harder for higher values of p and q, test are done for varying values of n and m while $p = 4$, $q = 4$. In table 5.5, results of the tests regarding initialization are summarized. In terms of best solution, any difference is not observed, but proposed ratios provides denser clustering in the worst solutions. Furthermore, main difference among the tests is observed on number of best solutions. The setting with proposed initialization ratios clearly obtains the most number of best solutions.

We take crossover probability as 100% and give 1% probability for each type of mutation. It should be noted that one individual can be mutated at most once in a generation. Lastly, nearest 10 possible centroids for each cluster are investigated during neighborhood search operation.

5.4 Computational Results for Simulated Data Sets

The CCBFS algorithm is coded in MATLAB R2021a and simulations to test the algorithm has been carried out on 64-bit Windows 10 PC with 3.6 GHz 12 core Intel Xeon E-2246G processor and 16 GB RAM.

Initially, we test the effects of neighborhood search on CCBFS algorithm as mentioned in Chapter 4.2.6. Since data sets with higher number of clusters are harder to solve, neighborhood search is tested when $q = 4$. In terms of best solutions, the tests with no neighborhood search are always higher than or equal to CCBFS algorithm. Higher differences has observed for the worst solutions. Furthermore, main difference for neighborhood search is observed on number of best solutions. Since it increases the exploitation capacity, algorithm with no local search is not able to find best results repeatedly. Detailed results of the experiments are presented in Table B.1.

Furthermore, we also test the number of steps in neighborhood search. We have compared the results of the proposed method with a recursive search method. In terms of best results, searching neighborhood recursively and in two steps obtain the same results in all data instances. Moreover, there is no significant difference between these two methods in terms of worst results. The only difference between two methods is the CPU time in which recursive version always obtains results in longer durations. Therefore, we have utilized two step neighborhood search in our computational tests.

In order to test the effects of memetic algorithm's abilities on finding better solutions, results of the CCBFS algorithm are compared with the best results from the initialization step. When we analyze the development through generations, a continuous decrease on objective function is observed. Although the generation which obtains the best result is highly variable for each data instance, taking the number of generation parameter as 200 is more than enough. Development of the fitness value with respect to the generation is illustrated for 4 data instances in Figure B.1.

To evaluate performance of the CCBFS algorithm, it is tested on the data sets that are described in Section 5.1. For the simulations, parameter settings explained in Section 5.3 is utilized. Detailed results for each value of p is provided in the Tables 5.6 - 5.8. In the result tables, selected p, n, m and q values in that simulation, ground truth objective, best, median and worst results among 50 trials, percentage gaps between best result vs. Z^c and worst result, number of best and worst solutions, CPU times and correctly selected features of the best result are provided. In the last column, feature selection of the CCBFS is compared with the created data and the common features for each cluster is reported. Therefore, values on this column is always less than or equal to the value of q.

Table 5.6: Simulation Results of CCBFS Algorithm for $p = 2$

p,n,m,q	Z^c	**Best**	**Median**	**Worst**	**Best vs.** Z^c	**Worst vs. Best**	**# of Best**	**# of Worst**	**Correctly Selected Features**	**CPU (s)**
2,80,5,2	10.06	10.06	10.06	10.06	0.0%	0.0%	50	50	[2,2]	18.82
2,80,6,2	12.36	12.33	12.33	12.33	-0.2%	0.0%	50	50	[2,2]	23.94
2,80,8,2	12.87	12.67	12.67	12.67	-1.6%	0.0%	50	50	[2,2]	17.82
2,80,10,2	13.90	13.90	13.90	13.90	0.0%	0.0%	50	50	[2,2]	18.46
2,80,12,2	10.58	10.37	10.37	10.37	-2.0%	0.0%	50	50	[2,2]	19.07

Table 5.6 continued from previous page

p,n,m,q	Z^c	Best	Median	Worst	Best vs. Z^c	Worst vs. Best	# of Best	# of Worst	Correctly Selected Features	CPU (s)
2,80,5,3	19.86	19.86	19.86	19.86	0.0%	0.0%	50	50	[3,3]	18.96
2,80,6,3	14.79	14.79	14.79	14.79	0.0%	0.0%	50	50	[3,3]	24.03
2,80,8,3	19.97	19.97	19.97	19.97	0.0%	0.0%	50	50	[3,3]	17.62
2,80,10,3	16.14	16.14	16.14	16.14	0.0%	0.0%	50	50	[3,3]	18.69
2,80,12,3	17.79	17.79	17.79	17.79	0.0%	0.0%	50	50	[3,3]	18.97
2,80,5,4	25.98	25.98	25.98	25.98	0.0%	0.0%	50	50	[4,4]	18.75
2,80,6,4	24.80	24.80	24.80	24.80	0.0%	0.0%	50	50	[4,4]	24.38
2,80,8,4	27.60	27.60	27.60	27.60	0.0%	0.0%	50	50	[4,4]	18.11
2,80,10,4	24.68	24.68	24.68	24.68	0.0%	0.0%	50	50	[4,4]	18.17
2,80,12,4	29.30	29.30	29.30	29.30	0.0%	0.0%	50	50	[4,4]	18.67
2,100,5,2	11.48	11.47	11.47	11.47	-0.1%	0.0%	50	50	[2,2]	16.90
2,100,6,2	13.54	13.54	13.54	13.54	0.0%	0.0%	50	50	[2,2]	17.12
2,100,8,2	12.72	12.22	12.22	12.24	-3.9%	0.1%	45	5	[2,2]	18.08
2,100,10,2	9.31	9.30	9.30	9.30	0.0%	0.0%	50	50	[2,2]	18.70
2,100,12,2	13.56	13.56	13.56	13.56	0.0%	0.0%	50	50	[2,2]	19.21
2,100,5,3	18.12	18.12	18.12	18.12	0.0%	0.0%	50	50	[3,3]	16.92
2,100,6,3	25.00	25.00	25.00	25.00	0.0%	0.0%	50	50	[3,3]	17.50
2,100,8,3	23.02	22.97	22.97	22.97	-0.2%	0.0%	50	50	[3,3]	18.69
2,100,10,3	28.10	28.10	28.10	28.10	0.0%	0.0%	50	50	[3,3]	19.34
2,100,12,3	22.32	22.32	22.32	22.81	0.0%	2.2%	48	2	[3,3]	20.13
2,100,5,4	33.79	33.79	33.79	33.79	0.0%	0.0%	50	50	[4,4]	16.20
2,100,6,4	34.69	34.69	34.69	34.69	0.0%	0.0%	50	50	[4,4]	17.44
2,100,8,4	34.01	34.01	34.01	34.01	0.0%	0.0%	50	50	[4,4]	18.31
2,100,10,4	30.97	30.97	30.97	30.97	0.0%	0.0%	50	50	[4,4]	19.11
2,100,12,4	29.20	29.20	29.20	29.20	0.0%	0.0%	50	50	[4,4]	19.88
2,200,5,2	31.50	31.37	31.37	31.37	-0.4%	0.0%	50	50	[2,2]	17.96
2,200,6,2	28.88	28.65	28.65	28.65	-0.8%	0.0%	50	50	[2,2]	18.88
2,200,8,2	33.90	33.78	33.78	33.78	-0.4%	0.0%	50	50	[2,2]	19.73
2,200,10,2	38.50	38.49	38.49	38.49	0.0%	0.0%	50	50	[2,2]	20.92
2,200,12,2	28.82	28.82	28.82	28.82	0.0%	0.0%	50	50	[2,2]	22.70
2,200,5,3	51.07	51.07	51.07	51.07	0.0%	0.0%	50	50	[3,3]	18.69
2,200,6,3	56.55	56.55	56.55	56.79	0.0%	0.4%	37	13	[3,3]	19.42
2,200,8,3	40.49	40.49	40.49	40.55	0.0%	0.1%	42	8	[3,3]	20.39
2,200,10,3	43.28	43.28	43.28	43.28	0.0%	0.0%	50	50	[3,3]	21.33
2,200,12,3	47.98	46.86	46.86	46.86	-2.3%	0.0%	50	50	[3,3]	23.08
2,200,5,4	63.67	63.67	63.67	63.67	0.0%	0.0%	50	50	[4,4]	17.71
2,200,6,4	68.43	68.43	68.43	68.43	0.0%	0.0%	50	50	[4,4]	18.83
2,200,8,4	60.02	60.02	60.02	60.02	0.0%	0.0%	50	50	[4,4]	20.12
2,200,10,4	68.99	68.99	68.99	69.23	0.0%	0.4%	43	7	[4,4]	20.99
2,200,12,4	52.27	52.27	52.27	52.84	0.0%	1.1%	45	5	[4,4]	22.64
2,500,5,2	55.71	55.71	55.71	55.71	0.0%	0.0%	50	50	[2,2]	22.85
2,500,6,2	79.17	78.98	78.98	79.00	-0.2%	0.0%	44	6	[2,2]	24.19
2,500,8,2	63.96	63.96	63.96	63.96	0.0%	0.0%	50	50	[2,2]	25.88

Table 5.6 continued from previous page

p,n,m,q	Z^c	Best	Median	Worst	Best vs. Z^c	Worst vs. Best	# of Best	# of Worst	Correctly Selected Features	CPU (s)
2,500,10,2	68.89	68.59	68.59	68.59	-0.4%	0.0%	50	50	[2,2]	28.42
2,500,12,2	68.80	68.69	68.69	68.69	-0.2%	0.0%	50	50	[2,2]	30.08
2,500,5,3	123.03	122.74	122.74	122.74	-0.2%	0.0%	50	50	[3,3]	22.48
2,500,6,3	115.89	115.89	115.89	117.34	0.0%	1.3%	37	13	[3,3]	24.48
2,500,8,3	109.32	109.31	109.31	111.64	0.0%	2.1%	41	1	[3,3]	26.99
2,500,10,3	101.36	101.35	101.35	101.39	0.0%	0.0%	34	16	[3,3]	28.28
2,500,12,3	97.62	97.62	97.62	98.60	0.0%	1.0%	49	1	[3,3]	29.50
2,500,5,4	146.84	146.84	147.19	148.35	0.0%	1.0%	23	2	[4,4]	22.14
2,500,6,4	155.22	155.22	155.22	158.89	0.0%	2.4%	49	1	[4,4]	23.72
2,500,8,4	148.85	148.85	148.85	150.35	0.0%	1.0%	46	4	[4,4]	25.65
2,500,10,4	162.07	162.07	162.07	162.07	0.0%	0.0%	50	50	[4,4]	28.51
2,500,12,4	136.30	136.21	136.21	138.17	-0.1%	1.4%	30	1	[4,4]	29.69
2,1000,5,2	143.87	143.87	143.87	143.87	0.0%	0.0%	50	50	[2,2]	30.96
2,1000,6,2	119.75	119.61	119.61	119.61	-0.1%	0.0%	50	50	[2,2]	32.60
2,1000,8,2	144.78	144.31	144.31	144.31	-0.3%	0.0%	50	50	[2,2]	36.15
2,1000,10,2	169.73	169.62	169.62	170.96	-0.1%	0.8%	49	1	[2,2]	55.18
2,1000,12,2	155.03	149.15	149.15	149.15	-3.8%	0.0%	50	50	[2,2]	59.28
2,1000,5,3	192.25	192.15	192.15	192.15	-0.1%	0.0%	50	50	[3,3]	30.48
2,1000,6,3	232.70	232.70	232.70	232.70	0.0%	0.0%	50	50	[3,3]	31.66
2,1000,8,3	223.45	223.45	223.45	223.45	0.0%	0.0%	50	50	[3,3]	35.27
2,1000,10,3	144.62	144.62	144.62	144.62	0.0%	0.0%	50	50	[3,3]	54.97
2,1000,12,3	220.51	220.51	220.51	220.51	0.0%	0.0%	50	50	[3,3]	58.31
2,1000,5,4	320.58	320.58	320.58	324.31	0.0%	1.2%	45	5	[4,4]	29.22
2,1000,6,4	309.87	309.87	311.84	318.51	0.0%	2.8%	23	1	[4,4]	31.57
2,1000,8,4	263.76	263.76	263.76	264.51	0.0%	0.3%	38	12	[4,4]	34.82
2,1000,10,4	309.68	309.68	309.68	312.18	0.0%	0.8%	33	4	[4,4]	54.74
2,1000,12,4	258.61	258.61	258.61	258.61	0.0%	0.0%	50	50	[4,4]	57.76

Table 5.7: Simulation Results of CCBFS Algorithm for $p = 3$

p,n,m,q	Z^c	Best	Median	Worst	Best vs. Z^c	Worst vs. Best	# of Best	# of Worst	Correctly Selected Features	CPU (s)
3,80,5,2	8.93	8.93	8.93	8.93	0.0%	0.0%	50	50	[2,2,2]	17.03
3,80,6,2	9.22	9.05	9.05	9.05	-1.8%	0.0%	50	50	[2,2,2]	18.48
3,80,8,2	9.09	8.99	8.99	8.99	-1.1%	0.0%	50	50	[2,2,2]	16.43
3,80,10,2	8.86	8.84	8.84	8.84	-0.2%	0.0%	50	50	[2,2,2]	17.42
3,80,12,2	8.42	8.42	8.42	8.42	0.0%	0.0%	50	50	[2,2,2]	18.04
3,80,5,3	11.36	11.36	11.36	11.36	0.0%	0.0%	50	50	[3,3,3]	18.17
3,80,6,3	15.76	15.66	15.66	15.77	-0.7%	0.7%	34	12	[2,3,3]	22.36
3,80,8,3	9.18	9.18	9.18	9.18	0.0%	0.0%	50	50	[3,3,3]	17.87
3,80,10,3	12.12	12.12	12.12	12.78	0.0%	5.4%	40	1	[3,3,3]	18.85

Table 5.7 continued from previous page

p,n,m,q	Z^c	Best	Median	Worst	Best vs. Z^c	Worst vs. Best	# of Best	# of Worst	Correctly Selected Features	CPU (s)
3,80,12,3	16.21	16.21	16.21	16.21	0.0%	0.0%	50	50	[3,3,3]	19.74
3,80,5,4	19.57	19.57	19.57	19.57	0.0%	0.0%	50	50	[4,4,4]	17.34
3,80,6,4	16.50	16.50	16.50	16.50	0.0%	0.0%	50	50	[4,4,4]	22.00
3,80,8,4	21.86	20.36	20.36	20.69	-6.8%	1.6%	13	1	[3,3,4]	17.83
3,80,10,4	17.83	17.83	17.83	18.24	0.0%	2.3%	49	1	[4,4,4]	18.74
3,80,12,4	19.60	18.53	18.53	19.31	-5.5%	4.2%	38	1	[3,3,4]	19.48
3,100,5,2	10.83	10.83	10.83	10.83	0.0%	0.0%	50	50	[2,2,2]	15.44
3,100,6,2	9.84	9.84	9.84	9.88	0.0%	0.4%	49	1	[2,2,2]	16.20
3,100,8,2	12.90	11.76	11.76	11.82	-8.9%	0.5%	12	5	[2,1,2]	16.77
3,100,10,2	10.56	10.52	10.52	10.52	-0.4%	0.0%	50	50	[2,2,2]	17.81
3,100,12,2	13.82	12.56	12.56	12.64	-9.1%	0.6%	34	5	[2,1,2]	18.58
3,100,5,3	12.91	12.91	12.91	12.91	0.0%	0.0%	50	50	[3,3,3]	16.75
3,100,6,3	16.09	16.09	16.09	16.13	0.0%	0.2%	22	3	[3,3,3]	17.31
3,100,8,3	17.64	17.64	17.64	17.89	0.0%	1.4%	31	5	[3,3,3]	18.48
3,100,10,3	20.17	19.43	19.43	20.43	-3.7%	5.2%	35	1	[3,2,3]	19.36
3,100,12,3	17.52	17.37	17.37	17.62	-0.9%	1.4%	49	1	[2,3,3]	20.24
3,100,5,4	22.45	22.45	22.45	22.56	0.0%	0.5%	29	1	[4,4,4]	16.02
3,100,6,4	21.80	21.80	21.80	21.80	0.0%	0.0%	50	50	[4,4,4]	17.07
3,100,8,4	22.38	21.43	21.43	21.43	-4.3%	0.0%	50	50	[3,4,4]	18.28
3,100,10,4	19.57	19.57	19.57	19.57	0.0%	0.0%	31	19	[4,4,4]	19.37
3,100,12,4	26.78	24.79	24.79	25.38	-7.4%	2.4%	49	1	[3,3,4]	20.18
3,200,5,2	20.31	20.31	20.31	20.31	0.0%	0.0%	50	50	[2,2,2]	16.79
3,200,6,2	20.36	16.82	16.82	16.82	-17.4%	0.0%	50	50	[1,2,2]	17.52
3,200,8,2	22.06	21.18	21.18	21.18	-4.0%	0.0%	50	50	[1,2,2]	18.73
3,200,10,2	23.46	23.03	23.03	23.03	-1.8%	0.0%	37	13	[2,2,2]	19.87
3,200,12,2	28.26	25.16	25.16	25.16	-11.0%	0.0%	50	50	[2,1,2]	22.16
3,200,5,3	39.84	38.98	38.98	39.84	-2.2%	2.2%	36	2	[3,2,3]	18.18
3,200,6,3	31.85	31.85	31.85	31.85	0.0%	0.0%	50	50	[3,3,3]	19.06
3,200,8,3	34.23	34.18	34.18	34.35	-0.2%	0.5%	42	3	[3,3,3]	20.44
3,200,10,3	29.63	28.39	28.39	29.96	-4.2%	5.5%	47	2	[2,3,3]	21.82
3,200,12,3	26.25	26.25	26.25	26.81	0.0%	2.2%	47	2	[3,3,3]	24.08
3,200,5,4	40.63	40.63	40.63	40.63	0.0%	0.0%	50	50	[4,4,4]	17.53
3,200,6,4	45.66	45.38	45.38	45.61	-0.6%	0.5%	41	9	[4,3,4]	18.87
3,200,8,4	51.96	47.90	47.90	47.90	-7.8%	0.0%	50	50	[3,3,4]	20.28
3,200,10,4	44.03	44.03	44.03	44.95	0.0%	2.1%	23	1	[4,4,4]	21.39
3,200,12,4	50.05	48.46	48.46	49.12	-3.2%	1.4%	32	1	[3,3,4]	22.16
3,500,5,2	59.72	59.18	59.18	59.74	-0.9%	1.0%	48	2	[2,2,2]	22.43
3,500,6,2	47.88	47.88	47.88	48.84	0.0%	2.0%	38	1	[2,2,2]	23.78
3,500,8,2	55.06	54.94	54.94	54.94	-0.2%	0.0%	50	50	[2,2,2]	26.11
3,500,10,2	51.94	51.40	51.40	51.40	-1.0%	0.0%	50	50	[2,2,2]	28.18
3,500,12,2	50.72	50.04	50.04	50.04	-1.3%	0.0%	50	50	[2,2,2]	30.26
3,500,5,3	85.19	81.19	81.46	82.04	-4.7%	1.0%	10	3	[3,2,3]	23.51
3,500,6,3	86.04	86.04	86.04	86.04	0.0%	0.0%	50	50	[3,3,3]	24.95

