NEIGHBORHOOD CONSTRUCTION-BASED MULTI-OBJECTIVE EVOLUTIONARY CLUSTERING ALGORITHM WITH FEATURE SELECTION

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

BY

CANSU ALAKUŞ

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

JULY 2018

Approval of the thesis:

NEIGHBORHOOD CONSTRUCTION-BASED MULTI-OBJECTIVE EVOLUTIONARY CLUSTERING ALGORITHM WITH FEATURE SELECTION

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

ABSTRACT

NEIGHBORHOOD CONSTRUCTION-BASED MULTI-OBJECTIVE EVOLUTIONARY CLUSTERING ALGORITHM WITH FEATURE SELECTION

Alakuş, Cansu
M.S., Department of Operational Research
Supervisor : Prof. Dr. Nur Evin Özdemirel
Co-Supervisor : Assoc. Prof. Dr. Cem İyigün

July 2018, 119 pages

In this study, we address the clustering problem with unknown number of clusters having arbitrary shapes, intracluster and/or intercluster density differences, no outliers or noise. The data set may be high-dimensional with a number of redundant features.

This study consists of two parts. In the first part, we propose a multi-objective evolutionary clustering algorithm, namely MOCNC, with three fundamental objectives of the clustering problem: compactness, separation, and connectivity. We use the multi-objective framework and nondominated sorting property of the well-known evolutionary algorithm NSGA-II to simultaneously optimize the compactness and separation objectives. To handle the connectivity objective, a special Neighborhood Construction (NC) algorithm is used as a preprocessor.

In the second part, we extend the MOCNC algorithm as MOCNC-F for the feature selection problem where the data sets may contain an unknown number of redundant features. In this algorithm, different subsets of features are selected in solutions and clustering is performed using the selected features. The output of MOCNC-F is a set of nondominated clustering solutions each with different compactness and separation values, and possibly with different feature subsets.

Our algorithms are unique in that they solve the feature selection and clustering problem simultaneously using the three fundamental objectives, which are compactness, separation, and connectivity, explicitly. The proposed algorithms do not need any user-defined problem parameters. We have experimented with the algorithms on generated and benchmark data sets, and obtained promising results based on selected performance criteria.

Keywords: Multi-Objective Clustering, Feature Selection, Evolutionary Algorithm, Neighborhood Construction

ÖZ

KOMŞULUK KURMA BAZLI ÇOK AMAÇLI EVRİMSEL KÜMELEME VE ÖZNİTELİK SEÇİMİ

Alakuş, Cansu
Yüksek Lisans, Yöneylem Araştırması Bölümü
Tez Yöneticisi : Prof. Dr. Nur Evin Özdemirel
Ortak Tez Yöneticisi : Doç. Dr. Cem İyigün

Temmuz 2018 , 119 sayfa

Bu çalışmada, gerçek küme sayısının bilinmediği, kümelerin gelişigüzel şekillere, küme içi ve/veya kümeler arası yoğunluk farklarına sahip olduğu, aykırı nokta ve gürültü içermeyen veri setleri üzerinde çalışılmıştır. Veri kümeleri çok boyutlu olup gereksiz öznitelikler içerebilir.

Bu çalışma iki kısımdan oluşmaktadır. İlk kısımda, kümeleme probleminin sıkılık, ayrıklık ve bağlantılılık temel amaçları ile çalışan MOCNC isimli çok amaçlı evrimsel bir algoritma önerilmiştir. Sıkılık ve ayrıklık amaç fonksiyonlarını aynı anda eniyilemek için NSGA-II algoritmasının çok amaçlılık ve baskın olmayan sıralama özellikleri kullanılmıştır. Bağlantılılık amaç fonksiyonu için ise NC komşuluk kurma algoritması ön işleme aşaması olarak kullanılmıştır.

İkinci kısımda, MOCNC algoritmasını, anlamsız öznitelik sayısının bilinmediği öznitelik seçimi problemi için MOCNC-F olarak genişlettik. Bu algoritmada farklı öznitelik alt kümeleri seçilir ve kümeleme seçilen özniteliklerle uygulanır. MOCNC-F algoritmasının çıktısı farklı sıkılık ve ayrıklık amaç fonksiyonu değerlerine sahip ve muhtemelen farklı özniteliklerle oluşturulmuş domine edilmeyen kümeleme çözümleridir.

MOCNC-F sıkılık, ayrıklık ve bağlantılılık amaç fonksiyonlarını kullanarak öznitelik seçimi ve kümeleme problemini aynı anda çözen özgün bir algoritmadır. Kullanıcı tarafından tanımlanması gereken problem paremetresine ihtiyaç duymamaktadır. Algoritmalarımızı üretilmiş ve bilinen veri kümeleri üzerinde test ettik ve başarılı sonuçlar elde ettik.

Anahtar Kelimeler: Çok Amaçlı Kümeleme, Öznitelik Seçimi, Evrimsel Algoritma, Komşuluk Kurma

To my parents...

ACKNOWLEDGMENTS

First of all, I would like to express my sincere thanks to my supervisors Prof. Dr. Nur Evin Özdemirel and Assoc. Prof. Dr. Cem İyigün for their academic guidance, insight, and support. Their wisdom has guided me to new research areas.

Secondly, I would like to thank to my colleagues who are also my beloved friends. I thank to Sena Önen for the academic discussions we made, being a kind companion during the courses that we took together, and the enjoyable moments in our room. I also thank to Dilay Aktaş, Melis Özateş Gürbüz, and Altan Akdoğan for their motivational talks and support.