Table 5.7 continued from previous page

p,n,m,q	Z^c	Best	Median	Worst	Best vs. Z^c	Worst vs. Best	# of Best	# of Worst	Correctly Selected Features	CPU (s)
3,500,8,3	105.06	104.91	104.91	107.31	-0.1%	2.3%	47	3	[3,3,3]	28.17
3,500,10,3	90.75	90.64	90.90	92.33	-0.1%	1.9%	15	2	[2,3,3]	30.38
3,500,12,3	87.26	83.35	83.35	87.19	-4.5%	4.6%	47	3	[3,2,3]	32.67
3,500,5,4	103.12	103.12	103.12	105.41	0.0%	2.2%	39	1	[4,4,4]	20.88
3,500,6,4	94.40	94.40	94.40	97.47	0.0%	3.2%	36	2	[4,4,4]	22.91
3,500,8,4	96.99	96.99	96.99	96.99	0.0%	0.0%	50	50	[4,4,4]	25.37
3,500,10,4	102.08	102.08	102.08	105.84	0.0%	3.7%	20	1	[4,4,4]	27.89
3,500,12,4	111.89	107.38	107.38	107.44	-4.0%	0.1%	35	6	[4,3,4]	29.98
3,1000,5,2	79.60	79.60	79.60	80.05	0.0%	0.6%	45	5	[2,2,2]	30.39
3,1000,6,2	124.05	123.52	123.52	123.52	-0.4%	0.0%	50	50	[2,2,2]	32.51
3,1000,8,2	106.86	106.49	106.49	106.49	-0.3%	0.0%	50	50	[2,2,2]	36.30
3,1000,10,2	91.15	90.51	90.51	90.51	-0.7%	0.0%	50	50	[2,2,2]	54.40
3,1000,12,2	116.58	115.70	115.70	116.44	-0.8%	0.6%	46	4	[2,2,2]	58.48
3,1000,5,3	158.61	158.61	158.61	158.61	0.0%	0.0%	12	38	[3,3,3]	32.42
3,1000,6,3	144.34	137.16	137.16	137.59	-5.0%	0.3%	18	24	[2,3,3]	34.40
3,1000,8,3	136.41	136.40	136.40	141.30	0.0%	3.6%	15	1	[3,3,3]	38.80
3,1000,10,3	173.99	173.71	173.71	173.71	-0.2%	0.0%	50	50	[3,3,3]	59.62
3,1000,12,3	155.04	155.04	155.04	158.47	0.0%	2.2%	49	1	[3,3,3]	62.95
3,1000,5,4	176.32	176.32	176.32	179.74	0.0%	1.9%	47	3	[4,4,4]	27.93
3,1000,6,4	159.98	159.98	159.98	160.76	0.0%	0.5%	28	3	[4,4,4]	30.77
3,1000,8,4	204.78	204.78	204.78	207.40	0.0%	1.3%	30	2	[4,4,4]	35.96
3,1000,10,4	181.96	169.22	169.22	173.01	-7.0%	2.2%	23	2	[3,4,4]	54.58
3,1000,12,4	217.74	217.74	217.74	222.68	0.0%	2.3%	45	3	[4,4,4]	61.14

Table 5.8: Simulation Results of CCBFS Algorithm for $p = 4$

p,n,m,q	Z^c	Best	Median	Worst	Best vs. Z^c	Worst vs. Best	# of Best	# of Worst	Correctly Selected Features	CPU (s)
4,80,5,2	7.67	7.13	7.13	7.13	-7.0%	0.0%	50	50	[2,1,2,2]	21.55
4,80,6,2	8.96	8.15	8.15	8.35	-9.1%	2.5%	35	1	[1,2,2,2]	25.57
4,80,8,2	7.80	7.80	7.80	7.80	0.0%	0.0%	50	50	[2,2,2,2]	21.53
4,80,10,2	7.72	7.26	7.26	7.62	-5.9%	5.0%	24	2	[2,1,2,2]	22.85
4,80,12,2	7.50	6.96	6.96	7.83	-7.1%	12.4%	15	3	[1,2,2,2]	24.03
4,80,5,3	10.00	10.00	10.00	10.00	0.0%	0.0%	50	50	[3,3,3,3]	27.53
4,80,6,3	9.17	9.17	9.17	9.17	0.0%	0.0%	50	50	[3,3,3,3]	33.07
4,80,8,3	11.75	11.75	11.75	11.91	0.0%	1.4%	25	7	[3,3,3,3]	27.09
4,80,10,3	15.31	14.92	14.92	17.05	-2.5%	14.3%	7	1	[3,2,2,3]	28.40
4,80,12,3	9.51	9.51	9.51	9.51	0.0%	0.0%	50	50	[3,3,3,3]	29.59
4,80,5,4	11.57	11.57	11.57	11.57	0.0%	0.0%	50	50	[4,4,4,4]	19.31
4,80,6,4	14.81	14.72	14.72	14.72	-0.6%	0.0%	50	50	[3,4,4,4]	23.40
4,80,8,4	13.98	13.81	13.81	13.98	-1.2%	1.2%	25	8	[4,3,4,4]	19.72

Table 5.8 continued from previous page

p,n,m,q	Z^c	Best	Median	Worst	Best vs. Z^c	Worst vs. Best	# of Best	# of Worst	Correctly Selected Features	CPU (s)
4,80,10,4	12.89	12.89	12.89	12.89	0.0%	0.0%	50	50	[4,4,4,4]	20.80
4,80,12,4	19.30	19.30	19.30	19.83	0.0%	2.7%	26	1	[4,4,4,4]	21.92
4,100,5,2	6.92	6.91	6.91	6.91	-0.2%	0.0%	50	50	[2,2,2,2]	19.92
4,100,6,2	10.50	9.84	9.84	9.84	-6.3%	0.0%	50	50	[1,2,2,2]	20.66
4,100,8,2	11.91	11.50	11.50	11.90	-3.5%	3.5%	42	1	[2,2,2,2]	22.08
4,100,10,2	8.36	8.36	8.36	8.36	0.0%	0.0%	50	50	[2,2,2,2]	23.33
4,100,12,2	10.88	10.68	10.68	12.45	-1.8%	16.6%	19	2	[1,2,2,2]	24.78
4,100,5,3	12.30	12.30	12.30	12.30	0.0%	0.0%	50	50	[3,3,3,3]	25.00
4,100,6,3	12.27	12.27	12.27	12.27	0.0%	0.0%	50	50	[3,3,3,3]	25.67
4,100,8,3	14.28	14.11	14.11	14.28	-1.2%	1.2%	13	1	[3,3,2,3]	26.57
4,100,10,3	16.06	15.19	15.19	15.33	-5.4%	0.9%	9	3	[3,2,3,3]	28.41
4,100,12,3	13.84	13.84	13.84	13.84	0.0%	0.0%	50	50	[3,3,3,3]	28.88
4,100,5,4	15.72	15.72	15.72	15.84	0.0%	0.8%	49	1	[4,4,4,4]	19.63
4,100,6,4	18.74	18.52	18.52	18.52	-1.2%	0.0%	50	50	[3,4,4,4]	23.58
4,100,8,4	17.36	17.36	17.36	17.36	0.0%	0.0%	50	50	[4,4,4,4]	20.41
4,100,10,4	20.76	20.00	20.00	20.78	-3.6%	3.9%	9	2	[4,2,4,4]	21.34
4,100,12,4	23.82	20.89	20.94	22.43	-12.3%	7.4%	15	1	[4,2,4,4]	24.23
4,200,5,2	20.71	20.32	20.32	20.32	-1.9%	0.0%	44	6	[2,2,2,2]	22.02
4,200,6,2	15.65	15.59	15.59	15.87	-0.4%	1.8%	13	1	[1,2,2,2]	21.21
4,200,8,2	21.35	20.83	20.83	21.79	-2.4%	4.6%	48	1	[2,2,2,2]	23.13
4,200,10,2	17.16	17.12	17.12	17.12	-0.2%	0.0%	50	50	[2,2,2,2]	24.59
4,200,12,2	18.96	18.70	18.70	19.18	-1.4%	2.6%	31	1	[2,2,2,2]	27.04
4,200,5,3	25.10	25.10	25.10	25.57	0.0%	1.9%	13	1	[3,3,3,3]	27.06
4,200,6,3	29.73	29.73	29.73	29.73	0.0%	0.0%	50	50	[3,3,3,3]	27.60
4,200,8,3	29.56	29.56	29.56	30.16	0.0%	2.0%	19	5	[3,3,3,3]	29.81
4,200,10,3	27.43	27.43	27.43	28.33	0.0%	3.3%	48	2	[3,3,3,3]	31.02
4,200,12,3	24.93	24.93	24.93	24.93	0.0%	0.0%	50	50	[3,3,3,3]	35.63
4,200,5,4	33.37	33.27	33.27	33.37	-0.3%	0.3%	13	1	[4,3,4,4]	21.11
4,200,6,4	32.82	32.82	32.82	32.96	0.0%	0.4%	3	2	[4,4,4,4]	25.66
4,200,8,4	33.77	33.77	33.77	33.77	0.0%	0.0%	50	50	[4,4,4,4]	22.32
4,200,10,4	40.83	36.63	36.63	36.91	-10.3%	0.8%	37	1	[3,3,4,4]	23.92
4,200,12,4	38.03	38.03	38.38	40.49	0.0%	6.5%	5	1	[4,4,4,4]	27.06
4,500,5,2	39.23	39.14	39.14	39.14	-0.3%	0.0%	50	50	[2,2,2,2]	26.94
4,500,6,2	43.44	42.62	42.64	43.31	-1.9%	1.6%	10	1	[1,2,2,2]	28.95
4,500,8,2	56.33	56.17	56.17	56.17	-0.3%	0.0%	50	50	[2,2,2,2]	32.11
4,500,10,2	48.78	48.08	48.08	48.49	-1.4%	0.9%	12	2	[2,2,1,2]	34.96
4,500,12,2	50.18	46.46	52.34	54.70	-7.4%	17.7%	7	1	[2,1,2,2]	37.72
4,500,5,3	58.69	58.69	58.69	59.27	0.0%	1.0%	32	6	[3,3,3,3]	34.98
4,500,6,3	61.44	61.44	61.44	61.44	0.0%	0.0%	50	50	[3,3,3,3]	36.90
4,500,8,3	67.45	67.01	67.35	70.22	-0.7%	4.8%	23	1	[3,2,3,3]	41.25
4,500,10,3	71.90	68.73	68.80	70.78	-4.4%	3.0%	7	2	[3,2,3,3]	44.57
4,500,12,3	80.47	76.33	76.33	88.72	-5.1%	16.2%	34	1	[2,3,3,3]	47.97
4,500,5,4	75.75	75.75	76.79	78.29	0.0%	3.4%	12	1	[4,4,4,4]	29.82

Table 5.8 continued from previous page

p,n,m,q	Z^c	Best	Median	Worst	Best vs. Z^c	Worst vs. Best	# of Best	# of Worst	Correctly Selected Features	CPU (s)
4,500,6,4	78.51	78.51	78.51	78.56	0.0%	0.1%	23	13	[4,4,4,4]	35.45
4,500,8,4	92.09	92.09	92.09	93.45	0.0%	1.5%	12	1	[4,4,4,4]	32.53
4,500,10,4	89.16	89.16	89.16	94.77	0.0%	6.3%	26	1	[4,4,4,4]	34.23
4,500,12,4	99.55	98.92	98.92	103.52	-0.6%	4.6%	31	1	[4,3,4,4]	36.92
4,1000,5,2	99.94	99.19	99.19	99.78	-0.8%	0.6%	11	5	[2,2,2,2]	38.06
4,1000,6,2	99.57	98.41	98.41	98.41	-1.2%	0.0%	50	50	[2,2,2,2]	40.95
4,1000,8,2	102.14	102.00	102.00	105.02	-0.1%	3.0%	33	1	[2,2,2,2]	45.97
4,1000,10,2	72.55	72.49	72.49	72.49	-0.1%	0.0%	50	50	[2,2,2,2]	69.88
4,1000,12,2	114.87	107.31	111.57	117.54	-6.6%	9.5%	20	2	[1,2,2,2]	79.15
4,1000,5,3	132.09	131.87	131.87	133.26	-0.2%	1.1%	6	1	[2,3,3,3]	48.69
4,1000,6,3	132.10	132.10	132.10	134.63	0.0%	1.9%	29	2	[3,3,3,3]	57.83
4,1000,8,3	127.05	127.04	127.04	129.47	0.0%	1.9%	12	1	[3,3,3,3]	56.24
4,1000,10,3	126.13	126.13	126.13	126.13	0.0%	0.0%	50	50	[3,3,3,3]	72.24
4,1000,12,3	173.78	173.63	175.28	185.28	-0.1%	6.7%	1	1	[3,3,3,3]	82.01
4,1000,5,4	150.56	150.56	151.34	153.34	0.0%	1.8%	5	1	[4,4,4,4]	36.53
4,1000,6,4	148.63	146.41	146.41	150.36	-1.5%	2.7%	46	2	[3,4,4,4]	43.57
4,1000,8,4	158.48	158.48	159.05	160.65	0.0%	1.4%	14	1	[4,4,4,4]	43.64
4,1000,10,4	190.11	190.11	190.74	191.77	0.0%	0.9%	10	1	[4,4,4,4]	67.73
4,1000,12,4	198.71	193.73	195.87	199.53	-2.5%	3.0%	6	1	[4,3,4,4]	76.69

First of all, best result of the CCBFS algorithm are always less than or equal to the Z^c values in all data instances. It means that, proposed algorithm is either able to obtain same results with respect to data creation, or it deduces better clustering allocation than the data creation phase. As observed in the Tables 5.6 - 5.8, when the Z^c and Best result are the same, exactly q number of features are selected correctly in all clusters. Therefore, in these instances algorithm finds identical clustering and feature selection with the simulated data. CCBFS algorithm obtains the same clustering results in 113 over all 225 instances with respect to Z^c, while it is able to find better results than the Z^c in 108 instances. In these instances, CCBFS algorithm creates different clusters and/or select different features with respect to the simulated data.

As discussed in Chapter 4, number of clusters, p, affects the solution space dramatically. Therefore, when $p = 2$, CCBFS mostly finds the same results as in the data creation. On the other hand, the algorithm is able to obtain better results than Z^c, when number of clusters are more than 2.

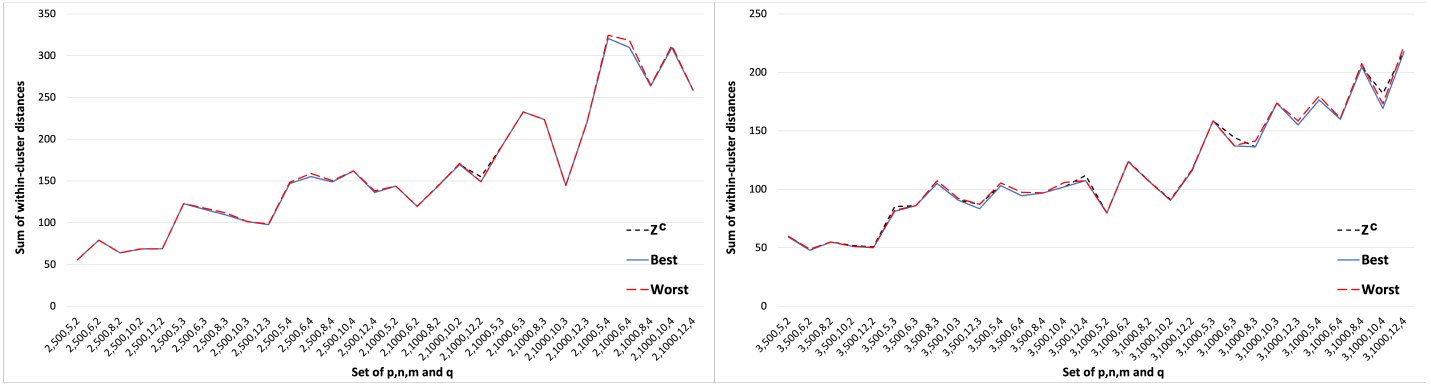
When the CPU times are compared, we have observed that the population size and number of clusters have dominant effect on our algorithm, as expected. Although the difference between two and three clusters are not significant, CPU times of the instances that $p = 4$ are much higher. In terms of number of data points, CPU times are higher especially for the instances with $n = 500$ and $n = 1000$. Last but not least, CPU times have increased in line with number of features, m.