Most importantly, I offer my greatest thanks to Can Barış Çetin to be with me all the time with love and patience. I always have his endless support when I was stuck.

I would also like to thank to examining committee members Prof. Dr. Gülser Köksal, Assist. Prof. Dr. Mustafa Kemal Tural, and Assist. Prof. Dr. Mustafa Gökçe Baydoğan for their time in reviewing this work and valuable comments.

Last but not least, special thanks go to my parents Nazire and Ali İhsan Alakuş, and my little sister Ecem Alakuş for their endless support and unconditional love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER 1

INTRODUCTION

Data mining is the process of extracting interesting and useful patterns from large amounts of raw data. These interesting and useful patterns are the knowledge obtained from the data. Data mining is an interdisciplinary field that makes use of tools and techniques from computer science statistics, operational research, and machine learning. The data mining techniques are used in a wide variety of application areas. As the amount of data collected increases, the need to extract valuable information from data gains more importance. Hence, the need for developing new techniques increases considering the requirements of different application areas.

The main data mining problems are classification, clustering, prediction, association rule learning, and outlier analysis. In this study, we focus on a clustering problem. The clustering problem is concerned with dividing a data set into groups of data points according to their similarities. The main aim of the clustering algorithms is to put similar data points into the same group and dissimilar points into different groups. Clustering helps the decision maker to summarize data points by grouping them into a smaller number of groups.

Clustering is an unsupervised learning method which aims to discover natural groupings in a data set without any external knowledge. It has important real life applications in different fields such as marketing, biology, bioinformatics, social sciences, climatology, security, and so on.

The clustering problem has some challenging characteristics. Firstly, the similarity measure should be selected considering the data type (numerical or categorical), which affects the choice and the calculation of the similarity measure. Secondly, the choice of the objective function(s) is not straightforward. Conceptually clusters in a solution should be compact and connected, and well-separated from each other. However, it is difficult to find suitable measures that satisfy all three requirements. Different choices of objective functions may result in different groupings of data points. Thirdly, as the true cluster labels of data points are not known a priori, evaluation of the resulting clustering is not straightforward and identification of a good clustering solution is not simple. Moreover, data sets may contain density differences and arbitrary shapes. Most traditional clustering algorithms fail to discover arbitrary shaped clusters and clusters having density differences. Connectivity-based objective functions and neighborhood definitions may be required in such cases. Also, data sets may have outliers and/or noise, which may mask the underlying clustering structure and mislead the clustering algorithm. Finally, the dimensionality (the number of features) and size (the number of points) of the data set affect the performance of the clustering algorithm. Some algorithms perform well on low dimensional data sets but show poor performance on high dimensional data sets. The size of the data set increases the computational challenges where the fast extraction of information from the data becomes more important steadily.

Due to these issues, several clustering algorithms are proposed in the literature. Each algorithm has different objectives, strengths and weaknesses. For example, partitional clustering algorithms and most of the metaheuristic approaches are mainly distance-based. The goal of these approaches is to divide the data set into clusters so that the points in the same cluster are gathered around a centroid and the clusters are disjoint. The main disadvantage of the partitional clustering algorithms is the need for defining the number of clusters a priori. Moreover, these traditional clustering algorithms mostly consider only a single objective of the clustering problem. On the other hand, hierarchical clustering algorithms construct a hierarchical decomposition of a data set. Different clustering solutions can be obtained by cutting the decomposition at different levels. The main disadvantage of these algorithms is that the merging or splitting decisions cannot be undone. Hence, once an incorrect decision is made at any level of the decomposition, it cannot be reversed.

The choice of clustering algorithm is affected from the application domain, the clustering goal, and the requirements of the specific area. After clarifying them, the requirements should be matched with the clustering algorithms considering their assumptions, strengths, and weaknesses.

In this study, considering the challenging issues of clustering, we study the clustering problem in which features are numerical, there may be density differences between and/or within clusters, there are no outliers or noise, clusters may have arbitrary shapes, and the data set may be high-dimensional. We also assume that the number of clusters is not known a priori.

This study has two parts. In the first part, we propose a multi-objective evolutionary clustering algorithm that can handle three objectives of the clustering problem, compactness, separation, and connectivity in a two-phase approach. We handle the compactness and separation objectives by using them as the objective functions of the evolutionary algorithm. Our proposed approach uses the multi-objective framework and nondominated sorting property of the well-known algorithm NSGA-II (Deb et al. 2002) to simultaneously optimize these two conflicting objectives. For the third objective, which is the connectivity of data points, we use a special Neighborhood Construction (NC) algorithm (İnkaya et al. 2015a) as the preprocessor of our evolutionary algorithm. NC uses the underlying connectivity and density information in the data set to construct neighborhoods specific to data points. It then forms closures considering overlaps of these neighborhoods. These closures ensure the connectivity of data points.

In the second part of the study, we extend the problem to feature selection and a new feature selection algorithm is proposed based on our multi-objective clustering algorithm. A data set may contain irrelevant (redundant) features. The redundant features may mask the clustering pattern and make it difficult to extract clusters. They may even distort the data such that data points become indistinguishable. Feature selection methods aim at detecting relevant features and reducing the dimensionality of the data set. In our extension to feature selection, the clustering problem defined above remains the same, but now the data sets may contain redundant features. In our second multi-objective evolutionary algorithm, it is aimed to select subsets of features and clustering is performed using the selected features. The output of the proposed algorithm is a set of nondominated clustering solutions each with different compactness and separation values, and possibly resulting from different feature subsets.