Number of total features, m, has also consists significant effects on the simulation results. It is observed that, when the number of features are over eight, differences between best and worst results are higher with respect to instances with 5, 6 and 8 features. Moreover, CPU times also increase with increasing number of features which is expected since problem space is enlarged by m.

Averages of the performance measures are summarized in Table 5.9 in terms of number of clusters, p, and number of data points, n by grouping data sets with number of features m and number of selected features q.

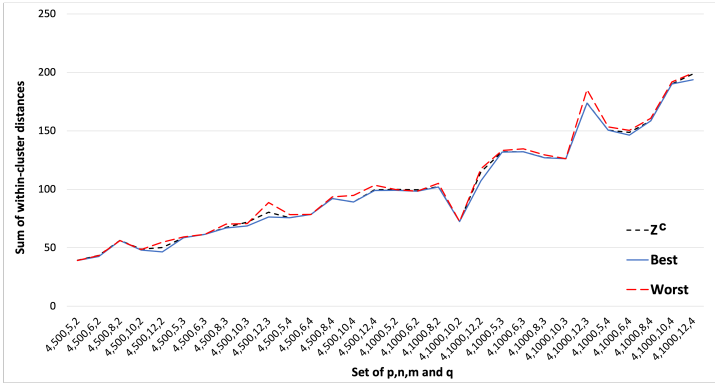
As shown in Table 5.9, CCBFS algorithm is able to find same results in all trials when $p = 2$ and $n = 80$. Moreover, while $p = 2$, number of best results are 50 in 55 of 75 instances, which means that same results are found in all 50 trials in these instances. Since best solution is always less than or equal to Z^c value, we deduce that the CCBFS is able to find successful clusters. In the remaining 20 instances, number of best results are 40 in average among 50 trials, while number of worst results are 5.4 in average.

Averages of number of best results have decreased for the p values of 3 and 4 which is an expected outcome, since the problem space increases dramatically by number of clusters. In the tests with three clusters, all results are the same in 31 over 75 instances, and this value has decreased to 26 for the tests with four clusters. Furthermore, number of best results are 40 and 30 in average for three and four clusters, respectively. On the other hand, number of worst solutions takes low values in almost all instances, so that it is assumed that the CCBFS algorithm has not stuck in a local optima in these tests. Number of worst results are above 10 out of 50 in only a few instances and it is observed that the gap between best and worst results are less than 1% in these instances.



(a) Two Clusters

(b) Three Clusters



(c) Four Clusters

Figure 5.1: Best and Worst Performances of CCBFS Algorithm vs. Z^c

Table 5.9: Summary of Performance Measures on Simulated Data Sets

p	n	Best vs. Z^c	Worst vs. Best	# of Best	# of Worst	CPU (s)
2	80	-0.3%	0.0%	50.00	0.00	19.63
	100	-0.3%	0.2%	49.53	0.47	18.23
	200	-0.3%	0.1%	47.80	2.20	20.23
	500	-0.1%	0.7%	43.53	3.00	26.19
	1000	-0.3%	0.4%	45.87	1.53	42.20
3	80	-1.1%	1.0%	44.93	1.07	18.65
	100	-2.3%	0.8%	39.40	2.80	17.86
	200	-3.5%	1.0%	43.67	2.20	19.93
	500	-1.1%	1.5%	39.00	1.60	26.50
	1000	-1.0%	1.0%	37.20	5.73	43.38
4	80	-2.2%	2.6%	37.13	1.53	24.42
	100	-2.4%	2.3%	37.07	0.73	23.63
	200	-1.1%	1.6%	31.60	1.47	25.95
	500	-1.5%	4.1%	25.27	2.07	35.69
	1000	-0.9%	2.3%	22.87	1.27	59.88

Another prominent advantage of the proposed algorithm is small percentage gap levels between best and worst solutions. It means that the algorithm is able to provide convenient clusters in each trial. For two cluster data sets, the average difference between worst and best solutions are only 0.3% and it is always less than 3%. Average differences for three and four clusters are 1.1% and 2.6%, respectively. Moreover, the difference is always below 5.5% for three clusters, and less than 10% for four clusters in 70 over 75 problem instances. In four of remaining five instances, it is observed that the difference between best and median results are less than 0.1%, so that CCBFS algorithm obtains reliable results even in these instances.

In order to show relationship between best and worst solutions of the trials with respect to ground truth objective value, Figure 5.1 is provided. The figure includes data sets with 500 data points and above. The instances that includes less than 500 data

points are not illustrated since the differences are very small to analyze. As illustrated in the Figure 5.1, the gap between the best and worst soluıtions are tight in almost all the cases. Moreover, they do not differ significantly from the Z^c value in all of the trials.

5.5 Comparison of Computational Results with Benchmark Algorithms

As stated in Section 3.2, we have compared the performance of CCBFS algorithm with Öz (2019) [35]. She has developed two heuristic algorithms in addition to three linearized models to solve clustering with cluster based features selection problem. For the comparison, data sets that are created by Öz (2019) are utilized. Moreover, the best results among the heuristic algorithms or linearized models are taken into consideration for each instance separately. We denote the best results of Öz (2019) as Z^*, and worst results as Z^*_w. It should be noted that, Öz (2019) finds the optimal results in 80 of 225 instances (36/75 for $p = 2$, 22/75 for $p = 3$, and 22/75 for $p = 4$). It is important to note that she is able to obtain optimal results in these cases through heuristic algorithms.

We have compared the results of CCBFS algorithm with respect to following performance measures:

% Difference in the Best Solution : Z_{Best} and Z^* values are compared with respect to following formula

$$\textit{\% Diff. in the Best Solution} = \left(\frac{Z_{Best} - Z^*}{Z^*} \right) * 100$$

% Difference in the Worst Solution : Z_{Worst} and Z^*_w values are compared with respect to following formula

$$\textit{\% Diff. in the Worst Solution} = \left(\frac{Z_{Worst} - Z^*_w}{Z^*_w} \right) * 100$$

# of Hits to the Best: This measure reports us the replication of the best results for both Z_{Best} and Z^*

Table 5.10: Summary of Result Comparison between CCBFS Algorithm and Öz (2019) [35]

	# of Z_{Best} **<**Z^*	**Avg. Diff. in** Z_{Best} **vs** Z^*	**# of** Z_{Worst}**<**Z_w^*	**# of Diff. in** Z_{Worst} **vs** Z_w^*	**# of Hits to Best in CCBFS**	**# of Hits to Best in [35]**
p = 2	0/75	-	41/75	-29.4%	41.5	34.9
p = 3	14/75	-2.5%	65/75	-32.2%	23.1	20.3
p = 4	16/75	-2.2%	75/75	-33.7%	24.7	16.9

For the % differences, negative values represent that CCBFS algorithm is able to obtain smaller within-cluster distances, and vice versa. Detailed results are given in Tables 5.11 - 5.13 at the end of this section. Furthermore, Table 5.10 provides a summary of results comparison for various values of p.

First of all, in terms of best solutions, Z_{Best} obtains within-cluster distances that are less than or equal to with respect to Z^* in all 225 cases. In the instances with $p = 2$, best solutions are the same in two studies for all 75 instances. However, CCBFS algorithm obtains significantly lower solutions in the worst solutions. Z_{Worst} is less than Z_w^* in 41 out of 75 instances and it is less than 29.4% in these 41 instances, in average. Furthermore, CCBFS algorithm also hits the best results in 41.5 out of 50 instances in average. Although best results in both methods are the same, proposed algorithm is able hit these results more.

Results has shown a similar structure for three and four cluster instances. Among 75 instances Z_{Best} obtains better results in 14 and 16 instances for $p = 3$ and $p = 4$, respectively. Moreover, Z_{Best} provides more than 2% improvements in these instances in average. Furthermore, CCBFS algorithm hits the best results in almost half of the trials in average. Therefore, it provides more reliable results than Öz (2019). Differences in the worst solutions are much more obvious on p values of three and four. When $p = 3$, Z_{Worst} is less than Z_w^* in 65 instances and there are 32.2% difference in these instances. Remarkably, Z_{Worst} obtains better solutions in all 75 instances when $p = 4$, and it provides 33.7% difference in average.

As result, CCBFS algorithm obtains much smaller differences between best and worst

solutions. Also, we are able to find better best results on more complex problems. Moreover, our proposed algorithm is able to hit the best results more frequently with respect to Öz (2019).

Table 5.11: Comparison of the Results for CCBFS Algorithm and Öz (2019) [35] on $p = 2$

p,n,m,q	% Diff. in the Best Solution	% Diff. in the Worst Solution	# of Best in CCBFS	# of Best in [35]	p,n,m,q	% Diff. in the Best Solution	% Diff. in the Worst Solution	# of Best in CCBFS	# of Best in [35]
2,80,5,2	0.0%	0.0%	50	50	2,200,10,3	0.0%	-39.3%	43	47
2,80,6,2	0.0%	-38.4%	50	48	2,200,12,3	0.0%	-40.8%	37	9
2,80,8,2	0.0%	-26.3%	44	4	2,200,5,4	0.0%	0.0%	50	50
2,80,10,2	0.0%	-36.8%	49	6	2,200,6,4	0.0%	1.1%	32	50
2,80,12,2	0.0%	-28.6%	31	45	2,200,8,4	0.0%	6.4%	48	50
2,80,5,3	0.0%	10.1%	47	47	2,200,10,4	0.0%	25.1%	38	50
2,80,6,3	0.0%	19.1%	49	50	2,200,12,4	0.0%	-36.5%	34	32
2,80,8,3	0.0%	10.8%	48	50	2,500,5,2	0.0%	0.0%	50	50
2,80,10,3	0.0%	-29.0%	37	36	2,500,6,2	0.0%	-44.8%	50	43
2,80,12,3	0.0%	-25.3%	5	39	2,500,8,2	0.0%	-22.7%	50	3
2,80,5,4	0.0%	0.0%	50	50	2,500,10,2	0.0%	-17.5%	50	1
2,80,6,4	0.0%	9.4%	49	50	2,500,12,2	0.0%	-14.6%	15	2
2,80,8,4	0.0%	5.5%	31	49	2,500,5,3	0.0%	0.0%	50	49
2,80,10,4	0.0%	30.3%	26	36	2,500,6,3	0.0%	-7.8%	50	23
2,80,12,4	0.0%	1.7%	11	49	2,500,8,3	0.0%	-47.9%	49	35
2,100,5,2	0.0%	0.0%	28	35	2,500,10,3	0.0%	-32.4%	50	43
2,100,6,2	0.0%	-42.7%	31	9	2,500,12,3	0.0%	-11.6%	30	1
2,100,8,2	0.0%	-4.8%	48	32	2,500,5,4	0.0%	0.0%	50	50
2,100,10,2	0.0%	-36.3%	47	41	2,500,6,4	0.0%	0.0%	50	50
2,100,12,2	0.0%	-3.3%	36	1	2,500,8,4	0.0%	0.0%	50	49
2,100,5,3	0.0%	0.0%	50	50	2,500,10,4	0.0%	-30.8%	47	13
2,100,6,3	0.0%	0.0%	50	50	2,500,12,4	0.0%	-24.8%	38	28
2,100,8,3	0.0%	-42.4%	30	45	2,1000,5,2	0.0%	0.0%	50	50
2,100,10,3	0.0%	31.6%	28	21	2,1000,6,2	0.0%	0.0%	50	50
2,100,12,3	0.0%	-16.5%	24	32	2,1000,8,2	0.0%	-23.7%	50	5
2,100,5,4	0.0%	0.0%	50	50	2,1000,10,2	0.0%	-15.8%	50	13
2,100,6,4	0.0%	6.3%	48	50	2,1000,12,2	0.0%	-13.5%	32	10
2,100,8,4	0.0%	24.5%	38	50	2,1000,5,3	0.0%	0.0%	50	50
2,100,10,4	0.0%	-32.5%	16	24	2,1000,6,3	0.0%	0.0%	50	50
2,100,12,4	0.0%	-25.1%	20	33	2,1000,8,3	0.0%	-45.1%	28	33
2,200,5,2	0.0%	-33.9%	42	40	2,1000,10,3	0.0%	-29.2%	50	35
2,200,6,2	0.0%	0.2%	35	50	2,1000,12,3	0.0%	-23.8%	47	5
2,200,8,2	0.0%	-34.6%	36	25	2,1000,5,4	0.0%	1.3%	48	50
2,200,10,2	0.0%	-35.7%	50	26	2,1000,6,4	0.0%	0.0%	50	50
2,200,12,2	0.0%	-32.6%	47	16	2,1000,8,4	0.0%	1.2%	44	50
2,200,5,3	0.0%	3.8%	37	48	2,1000,10,4	0.0%	-32.7%	48	12
2,200,6,3	0.0%	-27.6%	50	48	2,1000,12,4	0.0%	-35.4%	36	26
2,200,8,3	0.0%	-39.2%	49	12					

Table 5.12: Comparison of the Results for CCBFS Algorithm and Öz (2019) [35] on $p = 3$

p,n,m,q	% Diff. in the Best Solution	% Diff. in the Worst Solution	# of Best in CCBFS	# of Best in [35]	p,n,m,q	% Diff. in the Best Solution	% Diff. in the Worst Solution	# of Best in CCBFS	# of Best in [35]
3,80,5,2	**-8.8**%	-37.4%	35	13	3,200,10,3	0.0%	-26.6%	5	16
3,80,6,2	0.0%	-37.2%	42	14	3,200,12,3	**-0.9**%	-29.6%	4	2
3,80,8,2	0.0%	-9.6%	49	4	3,200,5,4	0.0%	-52.5%	50	36
3,80,10,2	**-2.0**%	-19.2%	9	1	3,200,6,4	0.0%	-54.5%	15	39
3,80,12,2	0.0%	-16.5%	12	1	3,200,8,4	0.0%	-55.7%	16	2
3,80,5,3	0.0%	0.0%	49	36	3,200,10,4	0.0%	-36.4%	1	33
3,80,6,3	0.0%	-47.6%	22	31	3,200,12,4	0.0%	-27.5%	1	23
3,80,8,3	0.0%	-18.2%	1	8	3,500,5,2	0.0%	-33.4%	50	17
3,80,10,3	**-0.1**%	-31.3%	1	2	3,500,6,2	0.0%	-33.4%	50	20
3,80,12,3	0.0%	-32.6%	1	17	3,500,8,2	0.0%	-12.2%	1	2
3,80,5,4	0.0%	-63.6%	31	33	3,500,10,2	0.0%	-14.1%	49	2
3,80,6,4	0.0%	-55.3%	43	31	3,500,12,2	0.0%	-6.2%	14	1
3,80,8,4	0.0%	-38.9%	8	42	3,500,5,3	0.0%	0.9%	35	37
3,80,10,4	0.0%	-44.4%	4	1	3,500,6,3	0.0%	0.0%	49	39
3,80,12,4	**-0.5**%	-27.3%	1	9	3,500,8,3	0.0%	-38.3%	25	27
3,100,5,2	0.0%	-36.8%	40	38	3,500,10,3	0.0%	-4.6%	18	10
3,100,6,2	**-4.7**%	-33.4%	14	9	3,500,12,3	0.0%	-5.3%	1	10
3,100,8,2	**-2.6**%	-13.6%	12	3	3,500,5,4	0.0%	0.5%	41	40
3,100,10,2	**-1.4**%	-21.0%	10	1	3,500,6,4	0.0%	-49.4%	43	40
3,100,12,2	**-3.0**%	-13.7%	9	1	3,500,8,4	0.0%	-58.4%	31	39
3,100,5,3	0.0%	0.0%	47	37	3,500,10,4	0.0%	-44.2%	10	7
3,100,6,3	0.0%	-54.4%	44	38	3,500,12,4	0.0%	-30.7%	3	1
3,100,8,3	0.0%	-19.7%	1	30	3,1000,5,2	0.0%	-28.0%	20	18
3,100,10,3	0.0%	-31.0%	2	11	3,1000,6,2	0.0%	-32.6%	10	25
3,100,12,3	0.0%	-29.6%	1	9	3,1000,8,2	0.0%	-21.9%	31	15
3,100,5,4	0.0%	-54.8%	43	38	3,1000,10,2	0.0%	-11.0%	31	1
3,100,6,4	0.0%	0.0%	50	37	3,1000,12,2	0.0%	-4.8%	14	1
3,100,8,4	0.0%	-49.3%	14	36	3,1000,5,3	0.0%	0.0%	31	40
3,100,10,4	0.0%	-42.8%	1	9	3,1000,6,3	0.0%	2.3%	18	34
3,100,12,4	**-2.0**%	-22.1%	1	3	3,1000,8,3	0.0%	-43.6%	50	38
3,200,5,2	0.0%	-38.8%	50	12	3,1000,10,3	0.0%	-32.9%	18	31
3,200,6,2	0.0%	-37.7%	36	19	3,1000,12,3	0.0%	-5.8%	7	9
3,200,8,2	**-4.8 %**	-15.2%	21	4	3,1000,5,4	0.0%	1.5%	35	38
3,200,10,2	**-1.1%**	-15.4%	11	1	3,1000,6,4	0.0%	3.3%	49	38
3,200,12,2	**-0.02 %**	-11.3%	15	1	3,1000,8,4	0.0%	-62.5%	17	38
3,200,5,3	0.0%	-53.9%	34	36	3,1000,10,4	0.0%	-54.4%	42	34
3,200,6,3	0.0%	-48.2%	50	40	3,1000,12,4	**-4.3**%	-36.1%	20	30
3,200,8,3	0.0%	-27.0%	16	32					

Table 5.13: Comparison of the Results for CCBFS Algorithm and Öz (2019) [35]

$p = 4$

p,n,m,q	% Diff. in the Best Solution	% Diff. in the Worst Solution	# of Best in CCBFS	# of Best in [35]	p,n,m,q	% Diff. in the Best Solution	% Diff. in the Worst Solution	# of Best in CCBFS	# of Best in [35]
4,80,5,2	0.0%	-15.1%	26	3	4,200,10,3	**-4.7**%	-19.7%	1	3
4,80,6,2	**-3.4**%	-16.6%	38	6	4,200,12,3	0.0%	-16.6%	17	14
4,80,8,2	0.0%	-13.0%	42	2	4,200,5,4	0.0%	-58.6%	45	21
4,80,10,2	**-0.03**%	-11.3%	36	1	4,200,6,4	0.0%	-59.6%	49	28
4,80,12,2	**-7.2**%	-14.8%	3	2	4,200,8,4	0.0%	-47.6%	10	40
4,80,5,3	0.0%	-64.8%	5	44	4,200,10,4	0.0%	-30.1%	41	19
4,80,6,3	0.0%	-29.1%	18	28	4,200,12,4	0.0%	-20.8%	5	10
4,80,8,3	0.0%	-31.3%	35	13	4,500,5,2	0.0%	-56.5%	50	14
4,80,10,3	0.0%	-11.9%	1	2	4,500,6,2	0.0%	-36.7%	50	15
4,80,12,3	0.0%	-5.8%	2	9	4,500,8,2	0.0%	-33.5%	29	3
4,80,5,4	0.0%	-56.4%	46	35	4,500,10,2	**-0.5**%	-9.3%	39	1
4,80,6,4	0.0%	-60.4%	24	29	4,500,12,2	**-0.1**%	-9.1%	26	2
4,80,8,4	0.0%	-51.1%	17	35	4,500,5,3	0.0%	-58.5%	30	43
4,80,10,4	0.0%	-27.3%	10	6	4,500,6,3	0.0%	-55.1%	34	32
4,80,12,4	0.0%	-15.5%	1	3	4,500,8,3	0.0%	-38.7%	33	17
4,100,5,2	0.0%	-25.9%	50	9	4,500,10,3	0.0%	-39.8%	9	12
4,100,6,2	0.0%	-18.6%	9	13	4,500,12,3	**-3.4**%	-4.8%	8	16
4,100,8,2	0.0%	-7.8%	49	1	4,500,5,4	0.0%	-56.8%	50	29
4,100,10,2	**-3.9** %	-11.3%	23	1	4,500,6,4	0.0%	-59.5%	36	30
4,100,12,2	**-0.2** %	-9.9%	7	1	4,500,8,4	0.0%	-38.6%	24	44
4,100,5,3	0.0%	-44.3%	48	40	4,500,10,4	0.0%	-37.4%	18	26
4,100,6,3	0.0%	-52.9%	22	31	4,500,12,4	0.0%	-31.8%	2	13
4,100,8,3	0.0%	-36.9%	26	15	4,1000,5,2	0.0%	-39.0%	37	27
4,100,10,3	**-2.7** %	-10.0%	1	3	4,1000,6,2	0.0%	-40.0%	15	10
4,100,12,3	**-0.2** %	-14.9%	1	2	4,1000,8,2	0.0%	-45.4%	3	1
4,100,5,4	0.0%	-58.3%	50	35	4,1000,10,2	**-5.9**%	-8.2%	9	1
4,100,6,4	0.0%	-57.7%	50	38	4,1000,12,2	**-0.03**%	-5.2%	10	1
4,100,8,4	0.0%	-46.6%	50	37	4,1000,5,3	0.0%	-64.8%	15	42
4,100,10,4	0.0%	-32.3%	33	8	4,1000,6,3	0.0%	-64.2%	50	38
4,100,12,4	**-0.6**%	-11.7%	2	20	4,1000,8,3	0.0%	-44.7%	39	27
4,200,5,2	**-1.6**%	-31.2%	46	1	4,1000,10,3	0.0%	-38.7%	22	6
4,200,6,2	0.0%	-22.5%	50	1	4,1000,12,3	0.0%	-24.5%	8	1
4,200,8,2	0.0%	-11.5%	32	4	4,1000,5,4	0.0%	-65.7%	11	38
4,200,10,2	**-0.2**%	-7.4%	3	2	4,1000,6,4	0.0%	-51.7%	6	34
4,200,12,2	0.0%	-6.5%	27	1	4,1000,8,4	0.0%	-53.9%	41	34
4,200,5,3	0.0%	-64.1%	49	35	4,1000,10,4	0.0%	-46.5%	5	8
4,200,6,3	0.0%	-44.7%	17	17	4,1000,12,4	0.0%	-28.3%	1	23
4,200,8,3	0.0%	-36.2%	23	14					

CHAPTER 6

MEMETIC ALGORITHM FOR CLUSTERING WITH CLUSTER BASED FEATURE SELECTION FOR HIGH DIMENSIONAL DATA SETS (CCBFS-H)

The CCBFS algorithm provides successful clustering when there are high number of data points and a few features as discussed in Chapter 4. In clustering, selecting features from a high dimensional data set is also important to represent clusters in lower dimensions. However, the proposed could not provide fruitful insights in such data sets, since solution space of the memetic algorithm is not decreased significantly. Therefore, CCBFS algorithm is modified in order to obtain successful clusters in high dimensional data sets.

In this chapter, our modified memetic algorithm for clustering with cluster based feature selection for high dimensional data sets (CCBFS-H) is presented. In the Section 6.1, the algorithm is introduced and the process of the algorithm is explained. Subsequently, in Section 6.2, each step of the algorithm is described in detail.

6.1 Overview of CCBFS-H

The data set possesses the similar features as in the Chapter 4. There are m features in the data set which assumed to include relevant q and redundant $(m - q)$ features. On the other hand, there are n data points in a data set, and all data points in the data set are subject to be assigned one of p clusters. Main difference in this chapter is the data sets that we studied contain high number of features.

Although CCBFS algorithm obtains successful results by squeezing the solution space

for the data sets with high number of points, increasing values of m affects the space of CCBFS algorithm significantly, which is shown in Table 6.1. Since we store the information of selected features in the chromosome representation of the CCBFS algorithm, increasing number of features, m, and selected features q, affect the problem space significantly. Therefore, we modify the algorithm by discarding the selected features from the chromosome representation which is explained in detail in Section 6.2.

In our renewed memetic algorithm, chromosomes include the information of assignment and cluster centroids. When the assignment and cluster centroids are known, a distance value can be calculated for each feature. Then, the features are sorted in terms of their distance value, and q features with minimum distance values are selected for each cluster. Sum of the distance values for q features in each cluster represents the fitness value. The main aim of this representation is to work with data sets including high number of features. Since the fittest features are selected with respect to centroid and assignment selection, solution space is reduced significantly.

Furthermore, we add another type of local search method in this algorithm. First method aims to find fitter centroid selections as discussed in Section 4.2.6. The developed method in this Chapter aims to find better assignments in a pre-determined neighborhood.

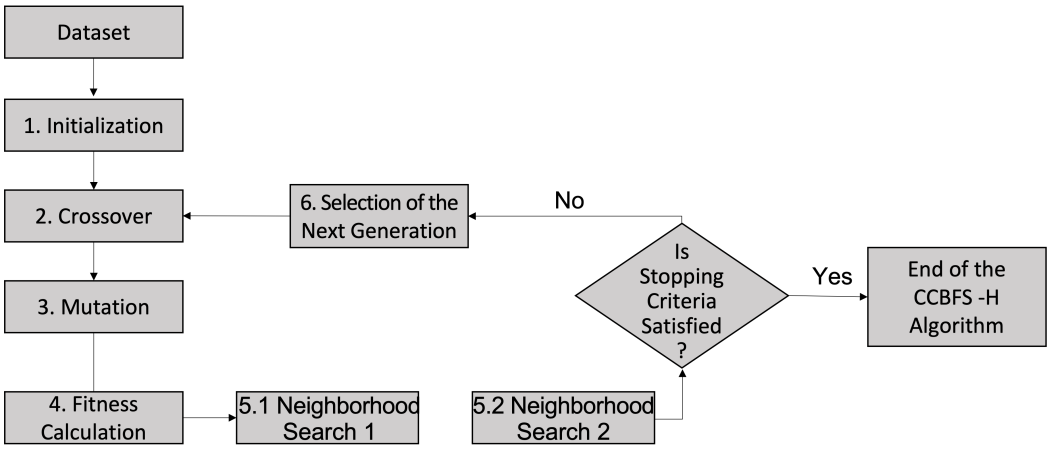


Figure 6.1: Flowchart of the CCBFS-H Algorithm

The flowchart of the memetic algorithm is shown in the Figure 6.1. Steps 1 and 2

are similar to CCBFS algorithm. However, due to the representation of chromosomes some exceptions are defined in these steps. Subsequently, one type of mutation operation is applied to the offsprings with a small probability. After the mutation, a fitness value is calculated for each individual in the population. In step 5, neighborhood search algorithms are applied to the fittest individual in the population. These algorithms are added into the proposed method in order to catch some centroids and assignments which might provide fitter solutions. Then, the parents of the next generation are selected from the feasible individuals of the current generation. The creation of offspring process continues until the stopping criteria is satisfied. At the end of the algorithm, the fittest offspring through all generations is taken as the best individual and its phenotype is reported as the best result of the CCBFS–H.

6.2 Description of CCBFS-H Algorithm

In this section, each step of CCBFS-H algorithm is explained in detail.

6.2.1 Chromosome Representation

Chromosome representation is the most significant alteration in development of the CCBFS-H algorithm. Since this algorithm is developed for the inputs with high number of features, storing the feature information on genotype of the chromosomes will be resulted as immense increase on the solution space. Thus, we discard the the information of the features on the chromosome representation and utilize the assignment and cluster centroid information as illustrated in the Figure 6.2.

There are 2 components in the chromosome representation. In the first component, assignments of data points are stored. Each cell stores the information of the assignment of the related data point and it could take a value in the range of 1 and p. Therefore, length of the component one is equal to the number of data points n. Second component of a chromosome holds the indices of cluster centroids which represents the related data point, as in the chromosome representation of CCBFS algorithm. It is important to note that a centroid of a cluster should be assigned accordingly in the first component of the chromosome. Furthermore, in accordance with the problem

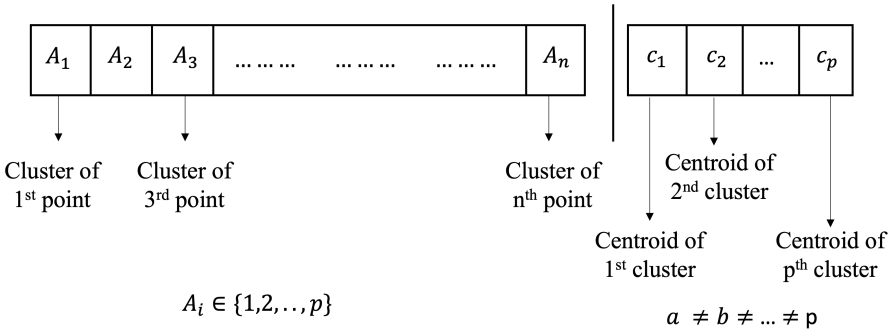


Figure 6.2: Chromosome Representation of CCBFS-H

definition, one data point can be assigned as centroid to at most one cluster. Length of the second component equals to the number of clusters p.

In Figure 6.3, an example for the chromosome representation is provided. In this example, we have three clusters and twelve data points. Here, 5^{th}, 8^{th} and 11^{th} data points are clustered as cluster 1, and 8^{th} data point is selected as the centroid. Similarly, 2^{nd}, 3^{rd}, 9^{th} and 12^{th} data points form a cluster and the remaining five data points form another one.

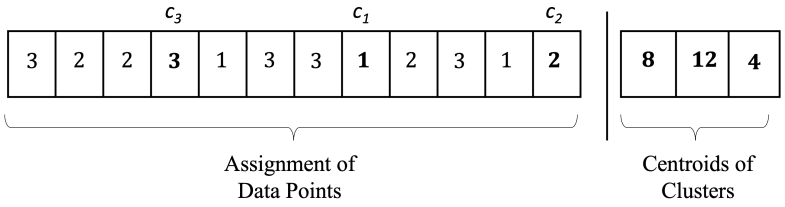


Figure 6.3: Chromosome Representation of CCBFS-H

This representation keeps only $(N + p)$ values so that it utilizes low amount of memory. In order to obtain the phenotype of an individual from this representation, the least costly features are selected from this assignment which is explained in the fitness calculation.

Last but not the least, chromosome representation of the CCBFS-H algorithm enables us to work with the data sets including high number of features. As studied in Chapter 7, we test CCBFS-H algorithm in the data sets that have high number of m and q values. Since features can be analyzed after the selection of centroids and assignments, a

dramatic decrease is obtained in the solution space of CCBFS-H algorithm. The total solution space of an instance in this algorithm can be formulated as follows:

$$\binom{n}{p}\left(p^{n-p}\right) \tag{6.1}$$

In the first term of Equation 6.1, cluster centroids are selected among n data points. These centroids are automatically assigned to their own clusters and remaining $(n-p)$ data points are assigned either one of the p clusters which is represented in the second term. It should be noted that the m and q values do not affect the solution space, therefore the solution space remains the same on constant values of n and p as shown in Table 6.1. Comparison of the solution sizes with the original problem and proposed algorithms is also provided in this Table.

Table 6.1: Solution Space on Various Values of n, m and q

p	n	m	q	# of Solutions in the Original Problem	# of Solutions in CCBFS Algorithm	# of Solutions in CCBFS-H Algorithm
2	80	12	4	$1.48 * 10^{29}$	$7.74 * 10^{8}$	$9.55 * 10^{26}$
2	100	12	4	$1.55 * 10^{35}$	$1.21 * 10^{9}$	$1.57 * 10^{33}$
2	100	500	30	$1.32 * 10^{126}$	$1.03 * 10^{100}$	$1.57 * 10^{33}$
2	100	1000	30	$3.74 * 10^{144}$	$2.92 * 10^{118}$	$1.57 * 10^{33}$

6.2.2 Initialization

The initialization concept of the CCBFS-H algorithm is same as in the CCBFS algorithm, since the idea is the same. In order to obtain individuals in a wide range of variations, we create initial population from three different methods. Details of the initial population creation is already explained in Section 4.2.2 and in Algorithm 2.

6.2.3 Uniform Crossover Operator

All of the parents are grouped as pairs in order to create offsprings through crossover operation. So that half of the population size (pop_{size}) is subject to crossover. The

uniform crossover operator is applied to each component of the chromosomes by utilizing a crossover mask. Therefore, it includes two parts and up to four offsprings are generated from two parents. Uniform crossover is selected for crossover since it increases the exploration capabilities of the algorithm, and it is an unbiased operator in terms of ordering. First two offsprings are generated through the uniform crossover of parents' assignments. A crossover mask is created via random binary variables and crossover is realized. Similarly, a crossover mask is created for the centroids and offspring three and four are generated from the crossover from second components of their parents. For a certain data point, if randomly chosen value in the crossover mask is 1, then parent 1's genotype is transferred to offspring 1 and similarly parent 2's genotype is transferred to offspring 2. On the other hand, if randomly chosen value is 0, then corresponding genotype of parent 1 is transferred to offspring 2. Creation of four offsprings via uniform crossover is illustrated in Figure 6.4.

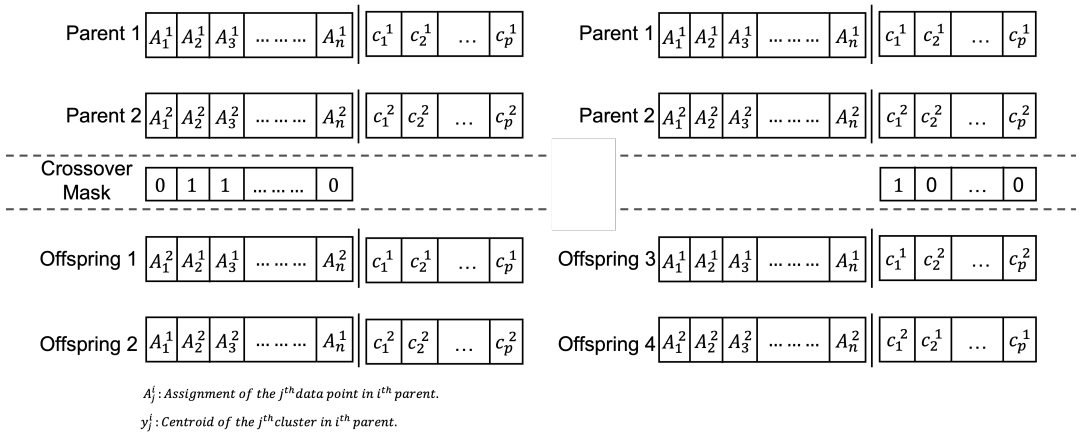


Figure 6.4: Uniform Crossover Operator for CCBFS-H

Since two components of the chromosomes are related with each other, we also need to grant two exceptions on our crossover operator.

i) While generating the first two offsprings, we directly transfer centroids of the parents to the offsprings. So, the assignment of the corresponding data point in the first component of the chromosome should be same as the second component. Therefore, we separate corresponding assignments of the centroids, and they are not considered in the crossover operation. This exception is represented in the Figure 6.5. In the example 3^{rd} and 72^{nd} data points are centroids

of parent 1, and 1st and 73^{rd} data points are centroids of parent 2, and no crossover mask is generated for them since the parents' centroid selection is copied to these offsprings. (line 7)

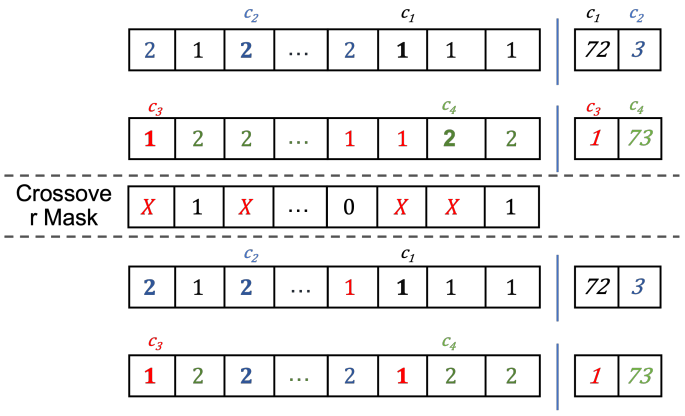


Figure 6.5: An example for Crossover Exception I

ii) The second exception is utilized while generating last two offsprings. As in the first exemption we directly transfer the assignment information in the centroids that are switched during crossover in order to keep assignment-centroid coherence in the offsprings. An example for the second exception is illustrated in the Figure 6.6. (lines 23-26)

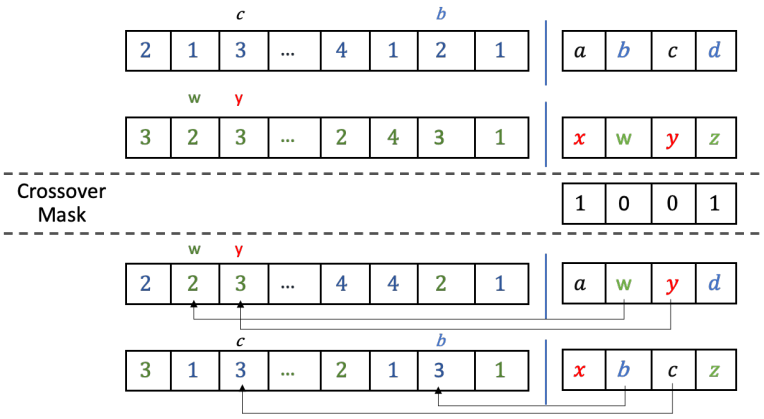


Figure 6.6: An example for Crossover Exception II

Apart from the exceptions, it is important to note that some of the offsprings might be infeasible for our problem. For instance, after the crossover operation one offspring may have a data point that is the centroid for more than one clusters. This situation is

beyond the problem definition and such offsprings are discarded from the population and neglected in further steps.