Our approach for feature selection and clustering is unique in that it addresses the compactness, separation, and connectivity objectives explicitly. Our algorithms are the only ones in literature, which can solve the feature selection and clustering problems simultaneously using these three fundamental objectives. The proposed algorithms do not need any user-defined problem parameters. To the best of our knowledge, there is no algorithm that takes into account the feature selection and clustering problems simultaneously in a multi-objective perspective, where both the number of clusters and the number of redundant features are unknown. Experimental results show that our approach is successful in selecting the relevant features and finding the target clustering.

Outline of the thesis is as follows. In Chapter 2, we give a background for the general clustering problem and its characteristics. We classify the clustering algorithms in the literature and describe their assumptions, advantages, and disadvantages. Then, we explain the general feature selection problem and the feature selection methods in the literature by discussing their strengths and limitations. Finally, we provide a detailed literature review on the multi-objective clustering and feature selection algorithms. We classify the studies in literature concerning our problem and present them by explaining their goals, underlying assumptions, strengths, and limitations.

In Chapter 3, we give the detailed problem definition by referring to the characteristics of the clustering problem and the related literature. Then, we introduce our proposed solution approach for the multi-objective clustering problem. Finally, we briefly describe the extension of our multi-objective clustering algorithm for the feature selection problem.

The proposed multi-objective clustering algorithm (MOCNC) is explained in detail in Chapter 4. The experimental results of the algorithm are presented and discussed in Chapter 5. Chapter 6 describes the feature selection extension of the multi-objective clustering algorithm (MOCNC-F) in detail. The experimental results for the feature selection problem is given in Chapter 7. Chapter 8 summarizes the main findings of this study and concludes with future research directions.

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

The clustering problem is a well-known problem in the literature studied by different disciplines. Each discipline constructs its own terminology for the clustering problem on hand. Hence, it is hard to construct a generally accepted definition of the clustering problem. Jain et al. (1999), Xu and Wunsch (2005), Berkhin (2006) and Jain (2010) review the terminology and techniques of clustering as well as its widely used application areas. They classify the studies in literature based on a similar scheme. Following this classification, in this chapter we firstly present the clustering problem and its characteristics in Section 2.1. Then, we briefly describe the traditional single-objective clustering algorithms in the literature in Section 2.2. In Section 2.3, we describe the problem of feature selection in clustering and present a brief literature review for feature selection algorithms for clustering. After giving a background and brief review for the clustering problem and the feature selection problem, Section 2.4 is dedicated to the literature of our specific problem of multi-objective clustering and feature selection.

2.1 The Clustering Problem and Its Characteristics

The machine learning techniques can be grouped into two main categories namely supervised learning and unsupervised learning. In supervised learning, the labels of the data are available and used to train the algorithm. Then, the trained algorithm is used to classify new data points. For instance, in classification there is a set of labeled data and this data set is used to classify newly observed unlabeled data points. On the other hand, in unsupervised learning, the labels are unknown and cannot be used during the learning process. Hence, unsupervised learning algorithms try to extract meaningful information from the unlabeled data (Witten et al. 2016). From the machine learning point of view, the unlabeled data has some hidden patterns and the aim of clustering is to understand the structure of the data and find the hidden patterns to summarize the data to obtain valuable information (Berkhin 2006).

Clustering groups the data set based on the similarities of the data points. The main aim of the clustering algorithms is to put similar data points into the same group and dissimilar points into different groups. Clustering helps to summarize data points by grouping them into a smaller number of groups. Representing the data by fewer groups has a great impact on data sets with a large number of points and attributes. Hence, clustering is related with various disciplines that deal with large data sets (Berkhin 2006).

To give a formal definition of the clustering problem, consider a data set D including N data points each having d attributes. A clustering algorithm tries to partition data set D into k groups (clusters) by using a similarity measure so that the data points in the same cluster are more similar to each other than the points in different clusters.

A data set may consist of points with continuous, binary or categorical attributes. The type of the attributes affects the calculation of similarity measure during clustering. The similarity between data points with all quantitative attributes is measured with distance metrics such as Manhattan, Euclidean, or Minkowski distances. The similarity between a pair of points having categorical or binary features is calculated with measures such as Hamming distance, Jaccard coefficient, simple matching coefficient, Rand Index, and so on (Xu and Wunsch 2005).

Clustering problem has some challenging issues in its nature. Due to these challenging issues, there is no single clustering algorithm applicable to all kinds of data sets. Since clustering algorithms have some assumptions on the structure of data set, similarity measures used in the algorithms, and objective functions, clustering in a specific application area and properties of the data set may lead to selecting different clustering algorithms and so obtaining different clustering solutions. For example, data sets may have arbitrary shaped cluster structures and/or density variations within and between clusters. These properties require special treatment. Moreover, one of the major difficulties in a clustering problem is defining a "good" clustering solution. Conceptually, a cluster should be compact, well-separated, and connected. However, mostly it is difficult to find appropriate measures for compactness, separation and connectivity of a clustering solution. Moreover, combining these objectives to obtain numerically comparable objective function values is not straightforward. Since the data set is unlabeled, the number of clusters is not known a priori. Without knowing the true labels, evaluating a clustering solution is vague. There are some validity indices to measure the quality of a clustering solution, but there is no consensus on a unique way to measure a clustering solution's quality. Moreover, clustering large data sets requires long computing times and large storage requirements.