Pseudo-code of the uniform crossover operation for CCBFS-H is provided in Algorithm 8

6.2.4 Mutation Operator

Random mutation operator is also designed in the CCBFS-H algorithm, as in the CCBFS algorithm. The aim of this operator is to switch assignment of the data points. In this algorithm we only use one type of mutation, since we also search through assignments in the neighborhood search step. Algorithm 9 explains the flow of the mutation operator.

All of the individuals in the population with a pre-defined probability ($prob_{mut}$) might be exposed to mutation, and this probability is considered as constant through generations. In case the mutation operator is applied to an offspring, two data points in this offspring from different clusters are selected. Then, their clusters are interchanged. It should be noted that, one of the selected data points for mutation might be cluster centroid. In this case, we discard these individuals from the population in the fitness calculation step. To illustrate, an example is given in Figure 6.7. In the example, the mutation operator is applied to the clustering problem where the number of data points and the number of clusters are equal to 14 and 2, respectively. Here, 9^{th} and 12^{th} data points are selected from first and second clusters and their assignments are switched.

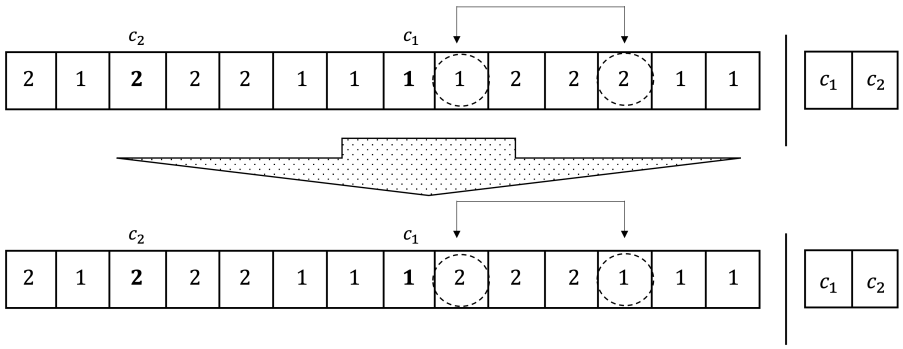


Figure 6.7: An example for Mutation in CCBFS-H Algorithm

Algorithm 8: Uniform Crossover Operation for CCBFS-H

Input : Current population

Output: Offsprings

$k = 1$
while $k \leq pop_{size}/2$ **do**
    Offspring1_centroids = Parent1_centroids
    Offspring2_centroids = Parent2_centroids
    Create a $1 * n$ vector of boolean, $mask_1$
    for *a=1,...,n* **do**
        if $mask_1(a) = 1$ *OR* $a \in c_i^j$ **then**
            Offspring1_assignment(a)=Parent1_assignment(a)
            Offspring2_assignment(a)=Parent2_assignment(a)
        else
            Offspring1_assignment(a)=Parent2_assignment(a)
            Offspring2_assignment(a)=Parent1_assignment(a)
        end if
    end for
    Offspring3_assignment = Parent1_assignment
    Offspring4_assignment = Parent2_assignment
    Create a $1 * p$ vector of boolean $mask_2$
    for *b=1,...,p* **do**
        if $mask_2(b) = 1$ **then**
            Offspring3_centroid(b)=Parent1_centroid(b)
            Offspring4_centroid(b)=Parent2_centroid(b)
        else
            Offspring3_centroid(b)=Parent2_centroid(b)
            Offspring4_centroid(b)=Parent1_centroid(b)
            Offspring3_assignment(c_b)= Parent2_assignment(c_b)
            Offspring4_assignment(c_b)= Parent1_assignment(c_b)
        end if
    end for
end while

Algorithm 9: Mutation Operation for CCBFS-H

Input : Offsprings in the Current Population, Probability of mutation ($prob_{mut}$)

Output: Offsprings after Mutation Operation

```
r=rand[0,1]
if r ≤ prob_mut then
    k=0
    while k==0 do
        Select assignments of two data points randomly, A_1 and A_2
        if A_1 ≠ A_2 then
            Switch the clusters of selected data points
            k=1
        end if
    end while
end if
```

6.2.5 Fitness Function

After the mutation operation, assignment of data points to the clusters is done and fitness values for both parents and offsprings are calculated. Process of the fitness calculation in CCBFS-H algorithm is highly similar as in the CCBFS algorithm. However, as the representation of the chromosomes are different, fitness calculation method is also differ.

We have selected the objective function of the clustering problem as the fitness function, which is the total sum of the rectilinear distances between data points and their cluster centers.

To calculate the fitness of an offspring, we calculate the distance values for each feature in the data set by considering cluster centroids and assignments. Subsequently, we order features for each cluster and the q features with least distance value are selected. Finally, sum of distance values from selected features for all clusters is obtained and within-cluster distance value is assigned as the fitness value of the corresponding offspring.

Algorithm 10: Fitness Calculation for CCBFS-H

Input : Data set, Offspring's Chromosome

Output: Fitness Value, Cluster Assignment

```
for i=1,...,p do
    for j=1,...,m do
        Calculate L_1 − norm distance of j^th feature for i^th cluster
    end for
    Order the features with respect to total distance in i^th cluster
    Select q features that provides minimum distances
    Calculate within-cluster distance of i^th cluster
end for
Calculate the sum of within-cluster distances as fitness value
```

6.2.6 Neighborhood Search

In CCBFS-H Algorithm we utilized two different local search algorithms in order to search potential centroids, and avoid to make poor assignments. But the main purpose is testing the potential centroids and data points in the neighborhood which might provide lower within-cluster distances.

Algorithm 11: Neighborhood Search 2 for CCBFS-H

Input : Chromosome of the Fittest Offspring, (n_{cand})

Output: Chromosome of the Fittest Offspring

```
for i=1,...,n_cand do
    for j=1,...,p do
        Select the j^th farthest data point in cluster i
        Calculate its distance to all cluster centroids, and find the closest
         cluster k,
        Switch the assignment of j^th farthest data point in cluster i to cluster k
    end for
end for
```

The first neighborhood search is highly similar to the search method in Section 4.2.6, and there are minor differences since chromosome representation is different. Here,

selected features are determined in accordance with the assignment of the data points and selected centroids. Then, the same neighborhood search process is applied as in the Section 4.2.6. Thanks to this local search method, better centroids might be found, and if there is a fitter centroid selection, this individual is added into the population as a new member.

Furthermore, since assignments in the chromosome representation are updated through crossover operator, some of the assignments might be unfit for their clusters. Thus, a second neighborhood search is developed which aims to detect farthest data points in a cluster and search a closer cluster centroid for these data points. Second neighborhood search in CCBFS-H is applied after the first neighborhood search. Here, initially we detect the farthest number of candidate assignments (n_{cand}) in each cluster. Then, each of these data point's distance to each cluster centroid is calculated, and these data points are assigned to their nearest cluster. Subsequently, in case a fitter assignment is found, updated individual is attached to the current population. The process for the second neighborhood search is explained in Algorithm 11. Moreover, effects of the proposed neighborhood search on the CCBFS-H algorithm is tested in this study. Experimental results of these tests are presented in Section 7.2.

6.2.7 Selection of the Next Generation

The selection of the next generation concept of the CCBFS-H algorithm is same as in the CCBFS algorithm which is explained in Chapter 4 and in Algorithm 7. The aim of this selection rule is to improve poor parents and avoid premature convergence.

CHAPTER 7

COMPUTATIONAL RESULTS OF CCBFS-H

In this Chapter, computational results of the CCBFS-H algorithm will be explained. Simulated data set creation for high-dimensional data is explained in Section 7.1. After that, utilized performance measures and parameter settings for this algorithm is summarized in Section 7.2. Subsequently, computational results from the simulated data sets are presented in Section 7.3. Finally, computational results of the proposed method on a benchmark data set is reported in Section 7.4.

7.1 Simulated Data Sets

In order to analyze results of the CCBFS-H algorithm, we generate simulated high dimensional data sets. Other features of utilized data set are highly similar with the data sets in Section 5.1. The prominent features of the data sets that are utilized in section are as follows:

- Each instance in the data set is created either in different size or includes different selected features.
- All problem instances include q relevant and $m - q$ redundant features for each cluster.
- Values of relevant features are created from normal distribution, and uniform distribution is utilized for redundant features.

In this Chapter, 9 data instances are created in order to test CCBFS-H algorithm. Three different values are taken for both number of features, m, and number of relevant features, q. It is important to note that all columns are standardized in all data

instances as explained in Section 5.1. Details of the data set utilized in this Chapter is summarized in the Table 7.1.

Table 7.1: Details of the Simulated Data Sets

Number of Clusters (p)	2
Number of Data Points (N)	100
Number of Features (m)	{100, 300, 500}
Number of Relevant Features (q)	{10, 30, 50}

7.2 Performance Measures and Parameter Settings

Performance Measures:

In order to evaluate the results, we utilise same performance measures explained in Chapter 5 (Best vs. Z^c, Worst vs. Best, # of Best, # of Worst, CPU Time, and correct selection of features). Moreover, we add another performance measure of ARI in order to compare partitionings between CCBFS-H algorithm and simulated data sets.

Adjusted Rand Index (ARI):

Adjusted Rand Index (ARI) [23] is a widely utilized metric to evaluate unsupervised clustering problems which measures similarity between two partitioning. The ARI is developed over Rand Index (RI) [36], and the idea behind both of the indexes is to reward correct partitioning of data point pairs and penalize the wrong partitions. The possible occurrences for pairs are shown in a contingency matrix, Table 7.2.

Table 7.2: Contingency Matrix for Partitions A and B

	B	
A	Pairs in the same group	Pairs in different groups
Pairs in the same group	a	b
Pairs in different groups	c	d

RI can be calculated according to Equation 7.1. In this equation, a means number of pairs that are in the same cluster in both partitioning, b means number of pairs that are in different clusters in both partitioning, and n is the total number of data points.

$$RI = \frac{a + d}{a + b + c + d} \tag{7.1}$$

ARI is developed over RI, since RI may find successful results in some occasions by chance. Hence, an expected index value is utilized as a benchmark in ARI in order to consider random partitioning. Its upper bound is 1, which means compared partitionings are perfectly matched. On the other hand, negative values can be obtained as a lower bound which means that suggested clusters may have worse results than a random partitioning.

Formulation of the ARI is given in Equation 7.2 [46].

$$ARI = \frac{\binom{n}{2}(a+d) - [(a+b)(a+c)+(c+d)(b+d)]}{\binom{n}{2}^2 - [(a+b)(a+c)+(c+d)(b+d)]} \tag{7.2}$$

Parameter Settings:

For the parameter settings, we also followed a similar structure with Chapter 5. However, in line with the differences between two algorithms, we also need to make some alterations on parameter settings which is summarized in Table 7.3.

As we have one type of mutation in CCBFS-H algorithm, mutation probability is decreased to 1%. Proposed neighborhood search algorithm in CCBFS-H is another reason to decrease mutation probability, since it also provides searching capabilities on assignment.

Lastly, the number of candidates for neighborhood search is also changed as we are working with high-dimensional data and number of data points are limited.

We have tested the effects of second neighborhood search for CCBFS-H algorithm and report the results in Table B.2. In terms of best solutions any difference is not observed, but second neighborhood search increases the quality of results in the worst

solutions. Moreover, number of best results in the tests with no neighborhood search are always lower than or equal to the CCBFS-H algorithm. Therefore, we have applied the second neighborhood search in all of the tests in Chapters 7.3 and 7.4.

Table 7.3: Parameter Setting of the CCBFS-H Algorithm

Population Size	1000
Number of Generations	200
Ratio of Initial Generation by Type 1, 2 and 3	{50%, 25% , 25%}
Crossover Probability	100%
Mutation Probability	1%
Number of Candidates for Neighborhood Search 1	n/20
Number of Candidates for Neighborhood Search 2	n/50

7.3 Computational Results for Simulated Data Sets

Although CCBFS algorithm has provided successful results in the data sets with high number of points, it is not convenient for high-dimensional data sets. Therefore, CCBFS-H algorithm has developed in order to cluster such data sets as explained in Chapter 6. Similar to CCBFS algorithm, CCBFS-H algorithm is coded in MATLAB R2021a and simulations to test the algorithm has been carried out on 64-bit Windows 10 PC with 3.6 GHz 12 core Intel Xeon E-2246G processor and 16 GB RAM.

Öz (2019) [35] has proposed three linearized models and two heuristic methods. On simulated high dimensional data sets all of these proposed models do not provide clustering due to high requirement on memory.

Therefore, to test the performance of CCBFS-H algorithm, 9 synthetic data sets that are described in Section 7.1 and one benchmark data set is utilized. For the simulations, parameter settings explained in Section 7.2 is utilized. Detailed results for each simulation are provided in Table 7.4.

The table is constructed as in the Chapter 5. First column includes the information of parameters, p, n, m and q and Z^c value is given in the second column. Remaining performance metrics are provided in the following columns.

Table 7.4: Results of CCBFS-H Algorithm on High Dimensional Data

p,n,m,q	Z^c	**Best**	**Median**	**Worst**	**Best vs.** Z^c	**Worst vs. Best**	**# of Best**	**# of Worst**	**ARI**	**Correctly Selected Features**	**CPU (s)**
2,100,100,10	95.78	95.55	95.55	95.55	-0.2%	0.0%	50	0	1.00	[10,9]	45.19
2,100,100,30	267.78	259.17	259.17	259.17	-3.2%	0.0%	50	0	1.00	[27,30]	38.15
2,100,100,50	473.89	472.06	472.06	472.06	-0.4%	0.0%	50	0	1.00	[48,50]	46.64
2,100,300,10	79.92	79.92	79.92	80.78	0.0%	1.1%	43	7	1.00	[10,10]	92.23
2,100,300,30	270.38	263.71	263.71	265.92	-2.5%	0.8%	45	2	1.00	[26,29]	81.54
2,100,300,50	456.18	441.19	441.19	441.19	-3.3%	0.0%	50	0	1.00	[45,46]	91.67
2,100,500,10	81.42	79.59	79.59	82.35	-2.2%	3.5%	33	7	1.00	[9,10]	120.55
2,100,500,30	262.32	260.68	260.68	260.68	-0.6%	0.0%	50	0	1.00	[29,29]	110.92
2,100,500,50	457.16	436.70	436.70	447.41	-4.5%	2.5%	36	1	1.00	[45,47]	110.23

In eight of the nine simulated instances results of CCBFS-H algorithm are less than Z^c values, while it obtains exactly the same clusters in one instance. In these eight instances, CCBFS-H algorithm assigns data points to clusters as in the data simulation. But, it obtains lower objective function by differentiating some of the features which provides denser clusters than the ones created in data simulation phase. It should be noted that, most of the selected features are still same with the data simulation phase, and that is an expected outcome from a successful algorithm, since data is created according to specific distribution functions.

CCBFS-H algorithm also provides small differences between best and worst solutions like CCBFS. As shown in Table 7.4 the difference is at most 3.5% among nine instances while the best and worst results are the same for 5 instances. Moreover, median value is the same with the best results in all nine instances.

In terms of CPU times, it is observed that number of features, m, has main effect on duration, while no significant variance is observed on different values of number of selected features, q. During the memetic algorithm, features are ordered and fittest ones are selected for each offspring. Therefore varying number of features directly affect the computational time, although solution space is the same for different m values in CCBFS-H.

Effects of Selecting Different Number of Features

In the previous problem setting, number of selected features are given, however in real life data sets, gathering data for each feature might be costly. As an example, we have tested q values up to 50 in the simulated data sets, however the algorithm might obtain the same clustering by selecting much less features. Considering that, we have studied simulated data sets by selecting different number of features with respect to their creation. We analyze the results in accordance with the following questions for different values of q.

- Does CCBFS-H able to assign data points to clusters similar to simulated data?
- Does CCBFS-H able to deduce relevant features similar to simulated data?

For the first question, ARI value is analyzed since higher ARI value represents higher similarity between two assignments. Number of common features is considered for

the second question. By looking at the common features we aim to monitor whether the CCBFS-H algorithm is able to make correct assignment with lower number of features, or not.

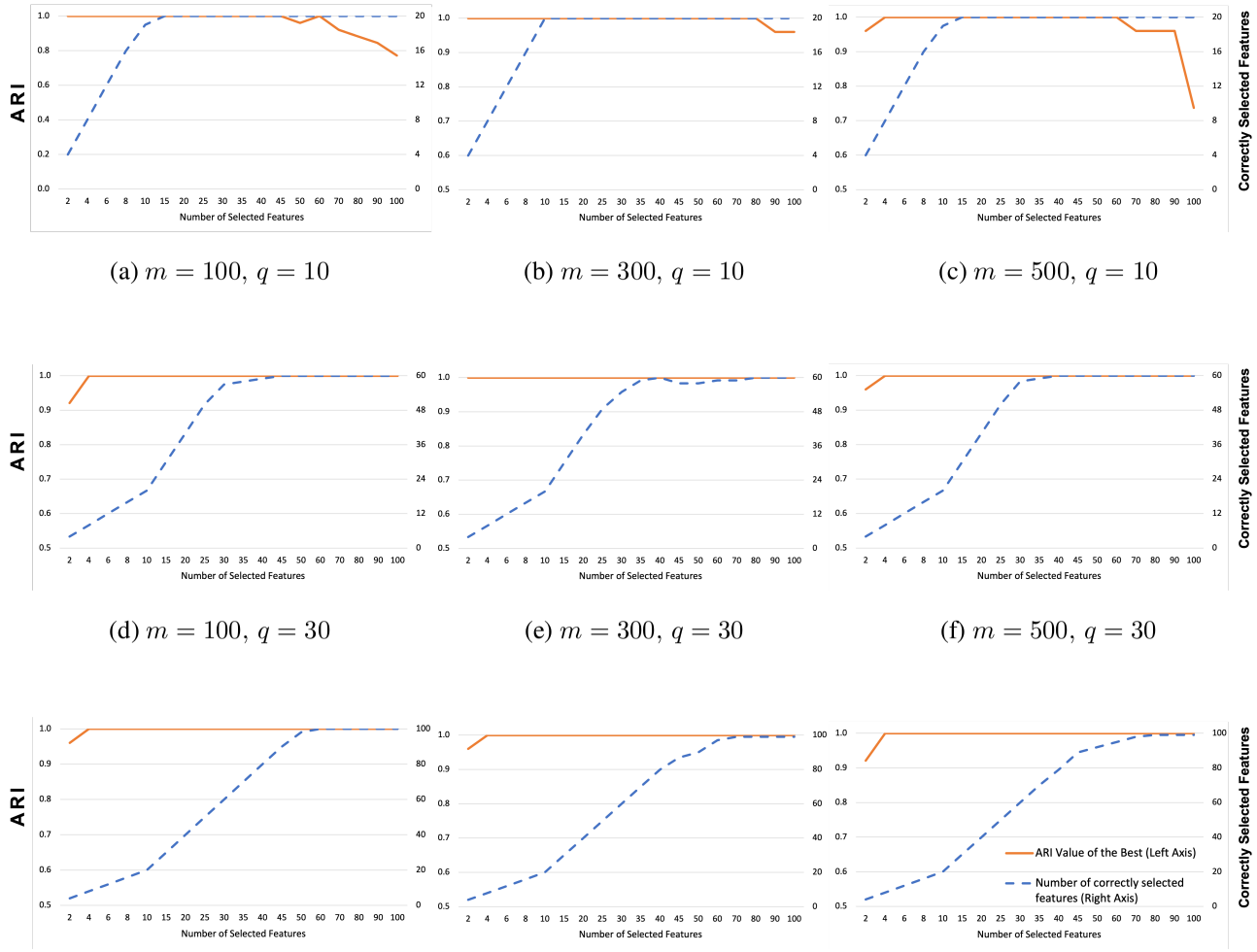
Detailed results for each simulation is given in the Appendix, in Tables A.1 - A.3. Here q refers the number of features that have normal distribution during data creation and Z^c calculation stage, and sq represents the number of selected features that are given to the algorithm. Best, median and worst results among 50 trials, percentage gap between best result and worst result, number of best and worst solutions, CPU times, adjusted rand index of the best result, and correctly selected features of the best result are provided in these tables.

Similar to previous simulations, difference between best and worst results remains low for variable number of features. The average difference is less than 1%, while the difference figures are always less than 5.3%. It is also observed that marginal increase of the objective function increases in line with number of selected features. It is an expected outcome since the algorithm initially selects the least cost features.

To illustrate the effects of different number of selected features on ARI and correct selection of features, Figure 7.1 is provided. As shown in all graphs, ARI values equal to 1 in most of the times which means CCBFS-H algorithm makes the same clustering with simulated data. Moreover, maximum value of ARI is reached by selecting less than 6 features in all instances. Therefore, to cluster a data set correctly, the CCBFS-H algorithm do not require to process all of the relevant features, and it is able to cluster them correctly with much less features.

A decline is observed on ARI when the number of selected features is high and $q = 10$. It is also an anticipated outcome since there are much more redundant features than relevant features in these instances. Here, CCBFS-H algorithm finds another relationship between redundant features so that the assignments are differentiated. Although the interval is very large in these examples, we can conclude that selection of correct number of features has significant effects on clustering problems.

On the other hand, as shown in Figure 7.1, correctly selected features around double of number of selected features which is the maximum value since $p = 2$. It means



(a) $m = 100,\ q = 10$ (b) $m = 300,\ q = 10$ (c) $m = 500,\ q = 10$

(d) $m = 100,\ q = 30$ (e) $m = 300,\ q = 30$ (f) $m = 500,\ q = 30$

(g) $m = 100,\ q = 50$ (h) $m = 300,\ q = 50$ (i) $m = 500,\ q = 50$

Figure 7.1: Results of ARI and Correctly Selected Features for Different Number of Selected Features, $p = 2,\ n = 100$

that, CCBFS-H algorithm selects the features that are created as relevant features during data creation. Therefore, CCBFS-H algorithm is also successful on finding the relevant features in data set for a clustering problem.

7.4 Computational Results for Benchmark Data Sets

In addition to synthetic data sets, we also test the capabilities of CCBFS-H algorithm on the Wisconsin Diagnostic Breast Cancer data set. This data includes 30 features and 569 data points with no missing values. There are two classes in this data set, namely, benign and malignant. Among 569 observations, 357 of them are labeled as benign and remaining 212 of them are malignant.

In this section we have two targets. Testing the success of CCBFS-H algorithm on a benchmark data set is taken as a first target. Another target is to show the effectiveness of cluster based feature selection over a clustering with feature selection by applying the same methodology. We set the second target since data sets in bio-informatics domain includes high number of features and data points in these data sets might be grouped by different sets of relevant features. Therefore, CCBFS-H algorithm might provide better results by utilizing much less features. In order to obtain results from clustering with feature selection, we altered the CCBFS-H algorithm to select same features for all clusters.

The results are summarized in Table 7.5. In this table, number of correctly classified data points for different values of number of selected features, q, is reported. Results of best and worst solutions and the averages of 50 trials for both algorithms are provided. To illustrate the developments of clustering results of both algorithms on q values, Figure 7.2 is presented.

As reported in Table 7.5, considering the best solutions, CCBFS-H is able to classify data points correctly by more than 90%. Correctly classified data points in CCBFS-H is lower for q values of 2 and 6. For the q values of 10 and above the results reaches a maturity on best solutions and a significant increase is not observed. Furthermore, a notable difference on worst solutions is not observed for these q values. Moreover, averages of 50 trials is increasing with q in almost all instances. It reaches peak when

Table 7.5: Correctly Classified Data Points of Best, Worst and Average Solutions on CCBFS-H and Clustering with Feature Selection Algorithms

	CCBFS-H			**Clustering with Feature Selection**		
q	Best	Worst	Average	Best	Worst	Average
2	524	448	493.6	470	454	458.9
6	529	480	507.1	488	473	478.0
10	538	497	517.7	500	483	497.0
14	541	507	523.7	512	483	498.8
18	538	501	522.8	517	482	504.9
22	536	517	528.0	534	506	514.1
26	536	505	528.2	536	514	527.3
30	541	511	528.6	541	511	528.6

all 30 features are utilized, although difference in averages with $q = 22$ is only 0.1%.

Considering above-mentioned arguments, CCBFS-H algorithm ensures reliable results on the benchmark data set. Moreover, after a certain value of selected features q, the algorithm creates clusters that ensures high number of correctly classified data points. It is a remarkable outcome since collecting data for all features might be costly in some circumstances.

When Figure 7.2 is analyzed, it is observed that there is a significant difference between two methods for q values below 20. It means that cluster specific feature selection is more successful than selection of same features for each clusters. The difference is higher for $q = 2$, and it is getting smaller in line with increasing values of q. After a certain value of $q = 22$, simulation results became indifferent, so that similar results can be obtained in both methods for higher values of q.

We also analyze the selected features of both algorithms for different values of selected features, q. Selected features of the best results among 50 trials are summarized in Table 7.6 for q values between 2 and 18. It is observed from the table that selected features in clustering with feature selection (CFS) problem are always selected by

one of the clusters in CCBFS-H. Since CCBFS-H algorithm provides better results, it can be posited that CCBFS-H is able to find same results with CFS, but it can obtain better results by differentiating feature sets for clusters. Moreover, it is observed that selected features in lower values of q are always selected in the increasing values of q. It is an expected outcome, since rectilinear distance is utilized so that the algorithm initially select features which provide minimum within-cluster distances. When q is increased, the algorithm selects new features with same rationale.

Table 7.6: Features selected by CCBFS-H and clustering with feature selection algorithms

q	**Algorithm**	**Cluster**	**Selected Features**
2	CCBFS-H	Cluster 1	{13 14}
		Cluster 2	{17 20}
	CFS	All Clusters	{13 14}
6	CCBFS-H	Cluster 1	{4 11 13 14 17 24}
		Cluster 2	{13 14 15 17 18 20}
	CFS	All Clusters	{11 13 14 17 20 24}
10	CCBFS-H	Cluster 1	{1 3 4 11 13 14 17 21 23 24}
		Cluster 2	{11 12 13 14 15 16 17 18 19 20}
	CFS	All Clusters	{4 11 13 14 15 17 20 21 23 24}
14	CCBFS-H	Cluster 1	{1 3 4 7 8 11 13 14 17 18 20 21 23 24}
		Cluster 2	{2 5 11 12 13 14 15 16 17 18 19 20 29 30}
	CFS	All Clusters	{1 3 4 11 13 14 15 17 18 20 21 23 24 30}
18	CCBFS-H	Cluster 1	{1 3 4 6 7 8 11 13 14 17 18 20 21 23 24 26 27 30}
		Cluster 2	{2 5 6 9 11 12 13 14 15 16 17 18 19 20 25 26 29 30}
	CFS	All Clusters	{1 3 4 8 11 13 14 15 16 17 18 19 20 21 23 24 29 30}

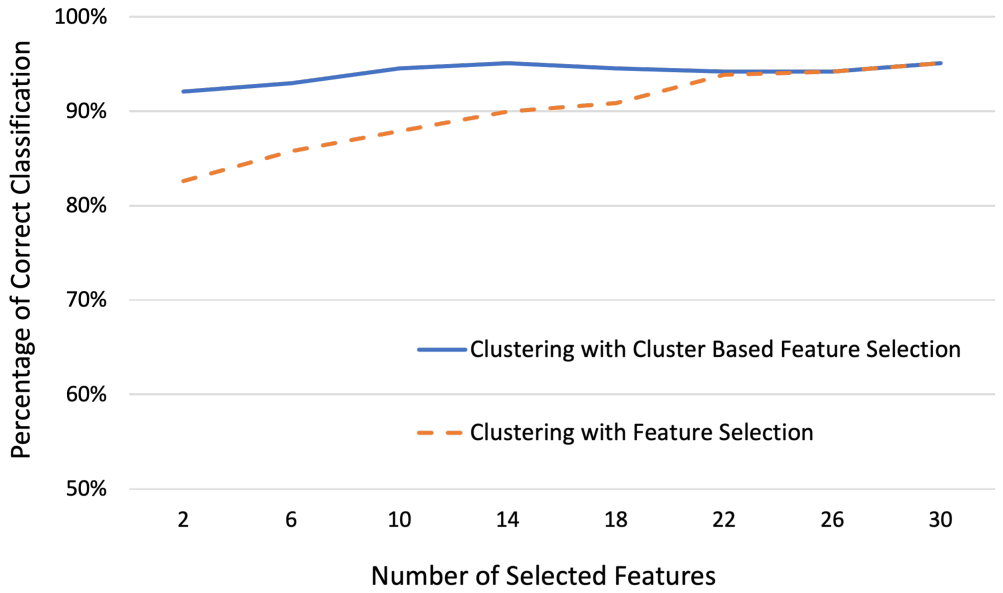


Figure 7.2: Comparison of CCBFS-H and Clustering with Feature Selection on Correct Classification

CHAPTER 8

CONCLUSIONS

In this thesis, we study clustering with hard partitioning and we focus on a clustering with cluster based feature selection problem. In the problem setting, each cluster includes one data point as a cluster centroid and each data point must be assigned to exactly one cluster. Moreover, relevant features are selected for each cluster separately.

Dimensions of the data set determine the number of features and number of data points. The information of number of clusters and number of features that should be selected are determined in advance, similar to other partitional clustering methods. In the problem setting, hard partitioning is applied by assigning each data point to exactly one cluster, so that disjoint clusters are constructed. As a similarity measure between data points, we utilize rectilinear distance. Since pre-determined number of features are selected, distance between a data point to a cluster centroid is calculated with respect to selected features.

We represent the problem as a nonlinear mixed integer mathematical model. However, it is an NP-Hard problem and the mathematical model can not obtain convenient results in meaningful durations, especially for bigger data sets. In this context, we have developed a memetic algorithm to find successful results in shorter durations.

Objective of the first memetic algorithm is finding the clustering with least within-cluster distance by using the information of feature selection and cluster centroids. Since assignment of data points become obvious when the centroids and selected features are known, the assignments are not stored in chromosome representation. Thanks to the structure of chromosomes, problem space in the CCBFS algorithm is

reduced significantly.

We have tested the proposed algorithm with various number of features, data points and clusters. The results show that CCBFS algorithm is able to obtain successful clustering results in all cases. Moreover, differences between best and worst solutions among all trials are quite low so that the algorithm is able to obtain reliable results. The algorithm also hits its worst solution in less than 4% of the instances, in average. Therefore, it is able to quit from the solutions which might be local optima.

Although CCBFS algorithm obtains convenient results on the data sets with high number of objects, it has deficiencies on the high dimensional data sets. Therefore, we developed a novel CCBFS-H algorithm by updating the first algorithm, which has a focus on finding successful clustering results on high dimensional data. The most fundamental change in the updated algorithm is chromosome representation of the problem. CCBFS-H algorithm stores assignment and centroid information in the chromosomes instead of selected features. Due to this structure, increasing number of features does not affect the problem space harshly and the proposed algorithm is able to obtain results within reasonable durations.

CCBFS-H algorithm is tested on both synthetic and a benchmark data sets. Results of the CCBFS-H algorithm are also successful on within-cluster distance of the best solutions. Furthermore, differences between the best and worst solutions in CCBFS-H are around 1% in average. In addition to obtaining convenient clustering, the algorithm is also successful on finding relevant features which is important to obtain meaningful information from the data set. We also found that CCBFS-H algorithm can create same clusters by using lower number of features.

This is the first study that developed memetic algorithm approach on clustering with cluster based feature selection problem. Since any feature can be selected in any of the clusters, our proposed method includes the clustering with feature selection problem. We test the advantage of cluster based feature selection on the benchmark data set and we have found that it obtains more successful results than fixed feature selection problem, especially when the number of selected features are low.

In this study, rectilinear distance is utilized as a similarity measure, but different dis-

tance measures can be utilized to test both of the algorithms as a future research direction.

Objectives of the proposed algorithms are to minimize within-cluster distances. They can also be tested on different objectives such as maximization of inter-cluster distances.

Finally, our proposed algorithm can be updated to solve k-means clustering. If the information of selected features and assignment of data points stored in chromosome representation, cluster centers can be determined as the average of assigned data points which can be transformed into k-means clustering.

REFERENCES

[1] Ö. Akay, E. Tekeli, and G. Yüksel. Genetic algorithm with new fitness function for clustering. *Iranian Journal of Science and Technology, Transactions A: Science*, 44:865–874, 2020.

[2] S. Alelyani, J. Tang, and H. Liu. Feature selection for clustering: A review. *Data Clustering*, pages 29–60, 2018.

[3] A. H. Beg and M. Z. Islam. Clustering by genetic algorithm-high quality chromosome selection for initial population. In *2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA)*, pages 129–134. IEEE, 2015.

[4] A. Ben-Israel and C. Iyigun. Probabilistic d-clustering. *Journal of Classification*, 25(1):5–26, 2008.

[5] S. Benati and S. García. A mixed integer linear model for clustering with variable selection. *Computers & operations research*, 43:280–285, 2014.

[6] S. Benati, S. García, and J. Puerto. Mixed integer linear programming and heuristic methods for feature selection in clustering. *Journal of the Operational Research Society*, 69(9):1379–1395, 2018.

[7] P. Berkhin. A survey of clustering data mining techniques. In *Grouping multidimensional data*, pages 25–71. Springer, 2006.

[8] J. Bezdek, R. Enrlich, and W. Full. Fcm: The fuzzy c-means clustering algorithm. computers&geosciences, volume 10, issue 2-3, 1984.

[9] M. J. Brusco. Clustering binary data in the presence of masking variables. *Psychological Methods*, 9(4):510, 2004.