Objective Functions

Clustering aims to group data points into clusters in order to achieve three objectives: compactness, separation and connectivity. Compactness is achieved by putting similar data points into the same cluster whereas separation is achieved by putting dissimilar data points into different clusters. As the third objective connectivity is achieved by putting a point and its closest neighbors into the same cluster. Although, the objectives of clustering are conceptually clear, defining appropriate measures for these objectives is not straightforward.

In the literature, several compactness measures are proposed for different clustering algorithms. These measures can be grouped into two categories as representative point based and individual pointwise compactness measures (Celebi and Aydin 2016). Representative point based compactness measures rely on finding a representative point for a cluster such as centroid or medoid. Calculating the similarity or dissimilarity between the data points in a cluster and the corresponding representative is given in Equation 2.1. These type of compactness measures perform well on spherical and homogenous data sets. However, they are not appropriate for the arbitrary shaped clusters. On the other hand, individual pointwise compactness measures use either the sum or the maximum of all pairwise point similarities within a cluster as in Equations 2.2 and 2.3.

CCrA: centroid-based, average of distances between cluster centroids and points within clusters

$$\frac{1}{N}\sum_{k=1}^{K}\sum_{i\in C_k} d_{ic_k} \tag{2.1}$$

CPA: pointwise, average of all pairwise distances between points within clusters

$$\frac{1}{\sum_{k=1}^{K}\binom{n_k}{2}}\sum_{k=1}^{K}\sum_{i\in C_k}\sum_{\substack{j\in C_k\\ j>i}} d_{ij} \tag{2.2}$$

CPM: pointwise, maximum of pairwise distances between points within clusters

$$max\{d_{ij} : \forall i, j \in C_k, i \neq j, \forall k\} \tag{2.3}$$

where C_k and n_k denote the set of points and the number of points in cluster k, respectively. c_k represents the centroid of cluster k. N is the number of points in the data set, and K is the total number of clusters. d_{ij} is the Euclidean distance between points i and j.

Similar to compactness, there are several separation measures used as an objective function. In the literature, distances between cluster representatives i.e. centroids or medoids of the clusters as in Equation 2.4. Linkage metrics such as *single-link*, *average-link* or *complete-link* are the most commonly used separation measures. Complete-link calculates the distance between the farthest points belonging to two different clusters. Average-link takes into account all pairwise point distances between two clusters, see Equation 2.5. Single-link calculates the distance between the closest points belonging to two different clusters as in Equation 2.6.

SCrA: centroid-based, average of all pairwise distances between cluster centroids

$$\frac{1}{\binom{K}{2}}\sum_{k=1}^{K-1}\sum_{l=k+1}^{K} d_{c_k c_l} \tag{2.4}$$

SPA: pointwise, average of all pairwise distances between points in different clusters

$$\frac{1}{\sum_{k=1}^{K-1}\sum_{l=k+1}^{K}(n_k * n_l)}\sum_{k=1}^{K-1}\sum_{l=k+1}^{K}\sum_{i\in C_k}\sum_{j\in C_l} d_{ij} \tag{2.5}$$

SPM: pointwise, minimum of pairwise distances between closest points in different clusters

$$min\{d_{ij} : \forall i \in C_k, \forall j \in C_l, \forall k, l, k \neq l\} \tag{2.6}$$

Here $d_{c_k c_l}$ refers to the Euclidean distance between the centroids of clusters k and l.

Connectivity is achieved by putting a point and its closest neighbors into the same cluster. Since connectivity considers the neighbors of data points, there is a need for a neighborhood definition for a data set. Some clustering algorithms construct the neighborhood by using a single similarity measure and some use special neighborhood definitions to measure the connectivity such as ε-neighborhood and k-nearest neighborhood (kNN). In ε-neighborhood, when the distance between two points is smaller than a prespecified distance ε, then these points are called as neighbors. In kNN, a fixed number of closest data points form the neighborhood of a data point. Connectivity performs well in detecting arbitrary shaped clusters.

As mentioned earlier, each objective has advantages and disadvantages on different data sets. Hence, there is no single objective function that performs well for all data sets and all clustering algorithms. The data set properties is critical for choosing the appropriate objective function. Using only one of these objective functions is not sufficient to obtain a good clustering solution for most data sets. There are some clustering algorithms using a combination of multiple objective functions as a single objective, e.g. maximizing the ratio between separation and compactness. However, reducing multiple objectives to a single objective results in information loss and does not provide good clustering solutions.

The Number of Clusters

The clustering problem is classified as unsupervised learning due to lack of prior knowledge of the true data labels. In addition to not knowing the true labels, most of the time the number of clusters is unknown as well. Hence, clustering algorithms are not provided the true number of clusters in grouping data points. Most of the traditional clustering algorithms assume the number of clusters are known a priori. However, in most real world data sets this information is not available before the clustering process. One way of estimating the number of clusters is to apply the clustering algorithm with a different number of clusters and select the solution that results with the best performance. In general, as the number of clusters increases, compactness gets better but separation gets worse. Also, applying the clustering algorithm for a wide range of number of clusters is time consuming. Moreover, evaluating the clustering performance for different number of clusters and selecting the best solution may be subjective or becomes harder as the size and dimensionality of the data set increases.

Arbitrary Shapes and Density Differences

Up to three dimensions one can plot the data set and assess the shape of the clusters and the density differences. Most commonly clusters may have spherical shapes but there are arbitrary shaped clusters as well. Detection of different shapes of clusters in a data set may need different objective functions. For example, in spherical clusters the similarity between data points can be measured as the distance to the cluster centroid since the spherical clusters are compactly dispersed around the cluster centroid. On the other hand, an arbitrary shaped cluster may not be detected with this compactness objective but needs a connectivity based objective.

In addition, the data set may contain between cluster and within cluster density differences. Between cluster density difference means that clusters are homogeneous in themselves but their densities are different from each other. On the other hand, a cluster may contain density difference in itself which is called within cluster density difference. Some data sets contain both types of density differences which makes extracting clusters difficult.

Outliers and Noise

An outlier is a point which is dissimilar from all other points in the data set. Outliers distort the clustering structure of the data set. Hence, most clustering algorithms are not successful in extracting clusters from data sets with outliers. In literature, some outlier detection mechanisms are proposed and used before applying the clustering algorithms. Also, there are some clustering algorithms that can handle outliers while clustering the data points.

Some data sets contain noise which is different from outliers. In this case, there are many points in the data set that do not belong to any of the clusters. Presence of noise creates similar problems as in the case of outliers.

High Dimensionality

In many application areas, data sets have many features or attributes. As the number of features increases, the data points become sparse in feature space and become more indistinguishable, hence the curse of dimensionality. Most of the traditional clustering algorithms use a distance measure to calculate the similarity between data points. However, when the dimensionality increases the distance measure becomes ineffective or less powerful to distinguish data points as the distance values become close together.

Moreover, all features may not be meaningful in the clustering process. Possibility of containing redundant (irrelevant) features increases as the dimension increases. The redundant features sometimes prevent the clustering algorithm from extracting the clusters. Also, dealing with many features is time consuming for most of the clustering algorithms.

Scalability

As the size of the data set and the number of clusters increase, the computing time and memory requirements increase. Therefore, clustering algorithms should be scalable for execution within a reasonable time while the size of the data sets increases.

2.2 Traditional Single Objective Clustering Algorithms

The variety of the clustering algorithms makes the classification of algorithms into distinct groups complicated. Each algorithm has different assumptions, strengths and weaknesses. According to some comprehensive reviews for clustering algorithms, traditional single objective clustering algorithms are mainly divided into three groups namely partitional clustering algorithms, hierarchical clustering algorithms, and metaheuristic algorithms for clustering (Jain et al. 1999, Xu and Wunsch 2005, Berkhin 2006).

2.2.1 Partitional Clustering Algorithms

Partitional clustering algorithms take the clustering problem as an optimization problem. These algorithms divide the data set into disjoint clusters by optimizing a criterion and end up with a single partition. The objective function is mostly selected as the total distance or variation (squared distance) between the data points and the cluster representatives. Hence, the algorithms tend to work well on data sets having well-separated and compact clusters. The main advantage of the partitional clustering algorithms is the relatively lower time and memory requirements. They are advantageous especially when dealing with large data sets. The main problem of these algorithms is choosing the number of clusters to give as an input to the algorithm. Also, the performance of these algorithms depends on initial selection of cluster representatives. Besides, these algorithms are limited with data sets having numerical features. The most commonly used partitional clustering algorithm is *k-means* (MacQueen et al. 1967).

2.2.2 Hierarchical Clustering Algorithms

Hierarchical clustering algorithms produce nested series of partitions. These partitions are generally represented with tree-like structures called dendrograms. Each leaf node in a dendrogram represents a data point and the root node represents the whole data set. At each level of the tree there is a different partitioning of the data set.

Hierarchical clustering algorithms can be categorized into two according to the direction of processing the tree: agglomerative and divisive. In agglomerative methods, clusters with single points are merged according to a criterion until obtaining a single cluster including all points. On the other hand, the divisive methods start with a single cluster of the whole data set and continue dividing it into subsets until obtaining single point clusters.

Hierarchical clustering algorithms use linkage metrics to merge or split the clusters. The most commonly used linkage metrics are single-link, average-link and complete-link. In single-link, the splitting or merging is based on the minimum distance between different clusters. Similarly, average-link and complete-link use the average and maximum of the inter-cluster distances, respectively.

In hierarchical clustering algorithms the errors made in previous iterations cannot be corrected. Since the hierarchical clustering algorithms are greedy, if a wrong merging or splitting decision is made in an iteration, its effects last until the end. Moreover, the usage of above linkage metrics leads hierarchical clustering algorithms to generate spherical clusters. These algorithms are not successful at finding arbitrary shaped clusters. Besides, these algorithms are sensitive to noise and outliers. Another disadvantage of hierarchical clustering algorithms is the excessive memory requirement. The main advantage of the hierarchical algorithms is that there is no need to know the correct number of clusters a priori. However, extra methods may be required to obtain the best partition from the dendrogram. The well-known algorithms of this type are CURE (Guha et al. 1998), BIRCH (Zhang et al. 1996), and CHAMELEON (Karypis et al. 1999).

2.2.3 Metaheuristic Algorithms for Clustering

The clustering problem is an NP-hard problem. Searching for all possible clustering partitions is computationally expensive. Hence, solution space must be searched efficiently. Some heuristic approaches such as *k-means* are developed to solve the clustering problem. However, these conventional heuristics may get stuck at local optima. Metaheuristics have exploratory properties to search the solution space and to find the global optimum by avoiding the local optima. There are several types of metaheuristics applied to the clustering problem such as Evolutionary Algorithms, Simulated Annealing, and Tabu Search. Among these, Evolutionary Algorithms (EA) are the most commonly used ones. Rayward-Smith (2005) provides a brief review of metaheuristic implementations for the clustering problem. Hruschka et al. (2009) focuses on the evolutionary algorithm applications on clustering. This study covers traditional EA applications, multi-objective clustering algorithms, and cluster-ensemble clustering applications.