[10] D. Chen, K. C. Chan, and X. Wu. Gene expression analyses using genetic algorithm based hybrid approaches. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 963–969. IEEE, 2008.

[11] F. Chiyoshi and R. D. Galvão. A statistical analysis of simulated annealing applied to the p-median problem. *Annals of Operations Research*, 96(1):61–74, 2000.

[12] J. E. Corter and M. A. Gluck. Explaining basic categories: Feature predictability and information. *Psychological bulletin*, 111(2):291, 1992.

[13] J. G. Dy and C. E. Brodley. Feature selection for unsupervised learning. *Journal of machine learning research*, 5(Aug):845–889, 2004.

[14] A. E. Ezugwu, A. K. Shukla, M. B. Agbaje, O. N. Oyelade, A. José-García, and J. O. Agushaka. Automatic clustering algorithms: a systematic review and bibliometric analysis of relevant literature. *Neural Computing and Applications*, 33(11):6247–6306, 2021.

[15] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine learning*, 2(2):139–172, 1987.

[16] E. B. Fowlkes and C. L. Mallows. A method for comparing two hierarchical clusterings. *Journal of the American statistical association*, 78(383):553–569, 1983.

[17] H. Frigui and O. Nasraoui. Unsupervised learning of prototypes and attribute weights. *Pattern recognition*, 37(3):567–581, 2004.

[18] S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. *ACM Sigmod record*, 27(2):73–84, 1998.

[19] E. Hancer, B. Xue, and M. Zhang. A survey on feature selection approaches for clustering. *Artificial Intelligence Review*, 53(6):4519–4545, 2020.

[20] P. Hansen and N. Mladenović. Variable neighborhood search for the p-median. *Location Science*, 5(4):207–226, 1997.

[21] P. Hansen, N. Mladenović, R. Todosijević, and S. Hanafi. Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, 5(3):423–454, 2017.

[22] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[23] L. Hubert and P. Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[24] P. Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bull Soc Vaudoise Sci Nat*, 37:547–579, 1901.

[25] I. T. Jolliffe and J. Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.

[26] G. Karypis, E.-H. Han, and V. Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, 1999.

[27] Y. Kim, W. N. Street, and F. Menczer. Evolutionary model selection in unsupervised learning. *Intelligent data analysis*, 6(6):531–556, 2002.

[28] Z. F. Knops, J. A. Maintz, M. A. Viergever, and J. P. Pluim. Normalized mutual information based registration using k-means clustering and shading correction. *Medical image analysis*, 10(3):432–439, 2006.

[29] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[30] M. H. Law, M. A. Figueiredo, and A. K. Jain. Simultaneous feature selection and clustering using mixture models. *IEEE transactions on pattern analysis and machine intelligence*, 26(9):1154–1166, 2004.

[31] T. V. Levanova and M. Loresh. Algorithms of ant system and simulated annealing for the p-median problem. *Automation and remote control*, 65(3):431–438, 2004.

[32] Y. Li, M. Dong, and J. Hua. Localized feature selection for clustering. *Pattern Recognition Letters*, 29(1):10–18, 2008.

[33] J. MacQueen. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pages 281–297, 1967.

[34] N. Mladenović, J. Brimberg, P. Hansen, and J. A. Moreno-Pérez. The p-median problem: A survey of metaheuristic approaches. *European Journal of Operational Research*, 179(3):927–939, 2007.

[35] S. Önen Öz. Mixed integer programming and heuristics approachesfor clustering with cluster-based feature selection. Master's thesis, 2019.

[36] W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

[37] L. Rdusseeun and P. Kaufman. Clustering by means of medoids. In *Proceedings of the statistical data analysis based on the L1 norm conference, neuchatel, switzerland*, volume 31, 1987.

[38] C. K. Reddy and B. Vinzamuri. A survey of partitional and hierarchical clustering algorithms. In *Data clustering*, pages 87–110. Chapman and Hall/CRC, 2018.

[39] L. Rokach and O. Maimon. Clustering methods data mining and knowledge discovery handbook (pp. 321-352), 2005.

[40] E. Rolland, D. A. Schilling, and J. R. Current. An efficient tabu search procedure for the p-median problem. *European Journal of Operational Research*, 96(2):329–342, 1997.

[41] M. Rostami and P. Moradi. A clustering based genetic algorithm for feature selection. In *2014 6th Conference on Information and Knowledge Technology (IKT)*, pages 112–116. IEEE, 2014.

[42] A. Saxena, M. Prasad, A. Gupta, N. Bharill, O. P. Patel, A. Tiwari, M. J. Er, W. Ding, and C.-T. Lin. A review of clustering techniques and developments. *Neurocomputing*, 267:664–681, 2017.

[43] W. Sheng and X. Liu. A genetic k-medoids clustering algorithm. *Journal of Heuristics*, 12(6):447–466, 2006.

[44] L. Shi and S. Olafsson. Nested partitions method for global optimization. *Operations research*, 48(3):390–407, 2000.

[45] S. Solorio-Fernández, J. A. Carrasco-Ochoa, and J. F. Martínez-Trinidad. A review of unsupervised feature selection methods. *Artificial Intelligence Review*, 53(2):907–948, 2020.

[46] D. Steinley. Properties of the hubert-arable adjusted rand index. *Psychological methods*, 9(3):386, 2004.

[47] M. Sun, L. Xiong, H. Sun, and D. Jiang. A ga-based feature selection for high-dimensional data clustering. In *2009 Third International Conference on Genetic and Evolutionary Computing*, pages 769–772. IEEE, 2009.

[48] J. Tang, S. Alelyani, and H. Liu. Feature selection for classification: A review. *Data classification: Algorithms and applications*, page 37, 2014.

[49] N. X. Vinh, J. Epps, and J. Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *The Journal of Machine Learning Research*, 11:2837–2854, 2010.

[50] J. Wang, Z. Cheng, O. K. Ersoy, P. Zhang, and W. Dai. Multi-offspring genetic algorithm with two-point crossover and the relationship between number of offsprings and computational speed. *Journal of Computers*, 30(5):111–127, 2019.

[51] Y.-L. Wu, C.-Y. Tang, M.-K. Hor, and P.-F. Wu. Feature selection using genetic algorithm and cluster validation. *Expert Systems with Applications*, 38(3):2727–2732, 2011.

[52] R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678, 2005.

[53] B. Zhang, M. Hsu, and U. Dayal. K-harmonic means-a spatial clustering algorithm with boosting. In *International Workshop on Temporal, Spatial, and Spatio-Temporal Data Mining*, pages 31–45. Springer, 2000.

[54] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. *ACM sigmod record*, 25(2):103–114, 1996.

Appendix A

EXPERIMENTAL RESULTS OF THE CCBFS-H ALGORITHM

Simulation Results of CCBFS-H Algorithm

Table A.1: Simulation Results of CCBFS-H Algorithm with n = 100, p = 2 and m = 100

q	sq	Best	Median	Worst	Worst vs. Best	# of Best	# of Worst	ARI	Correctly Selected Features	CPU (s)
10	2	9.11	9.11	9.16	0.6%	36	7	1.00	[2,2]	42.41
10	4	23.02	23.02	23.27	1.1%	41	9	1.00	[4,4]	49.03
10	6	42.99	42.99	43.19	0.5%	39	11	1.00	[6,6]	45.43
10	8	65.63	65.63	66.77	1.7%	39	1	1.00	[8,8]	45.20
10	10	95.55	95.55	95.55	0.0%	50	50	1.00	[10,9]	45.19
10	15	187.22	187.22	189.76	1.4%	42	1	1.00	[10,10]	47.21
10	20	299.15	299.15	302.70	1.2%	41	3	1.00	[10,10]	45.79
10	25	419.04	419.04	421.99	0.7%	38	3	1.00	[10,10]	45.36
10	30	543.51	543.51	547.87	0.8%	37	1	1.00	[10,10]	45.15
10	35	670.53	670.53	673.42	0.4%	42	8	1.00	[10,10]	44.97
10	40	801.71	801.71	806.70	0.6%	38	1	1.00	[10,10]	45.34
10	45	936.92	936.92	941.58	0.5%	43	7	1.00	[10,10]	45.48
10	50	1078.08	1078.08	1084.09	0.6%	47	1	0.96	[10,10]	45.28
10	60	1372.67	1372.67	1375.48	0.2%	35	1	1.00	[10,10]	44.30
10	70	1689.65	1689.65	1690.10	0.0%	37	1	0.92	[10,10]	49.79
10	80	2025.12	2025.12	2028.52	0.2%	38	3	0.88	[10,10]	45.25
10	90	2400.83	2400.83	2402.86	0.1%	37	1	0.84	[10,10]	45.51
10	100	2856.65	2856.65	2865.78	0.3%	30	1	0.77	[10,10]	45.70
30	2	6.84	6.84	6.85	0.3%	10	4	0.92	[2,2]	39.79
30	4	14.54	14.54	14.68	1.0%	40	10	1.00	[4,4]	37.96
30	6	25.22	25.22	25.22	0.0%	50	50	1.00	[6,6]	37.76
30	8	36.86	36.86	36.86	0.0%	50	50	1.00	[8,8]	38.00
30	10	51.11	51.11	51.37	0.5%	48	2	1.00	[10,10]	38.29
30	15	90.56	90.56	90.56	0.0%	50	50	1.00	[15,15]	39.32
30	20	137.40	137.40	137.40	0.0%	50	50	1.00	[20,20]	40.87
30	25	192.08	192.08	192.08	0.0%	50	50	1.00	[25,25]	38.23
30	30	259.17	259.17	259.17	0.0%	50	50	1.00	[27,30]	38.15
30	35	347.81	347.81	347.81	0.0%	50	50	1.00	[28,30]	38.44
30	40	455.18	455.18	455.18	0.0%	50	50	1.00	[29,30]	38.40
30	45	571.01	571.01	571.01	0.0%	50	50	1.00	[30,30]	38.43
30	50	695.20	695.20	695.20	0.0%	50	50	1.00	[30,30]	38.24
30	60	957.15	957.15	957.15	0.0%	50	50	1.00	[30,30]	43.26
30	70	1242.72	1242.72	1242.72	0.0%	50	50	1.00	[30,30]	44.26
30	80	1544.79	1544.79	1544.79	0.0%	50	50	1.00	[30,30]	41.59
30	90	1882.87	1882.87	1882.87	0.0%	50	50	1.00	[30,30]	38.25
30	100	2322.74	2322.74	2322.74	0.0%	50	50	1.00	[30,30]	38.45

Table A.1 continued from previous page

q	sq	Best	Median	Worst	Worst vs. Best	# of Best	# of Worst	ARI	Correctly Selected Features	CPU (s)
50	2	7.32	7.32	7.33	0.1%	28	22	0.96	[2,2]	38.83
50	4	15.59	15.59	15.60	0.1%	44	6	1.00	[4,4]	38.07
50	6	25.91	25.91	26.00	0.4%	49	1	1.00	[6,6]	38.00
50	8	37.23	37.23	37.23	0.0%	50	50	1.00	[8,8]	37.89
50	10	49.45	49.45	49.45	0.0%	50	50	1.00	[10,10]	38.06
50	15	85.62	85.62	85.62	0.0%	50	50	1.00	[15,15]	49.36
50	20	126.17	126.17	126.67	0.4%	49	1	1.00	[20,20]	50.74
50	25	170.20	170.20	170.20	0.0%	50	50	1.00	[25,25]	46.83
50	30	216.45	216.45	216.45	0.0%	50	50	1.00	[30,30]	47.45
50	35	268.63	268.63	268.63	0.0%	50	50	1.00	[35,35]	47.32
50	40	330.01	330.01	330.01	0.0%	50	50	1.00	[40,40]	46.89
50	45	393.63	393.63	393.63	0.0%	50	50	1.00	[45,45]	46.44
50	50	472.06	472.06	472.06	0.0%	50	50	1.00	[49,49]	46.64
50	60	671.16	671.16	671.16	0.0%	50	50	1.00	[50,50]	47.55
50	70	917.12	917.12	917.12	0.0%	50	50	1.00	[50,50]	47.03
50	80	1186.01	1186.01	1186.01	0.0%	50	50	1.00	[50,50]	47.48
50	90	1505.55	1505.55	1505.55	0.0%	50	50	1.00	[50,50]	47.74
50	100	1919.66	1919.66	1919.66	0.0%	50	50	1.00	[50,50]	47.58

Table A.2: Simulation Results of CCBFS-H Algorithm with $n = 100, p = 2$ and $m = 300$

q	sq	Best	Median	Worst	Worst vs. Best	# of Best	# of Worst	ARI	Correctly Selected Features	CPU (s)
10	2	7.97	7.97	7.98	0.2%	48	2	1.00	[2,2]	95.89
10	4	19.09	19.09	19.09	0.0%	50	50	1.00	[4,4]	88.18
10	6	35.44	35.44	35.55	0.3%	39	11	1.00	[6,6]	87.72
10	8	55.68	55.68	56.01	0.6%	42	8	1.00	[8,8]	87.88
10	10	79.92	79.92	80.78	1.1%	43	7	1.00	[10,10]	92.23
10	15	176.33	176.33	177.27	0.5%	35	1	1.00	[10,10]	88.02
10	20	281.79	281.79	286.08	1.5%	40	1	1.00	[10,10]	87.94
10	25	394.11	394.11	397.42	0.8%	43	7	1.00	[10,10]	88.30
10	30	510.26	510.26	512.76	0.5%	29	2	1.00	[10,10]	88.79
10	35	628.02	628.02	629.58	0.2%	39	2	1.00	[10,10]	88.79
10	40	747.73	747.73	747.74	0.0%	44	6	1.00	[10,10]	88.98
10	45	867.66	867.66	874.38	0.8%	41	1	1.00	[10,10]	88.87
10	50	989.18	989.18	990.84	0.2%	45	5	1.00	[10,10]	88.35
10	60	1237.16	1237.16	1240.33	0.3%	49	1	1.00	[10,10]	88.77
10	70	1490.94	1490.94	1506.89	1.1%	45	1	1.00	[10,10]	88.90
10	80	1749.73	1749.73	1751.47	0.1%	47	2	1.00	[10,10]	89.16
10	90	2014.65	2014.65	2026.27	0.6%	40	1	0.96	[10,10]	88.90
10	100	2283.12	2283.12	2291.85	0.4%	36	1	0.96	[10,10]	89.35
10	200	5279.29	5279.63	5303.98	0.5%	17	1	0.88	[10,10]	85.33
10	300	9364.73	9368.05	9395.70	0.3%	9	2	0.33	[10,10]	82.44
30	2	7.25	7.25	7.36	1.5%	43	3	1.00	[2,2]	85.79
30	4	17.17	17.17	17.43	1.6%	37	1	1.00	[4,4]	76.61
30	6	28.11	28.11	28.13	0.1%	39	2	1.00	[6,6]	76.36
30	8	39.37	39.37	39.51	0.4%	40	10	1.00	[8,8]	76.56
30	10	51.32	51.32	51.32	0.0%	50	50	1.00	[10,10]	77.67
30	15	87.20	87.20	87.26	0.1%	45	1	1.00	[15,15]	77.66
30	20	134.15	134.15	135.18	0.8%	44	1	1.00	[20,20]	77.44
30	25	192.71	192.71	193.79	0.6%	41	1	1.00	[24,25]	77.74
30	30	263.71	263.71	265.92	0.8%	45	2	1.00	[26,29]	77.54
30	35	353.10	353.10	356.46	1.0%	41	1	1.00	[29,30]	77.56
30	40	455.34	455.34	458.96	0.8%	32	1	1.00	[30,30]	77.43

Table A.2 continued from previous page

q	sq	Best	Median	Worst	Worst vs. Best	# of Best	# of Worst	ARI	Correctly Selected Features	CPU (s)
30	45	564.86	564.86	564.86	0.0%	50	50	1.00	[29,29]	77.52
30	50	675.71	675.71	675.71	0.0%	50	50	1.00	[29,29]	77.68
30	60	905.17	905.17	911.25	0.7%	48	2	1.00	[29,30]	78.13
30	70	1143.21	1143.21	1148.45	0.5%	42	8	1.00	[29,30]	77.89
30	80	1387.90	1387.90	1394.54	0.5%	46	4	1.00	[30,30]	77.55
30	90	1638.98	1638.98	1645.18	0.4%	42	4	1.00	[30,30]	77.73
30	100	1895.99	1895.99	1901.88	0.3%	42	6	1.00	[30,30]	77.63
30	200	4779.86	4779.86	4779.86	0.0%	50	50	1.00	[30,30]	82.49
30	300	8868.46	8868.46	8868.46	0.0%	50	50	1.00	[30,30]	83.51
50	2	6.39	6.39	7.04	10.3%	48	1	0.96	[2,2]	88.49
50	4	13.85	13.85	14.00	1.0%	37	1	1.00	[4,4]	96.75
50	6	22.22	22.22	22.41	0.9%	44	6	1.00	[6,6]	91.04
50	8	31.03	31.03	31.23	0.7%	39	3	1.00	[8,8]	91.39
50	10	40.43	40.43	40.69	0.6%	49	1	1.00	[10,10]	95.28
50	15	68.54	68.54	69.48	1.4%	47	3	1.00	[15,15]	100.50
50	20	102.54	102.54	105.32	2.7%	43	1	1.00	[20,20]	94.08
50	25	142.99	142.99	144.95	1.4%	41	9	1.00	[25,25]	93.46
50	30	187.94	187.94	188.62	0.4%	42	1	1.00	[30,30]	91.88
50	35	236.81	236.81	237.57	0.3%	44	6	1.00	[35,35]	91.08
50	40	294.26	294.26	296.62	0.8%	42	2	1.00	[40,40]	90.97
50	45	362.87	362.87	362.87	0.0%	50	50	1.00	[43,44]	91.60
50	50	441.19	441.19	441.19	0.0%	50	50	1.00	[45,46]	91.67
50	60	617.37	617.37	623.05	0.9%	49	1	1.00	[48,49]	104.03
50	70	833.06	833.06	836.31	0.4%	49	1	1.00	[50,49]	107.37
50	80	1064.17	1064.17	1064.17	0.0%	50	50	1.00	[50,49]	101.04
50	90	1301.66	1301.66	1301.66	0.0%	50	50	1.00	[50,49]	102.14
50	100	1546.38	1546.38	1546.38	0.0%	50	50	1.00	[50,49]	105.12
50	200	4351.44	4351.44	4351.44	0.0%	50	50	1.00	[50,50]	107.75
50	300	8306.69	8306.69	8306.69	0.0%	50	50	1.00	[50,50]	92.44