2.3 Feature Selection

Each data point has a number of features (attributes) representing properties of that point. The number of features determines the dimensionality of the data set. As the dimensionality of the data set gets high, the distances between the pairs of data points, as measured by various dissimilarity measures, get close together. Then, the power of understanding the underlying patterns and obtaining useful information decreases, hence the curse of dimensionality (Bellman 1961). The data sets may have both relevant and irrelevant (redundant) features. The relevant features are the necessary ones to obtain information from the data sets. However, the irrelevant features do not add valuable information to the data. Moreover, sometimes, they may prevent us from recognizing the underlying patterns. Besides, containing redundant features might increase the computational time and storage requirements and decrease the prediction performance of the learning algorithm. To avoid the effects of the curse of dimensionality and obtain more generalizable models from data sets, dimensionality reduction techniques can be applied (Tang et al. 2014, Saeys et al. 2007).

The dimensionality reduction techniques can be grouped into two main categories: feature extraction and feature selection. Feature extraction techniques decrease the dimensionality by projecting the original features into a new feature space with lower dimensionality. The new features are the linear or nonlinear combinations of the original features (Tang et al. 2014). The well-known examples of feature extraction techniques are Principal Component Analysis, Canonical Correlation Analysis, Multidimensional Spacing, and Linear Discriminant Analysis (Guyon and Elisseeff 2006). On the other hand, feature selection techniques start with all original features, eliminates redundant features during the process and end up with a subset of the original features.

Recent comprehensive reviews are provided by Tang et al. (2014), Chandrashekar and Sahin (2014), and Saeys et al. (2007). The feature selection methods are generally grouped into two main categories, namely *filtering* and *wrapper* methods. Filtering methods are applied before the unsupervised or supervised learning algorithms, i.e. they eliminate the redundant features as a pre-preprocessing step and give the selected feature subset as input to the algorithms. Filtering methods consider the inherent properties of the data to filter out the redundant features. They mostly rank the features based on the information of correlation or dependency between features and then remove the ones below the predefined thresholds. Some examples of the *filtering* methods are Relief (Kira and Rendell 1992) and Mutual Information (Shannon 2001), and Fisher score (Duda et al. 2012).

On the other hand, *wrapper* methods evaluate the subsets of features by using the learning algorithms. In other words, *wrapper* methods search the feature space to find the best feature subset and eliminate the redundant ones. Each feature subset is given as input to the learning algorithm and the decision to eliminate or keep a feature is given by using the objective function value of the algorithm. The feature subset with the best objective function value is chosen. The number of possible solutions for an N dimensional data is 2^N. Since full enumeration is computationally expensive and time consuming, deterministic or randomized search algorithms are used. The common ones are Branch and Bound method, Hill-Climbing, Genetic Algorithms and Best-First Search.

Both *filtering* and *wrapper* methods have some advantages and disadvantages. The main advantage of *filtering* methods is their independence from the data mining algorithm. Hence, these methods are more scalable and faster than *wrapper* methods. However, *filtering* methods might not discover the dependencies and interactions between the features, which may lead to loss of information and poor performance. Besides, *filtering* methods ignore the interaction with the data mining algorithm. On the contrary, since *wrapper* methods interact with the data mining algorithm in evaluating the feature subsets, mostly they show good performance on extracting feature dependencies and eliminating redundant features. Main drawback of *wrapper* methods is high time and the memory requirements.

So far we provide a brief background for the clustering problem and its characteristics, and the feature selection problem. Rest of this chapter is for the literature review related to our problem.

2.4 Literature Review for Multi-Objective Clustering and Feature Selection

In literature, there are many single objective clustering algorithms having different clustering objectives. While some clustering objectives lead to good clustering solutions for a specific data set, they may perform poorly for others. Since data sets may have different shapes, densities and sizes, no single objective clustering algorithm can find good solutions for all data sets. Furthermore, there are some data sets for which single objective clustering algorithms lead to particularly poor solutions. For these reasons, using multiple clustering objectives together helps to explore tradeoffs between objective functions and may result in better clustering solutions (Handl and Knowles 2005a).

Freitas (2004) reviews the multi-objective clustering approaches in literature by referring to their advantages and disadvantages. There are three approaches named as weighted approach, lexicographic approach, and Pareto approach. In the weighted approach, multiple objectives are transformed to a single objective. User defined weights are assigned to the objectives, and these objectives are combined based on their weights. Then, the problem becomes a single-objective optimization problem. The main advantage of this approach is its simplicity. However, as the cost of its simplicity, it loses information by combining multiple and generally different kind of objectives into one. Also, setting the weights is not straightforward. In the second approach, priorities are given to the objective functions by the decision maker, and objectives are solved one by one as a single-objective optimization problem in the order of their priorities. Although this approach does not mix different kinds of objectives, it needs some parameters to be specified. In the last approach, a set of nondominated solutions are obtained by considering multiple objectives simultaneously. This approach is more advantageous than the others since the objectives are not combined, hence there is not loss of information. Also, there is no need to define weights, priorities, or thresholds.

The main aim of the multi-objective clustering algorithms is to simultaneously optimize different and mostly conflicting objectives in a single clustering algorithm. This enables considering the tradeoffs between different objectives. The output of a multi-objective clustering algorithm is nondominated clustering solutions on a Pareto front, each having different objective function values. Taking a broad view, based on the objectives of the studies, we group the literature on multi-objective clustering and feature selection into three, namely clustering, feature selection, and simultaneous feature selection and clustering. The classification of these studies is presented in Table 2.1.

Table 2.1: Classification of multi-objective clustering and feature selection studies

Clustering	**Feature Selection**	**Simultaneous Feature Selection and Clustering**
Chen and Wang (2005)	Handl and Knowles (2006b)	Dutta et al. (2013)
Demirtaş (2011)	Handl and Knowles (2006c)	Saha et al. (2014)
Du et al. (2005)	Kim et al. (2002)	Saha et al. (2016a)
Garza-Fabre et al. (2017)	Mierswa and Wurst (2006)	
Handl and Knowles (2004)	Morita et al. (2003)	
Handl and Knowles (2005a)		
Handl and Knowles (2005b)		
Handl and Knowles (2006a)		
Handl and Knowles (2007)		
Handl and Knowles (2012)		
İnkaya et al. (2015b)		
Martínez-Peñaloza et al. (2017)		
Matake et al. (2007)		
Özyer et al. (2004)		
Prakash and Singh (2015)		
Ripon et al. (2006)		
Saha and Bandyopadhyay (2010)		
Saha et al. (2016b)		
Santos et al. (2009)		
Tsai et al. (2012)		
Won et al. (2008)		

2.4.1 Multi-Objective Metaheuristic Algorithms for Clustering

The clustering problem is NP-hard, and when the problem has multiple objectives it becomes even harder. To find good solutions to the multi-objective clustering problem mostly heuristic approaches are proposed in the literature (Ferligoj and Batagelj 1992).

In the literature, metaheuristic approaches to the unsupervised clustering problem consist of evolutionary algorithms (EA), simulated annealing (SA), particle swarm optimization (PCO), and ant colony optimization (ACO). However, most of the multi-objective metaheuristic approaches to the clustering problem are based on evolutionary algorithms.

Handl and Knowles (2004) present VIENNA (Voronoi Initialised Evolutionary Algorithm) which is based on multi-objective evolutionary algorithm PESA-II (Corne et al. 2001) for the unsupervised clustering problem. Their objective functions are minimizing compactness and maximizing connectivity. The second objective function enables the algorithm to be applied to arbitrary shaped clusters. However, VIENNA needs the number of clusters to be known a priori. Decision variables (genes) of the algorithm are the cluster labels of the data points. In the initialization step, they use Voronoi representation to construct the initial population. After generating the members of the initial population, they continue with direct representation.

Chen and Wang (2005) present a clustering algorithm based on a well-known multi-objective evolutionary algorithm, NSGA-II (Deb et al. 2002). The objective functions of the proposed algorithm are similar to those of VIENNA. The main difference between these two algorithms is the decision variables. The genes in the proposed algorithm represent the cluster centroids and the number of clusters. Hence, the number of clusters need not be known a priori.

To avoid fixing the number of clusters beforehand, Özyer et al. (2004), Du et al. (2005), and Won et al. (2008) use the number of clusters as one of the objective functions in their algorithms. They find a Pareto front such that each nondominated solution represents the total within cluster variation for the corresponding number of clusters.

The multi-objective clustering with automatic k-determination (MOCK) is proposed by (Handl and Knowles 2005a,b, 2007). MOCK mostly uses the idea presented in VIENNA (Handl and Knowles 2004). The objective functions are again compactness and connectivity with little modification in their calculations. MOCK is again based on PESA-II (Corne et al. 2001). The main differences between VIENNA and MOCK are in the representation schemes, the evolutionary operators, and the initialization procedure. In MOCK, because the number of clusters is unknown in advance the locus-based adjacency representation is used. This representation scheme also enables using standard crossover operators of evolutionary algorithms. Therefore, uniform crossover is used in MOCK. The mutation operator is the neighborhood-biased mutation which tends to increase the probability of mutation of the link to the farther neighbor. The major disadvantage of this mutation operator is the need for a user defined parameter, neighborhood size, since the value of this parameter affects the performance of clustering with different data set structures. The members of initial population are generated from two different single objective algorithms namely minimum spanning tree and *k-means*. With this initialization, the algorithm starts generations with either compact or connected solutions.

There are some studies to analyze the computational time and the search capability of the original MOCK algorithm. Matake et al. (2007) investigate the algorithm when the size of the data set is large. Tsai et al. (2012) use various crossover and mutation operators to increase the search diversity. Furthermore, the adopted multi-objective evolutionary algorithm of MOCK is analyzed as a source of improvement by Martínez-Peñaloza et al. (2017). For comparison SPEA-2 (Zitzler et al. 2001), NSGA-II (Deb et al. 2002) and MOEA/D (Zhang and Li 2007) are tried as the underlying evolutionary algorithm and NSGA-II is found the most successful one.

Garza-Fabre et al. (2017) propose an improved version of MOCK with major changes in the underlying evolutionary algorithm, initialization procedure and representation scheme. Due to the elitist behavior of PESA-II, they use NSGA-II in the improved version. To generate members of the initial population, they only use the minimum spanning tree based solutions since the benefit of solutions obtained from *k-means* to the clustering performance is negligible. Furthermore, in order to reduce the computational cost of the algorithm they propose two new reduced length representations.