Table A.3: Simulation Results of CCBFS-H Algorithm with $n = 100, p = 2$ and $m = 500$

q	sq	Best	Median	Worst	Worst vs. Best	# of Best	# of Worst	ARI	Correctly Selected Features	CPU (s)
10	2	10.05	10.07	10.47	4.2%	24	1	0.96	[2,2]	137.10
10	4	22.00	22.00	22.34	1.6%	30	1	1.00	[4,4]	126.44
10	6	35.33	35.33	36.42	3.1%	43	7	1.00	[6,6]	122.57
10	8	54.82	54.82	56.79	3.6%	47	1	1.00	[8,8]	119.89
10	10	79.59	79.59	82.35	3.5%	33	7	1.00	[9,10]	120.55
10	15	171.81	171.81	173.33	0.9%	39	4	1.00	[10,10]	127.11
10	20	279.43	279.43	281.76	0.8%	35	2	1.00	[10,10]	126.95
10	25	391.12	391.12	392.68	0.4%	48	2	1.00	[10,10]	122.58
10	30	503.61	503.61	504.89	0.3%	45	5	1.00	[10,10]	123.03
10	35	618.01	618.01	619.02	0.2%	45	5	1.00	[10,10]	122.62
10	40	733.72	733.72	735.87	0.3%	38	1	1.00	[10,10]	126.22
10	45	850.43	850.43	851.97	0.2%	37	13	1.00	[10,10]	120.61
10	50	968.80	968.80	973.17	0.5%	42	3	1.00	[10,10]	122.84
10	60	1209.28	1209.28	1220.25	0.9%	42	1	1.00	[10,10]	145.98
10	70	1452.28	1452.28	1463.68	0.8%	26	1	0.96	[10,10]	144.98
10	80	1698.21	1699.83	1708.93	0.6%	19	1	0.96	[10,10]	129.59
10	90	1948.00	1949.00	1956.01	0.4%	23	2	0.96	[10,10]	128.86
10	100	2200.21	2200.83	2214.08	0.6%	3	1	0.74	[10,10]	128.47
10	200	4870.59	4879.27	4898.03	0.6%	2	1	0.57	[10,10]	129.67
10	300	7863.57	7876.43	7912.12	0.6%	1	1	0.40	[10,10]	122.94

Table A.3 continued from previous page

q	sq	Best	Median	Worst	Worst vs. Best	# of Best	# of Worst	ARI	Correctly Selected Features	CPU (s)
10	400	11430.21	11450.94	11488.49	0.5%	1	1	0.43	[10,10]	122.55
10	500	15866.77	15877.03	15934.48	0.4%	6	1	-0.01	[10,10]	126.90
30	2	7.40	7.40	7.46	0.8%	41	3	0.96	[2,2]	108.65
30	4	15.59	15.59	15.61	0.1%	32	18	1.00	[4,4]	114.76
30	6	24.29	24.29	24.81	2.1%	38	3	1.00	[6,6]	110.78
30	8	36.10	36.10	36.47	1.0%	37	12	1.00	[8,8]	110.46
30	10	48.79	48.79	49.65	1.8%	40	3	1.00	[10,10]	110.67
30	15	89.70	89.70	90.23	0.6%	37	12	1.00	[15,15]	112.21
30	20	137.62	137.62	139.18	1.1%	34	1	1.00	[20,20]	114.27
30	25	194.83	194.83	194.83	0.0%	50	50	1.00	[25,25]	110.94
30	30	260.68	260.68	260.68	0.0%	50	50	1.00	[29,29]	110.92
30	35	347.03	347.03	349.24	0.6%	49	1	1.00	[29,30]	110.91
30	40	447.02	447.02	450.31	0.7%	43	2	1.00	[30,30]	110.73
30	45	557.22	557.22	557.22	0.0%	50	50	1.00	[30,30]	110.82
30	50	667.59	667.59	671.50	0.6%	49	1	1.00	[30,30]	110.89
30	60	894.21	894.21	899.26	0.6%	42	8	1.00	[30,30]	131.30
30	70	1126.24	1126.24	1129.97	0.3%	44	6	1.00	[30,30]	128.06
30	80	1363.65	1363.65	1366.16	0.2%	44	6	1.00	[30,30]	127.71
30	90	1606.37	1606.37	1606.97	0.0%	45	5	1.00	[30,30]	126.90
30	100	1851.07	1851.07	1858.82	0.4%	36	1	1.00	[30,30]	108.50
30	200	4485.59	4485.59	4487.72	0.0%	47	3	1.00	[30,30]	115.00
30	300	7462.70	7462.70	7462.94	0.0%	47	3	1.00	[30,30]	111.51
30	400	10931.79	10931.79	10932.04	0.0%	49	1	0.96	[30,30]	111.68
30	500	15354.65	15354.65	15355.82	0.0%	46	4	0.92	[30,30]	111.97
50	2	6.69	6.69	7.05	5.3%	46	2	0.92	[2,2]	112.57
50	4	14.41	14.41	14.53	0.8%	47	1	1.00	[4,4]	114.28
50	6	22.21	22.21	22.47	1.1%	38	12	1.00	[6,6]	110.97
50	8	30.78	30.78	31.75	3.2%	35	1	1.00	[8,8]	110.37
50	10	40.43	40.43	40.86	1.1%	41	9	1.00	[10,10]	124.99
50	15	67.69	67.69	68.74	1.6%	39	1	1.00	[15,15]	112.43
50	20	100.56	100.56	101.40	0.8%	45	5	1.00	[20,20]	110.09
50	25	137.65	137.65	139.93	1.7%	46	4	1.00	[25,25]	109.67
50	30	180.21	180.21	180.21	0.0%	50	50	1.00	[30,30]	110.13
50	35	230.45	230.45	230.45	0.0%	50	50	1.00	[35,35]	109.95
50	40	291.42	291.42	291.61	0.1%	47	3	1.00	[39,40]	110.25
50	45	355.21	355.21	355.21	0.0%	50	50	1.00	[44,45]	110.90
50	50	436.70	436.70	447.41	2.5%	36	1	1.00	[45,47]	110.23
50	60	622.71	622.71	628.88	1.0%	47	3	1.00	[48,47]	119.17
50	70	833.79	833.79	833.79	0.0%	50	50	1.00	[49,49]	126.24
50	80	1053.10	1053.10	1056.17	0.3%	39	11	1.00	[50,49]	123.81
50	90	1283.00	1283.00	1283.00	0.0%	50	50	1.00	[50,49]	123.26
50	100	1519.38	1519.38	1519.38	0.0%	50	50	1.00	[50,49]	112.76
50	200	4078.80	4078.80	4078.80	0.0%	50	50	1.00	[49,50]	115.28
50	300	6971.11	6971.11	6971.11	0.0%	50	50	1.00	[49,50]	120.11
50	400	10464.09	10464.09	10464.09	0.0%	50	50	1.00	[50,50]	119.26
50	500	14926.58	14926.58	14931.86	0.0%	49	1	1.00	[50,50]	120.61

Appendix B

RESULTS OF THE EXPERIMENTS CONDUCTED DURING ALGORITHM DEVELOPMENT

In this appendix, we provide experimental results which are conducted in order to develop proposed CCBFS and CCBFS-H algorithms.

Table B.1: Comparison of Best, Worst and Number of Best Solution Results on CCBFS Algorithm with and without Neighborhood Search

p,n,m,q	Best	Best - No NS	Worst	Worst - No Ns	# of Best	# of Best - No NS
4,80,5,2	7.13	7.13	7.13	7.63	50	6
4,80,6,2	8.15	8.15	8.35	8.41	35	4
4,80,8,2	7.80	7.80	7.80	8.50	50	1
4,80,10,2	7.26	7.26	7.62	7.79	24	2
4,80,12,2	6.96	6.96	7.83	7.89	15	4
4,80,5,3	10.00	10.00	10.00	10.40	50	2
4,80,6,3	9.17	9.17	9.17	9.70	50	10
4,80,8,3	11.75	11.75	11.91	12.47	25	2
4,80,10,3	14.92	14.92	17.05	16.58	7	1
4,80,12,3	9.51	9.51	9.51	10.58	50	1
4,80,5,4	11.57	11.57	11.57	12.63	50	16
4,80,6,4	14.72	14.72	14.72	15.55	50	16
4,80,8,4	13.81	13.81	13.98	14.71	25	2
4,80,10,4	12.89	12.89	12.89	14.13	50	2
4,80,12,4	19.30	19.30	19.83	22.84	26	3
4,100,5,2	6.91	6.91	6.91	7.12	50	7

Table B.1 continued from previous page

p,n,m,q	Best	Best - No NS	Worst	Worst - No Ns	# of Best	# of Best - No NS
4,100,6,2	9.84	9.84	9.84	10.41	50	11
4,100,8,2	11.50	11.50	11.90	12.02	42	2
4,100,10,2	8.36	8.36	8.36	8.82	50	2
4,100,12,2	10.68	10.68	12.45	12.14	19	1
4,100,5,3	12.30	12.30	12.30	12.80	50	4
4,100,6,3	12.27	12.27	12.27	13.48	50	3
4,100,8,3	14.11	14.11	14.28	15.09	13	2
4,100,10,3	15.19	15.21	15.33	16.49	9	2
4,100,12,3	13.84	13.84	13.84	14.84	50	1
4,100,5,4	15.72	15.72	15.84	16.30	49	11
4,100,6,4	18.52	18.52	18.52	19.31	50	1
4,100,8,4	17.36	17.43	17.36	18.77	50	2
4,100,10,4	20.00	20.00	20.78	21.56	9	3
4,100,12,4	20.89	20.89	22.43	23.16	15	1
4,200,5,2	20.32	20.32	20.32	21.00	44	2
4,200,6,2	15.59	15.62	15.87	16.26	13	1
4,200,8,2	20.83	20.87	21.79	22.26	48	1
4,200,10,2	17.12	17.12	17.12	18.52	50	1
4,200,12,2	18.70	18.74	19.18	20.01	31	1
4,200,5,3	25.10	25.10	25.57	26.32	13	1
4,200,6,3	29.73	29.80	29.73	31.90	50	1
4,200,8,3	29.56	29.56	30.16	31.35	19	1
4,200,10,3	27.43	27.65	28.33	30.20	48	1
4,200,12,3	24.93	25.08	24.93	28.22	50	1
4,200,5,4	33.27	33.27	33.37	34.56	13	2
4,200,6,4	32.82	32.82	32.96	35.05	3	1
4,200,8,4	33.77	33.94	33.77	37.69	50	1
4,200,10,4	36.63	36.63	36.91	39.78	37	1
4,200,12,4	38.03	38.03	40.49	41.24	5	1

Table B.1 continued from previous page

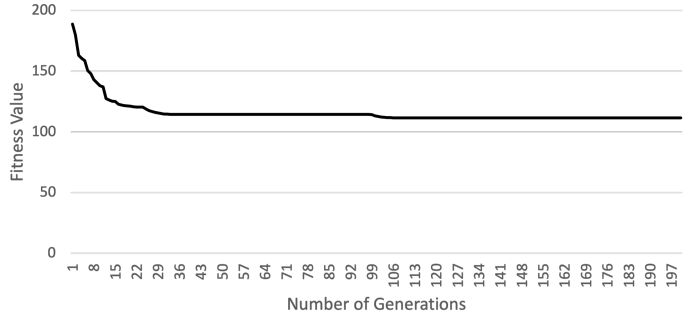
p,n,m,q	Best	Best - No NS	Worst	Worst - No Ns	# of Best	# of Best - No NS
4,500,5,2	39.14	39.32	39.14	41.18	50	1
4,500,6,2	42.62	42.76	43.31	43.86	10	1
4,500,8,2	56.17	56.38	56.17	61.40	50	1
4,500,10,2	48.08	48.34	48.49	51.19	12	1
4,500,12,2	46.46	46.65	54.70	55.04	7	1
4,500,5,3	58.69	58.80	59.27	62.49	32	1
4,500,6,3	61.44	62.28	61.44	67.47	50	1
4,500,8,3	67.01	67.98	70.22	71.73	23	1
4,500,10,3	68.73	69.24	70.78	75.98	7	1
4,500,12,3	76.33	77.94	88.72	92.88	34	1
4,500,5,4	75.75	76.51	78.29	80.15	12	1
4,500,6,4	78.51	79.91	78.56	86.24	23	1
4,500,8,4	92.09	93.05	93.45	100.59	12	1
4,500,10,4	89.16	89.74	94.77	100.59	26	1
4,500,12,4	98.92	100.00	103.52	113.93	31	1
4,1000,5,2	99.19	99.41	99.78	105.04	11	1
4,1000,6,2	98.41	99.04	98.41	101.96	50	1
4,1000,8,2	102.00	102.50	105.02	108.66	33	1
4,1000,10,2	72.49	73.24	72.49	82.00	50	1
4,1000,12,2	107.31	108.90	117.54	117.61	20	1
4,1000,5,3	131.87	133.41	133.26	139.20	6	1
4,1000,6,3	132.10	133.97	134.63	140.96	29	1
4,1000,8,3	127.04	128.89	129.47	145.82	12	1
4,1000,10,3	126.13	129.02	126.13	139.44	50	1
4,1000,12,3	173.63	175.85	185.28	189.96	1	1
4,1000,5,4	150.56	153.68	153.34	161.46	5	1
4,1000,6,4	146.41	147.61	150.36	158.31	46	1
4,1000,8,4	158.48	161.74	160.65	177.31	14	1
4,1000,10,4	190.11	191.45	191.77	214.37	10	1

Table B.1 continued from previous page

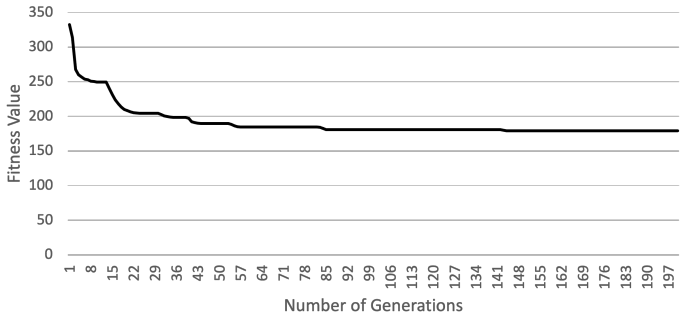
p,n,m,q	Best	Best - No NS	Worst	Worst - No Ns	# of Best	# of Best - No NS
4,1000,12,4	193.73	197.69	199.53	213.11	6	1

Table B.2: Comparison of Best, Worst and Number of Best Solution Results on CCBFS - H Algorithm with and without Second Neighborhood Search

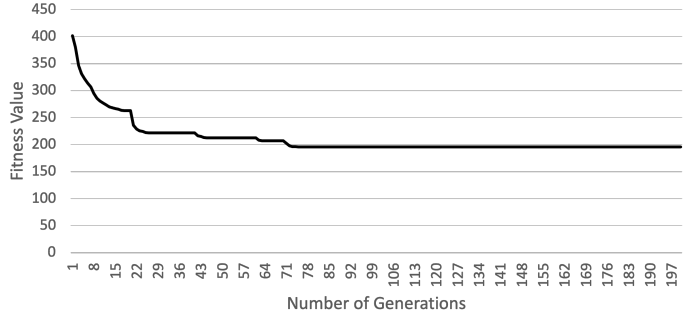
p,n,m,q	Best	Best - No NS 2	Worst	Worst - No NS 2	# of Best	# of Best - No NS 2
2,100,100,10	95.55	95.55	95.55	95.55	50	50
2,100,100,30	259.17	259.17	259.17	259.17	50	50
2,100,100,50	472.06	472.06	472.06	472.06	50	50
2,100,300,10	79.92	79.92	79.92	81.55	43	32
2,100,300,30	263.71	263.71	265.92	265.92	45	45
2,100,300,50	441.19	441.19	441.19	441.19	50	33
2,100,500,10	79.59	79.59	82.35	83.68	33	33
2,100,500,30	260.68	260.68	260.68	269.29	50	49
2,100,500,50	436.70	436.70	447.41	447.41	36	35



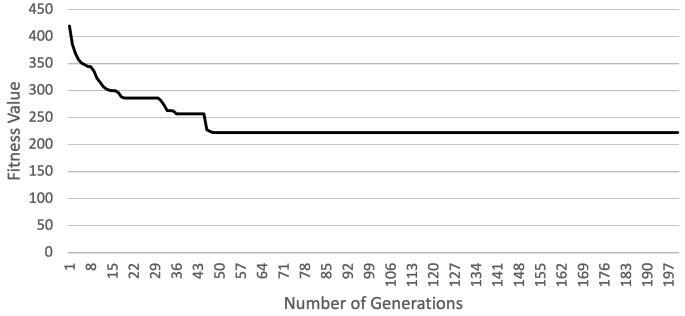
(a) $n = 1000$, $p = 4$, $m = 12$, $q = 2$



(b) $n = 1000$, $p = 4$, $m = 12$, $q = 3$



(c) $n = 1000$, $p = 4$, $m = 12$, $q = 4$



(d) $n = 1000$, $p = 3$, $m = 12$, $q = 4$

Figure B.1: Development of the Fitness Value with respect to Number of Generations for 4 Data Instances