They fix some of the links in the chromosomes by a user-defined parameter in the initialization process and continue generations with these reduced length chromosomes.

Choice of objective functions in multi-objective clustering algorithms affects the behavior of the algorithm. The objective functions can be grouped into three categories, namely objective functions related with compactness, separation, and connectivity. Although the objective functions under these groups fundamentally focus on intra-cluster similarity and inter-cluster dissimilarity, they have some differences. Some clustering objective functions take into account all data points whereas others include only representatives of the clusters or some portion of the data. Using partial data may cause information loss but improves the computational cost. On the other hand, using all data points in calculating clustering objective functions may yield more information about the underlying data structure (Handl and Knowles 2012).

Besides compactness and connectivity as the objective functions, compactness and separation are also used as two conflicting objectives in multi-objective clustering (Ripon et al. 2006, Demirtaş 2011). Furthermore, some clustering validity indices are used as objective functions for this purpose (Saha and Bandyopadhyay 2010, Saha et al. 2014, 2016a,b).

In addition to the evolutionary algorithms, there are ACO-based multi-objective clustering algorithms presented in Santos et al. (2009) and İnkaya et al. (2015b). PCO and SA-based algorithms are presented in Prakash and Singh (2015), and Saha and Bandyopadhyay (2010), respectively.

Handl and Knowles (2006a) and Saha et al. (2016a) focus on extensions of multi-objective unsupervised clustering to semi-supervised clustering by adding a supervised objective function, namely Adjusted Rand Index (ARI), to the current objective function set.

Table 2.2 presents a summary of the algorithms discussed here. The table includes the type of the clustering problem, objectives, whether or not the number of clusters is known beforehand, capability to detect different shaped clusters, and the meta-heuristic approach. For the problem column, “U” and “SS” stand for unsupervised clustering and semi-supervised clustering, respectively. “K” in the number of clusters column means that the number of clusters is known a priori whereas “U” means that it is unknown. In the shape column “S” is an abbreviation for spherical clusters, “A” is for arbitrary shaped clusters and “PS” is for point symmetric clusters.

2.4.2 Multi-Objective Metaheuristic Algorithms for Feature Selection

The proposed solution approaches to the multi-objective feature selection problem in literature are similar to the approaches for the clustering problem. Transforming multiple approaches to a single objective, giving priorities to objectives and solving them in that order, and finding the set of nondominated solutions by using multiple objectives simultaneously. The most advantageous approach is finding the Pareto solutions by using multiple objectives separately. Due to the problem complexity, heuristic approaches are proposed to find good solutions (Pappa et al. 2004).

Similar to the clustering problem, multi-objective metaheuristic approaches are also applied to the feature selection problem. Almost all multi-objective algorithms for feature selection in clustering use *wrapper* feature selection methods. The algorithms use binary representation to encode feature subsets. Some of them define the number of clusters as a decision variable and include it in the chromosome. Since the clustering problem is basically unsupervised, the multi-objective feature selection algorithms evaluate a feature subset by the ability to identify the underlying clustering pattern. For this purpose, the algorithms use clustering validity indices. To evaluate each feature subset, a clustering algorithm such as *k-means* or expected maximization clustering is used. Objective functions for obtained clustering solutions are calculated and used as the fitness values.

Handl and Knowles (2006b) use MOCK for feature selection where the decision variables are features to be used for clustering and the number of clusters. In the algorithm the feature subsets are evaluated by *k-means* where the number of clusters is taken from the chromosome. They compare four pairs of objective functions in which one is the number of features. The number of features is used as a bias to affect the other objective function’s behavior. The same authors extend the problem to the semi-supervised feature selection problem by adding an external validity index for clustering (Handl and Knowles 2006c).

Kim et al. (2002), Morita et al. (2003) and Mierswa and Wurst (2006) have also studied multi-objective evolutionary algorithms for unsupervised feature selection. They all use *k-means* to evaluate candidate clustering solutions. Their differences come from the underlying evolutionary algorithm, and the number and choice of the objective functions. Kim et al. (2002) take into account four objective functions simultaneously, which are compactness, separation, the number of features, and the number of clusters. The output of the algorithm is a Pareto front in four dimensions, which is difficult to interpret. Also, their compactness and separation metrics cause dependencies with the number of features and clusters. The proposed approach use ELSA as the multi-objective evolutionary algorithm. Morita et al. (2003) and Mierswa and Wurst (2006) use a different multi-objective evolutionary algorithm, NSGA-II, with the objective functions normalized DB-Index and the number of features. These objective functions eliminate the interdependencies and also make interpretation of outputs easier.

Table 2.3 summarizes the multi-objective metaheuristic algorithms for the feature selection problem. “U” in the problem column stands for unsupervised feature selection whereas “SS” for semi-supervised feature selection.

2.4.3 Simultaneous Multi-Objective Feature Selection and Clustering

Decomposition of feature selection and clustering as two problems and solving them separately usually cause loss of information. Hence, solving them simultaneously in a single algorithm leads to better solutions. Including cluster and feature information in the representation enables evaluating the solution by using both sources of information. Most common representation in the literature is using centroids for clusters and binary coding for features (Saha et al. 2014, 2016a).

Another solution representation for this problem is defining the decision variables as cluster centroids but computing the centroids by using only the selected features for a single solution (Dutta et al. 2013). Therefore, the feature information is kept indirectly in their solution representation. The main disadvantage of such representations is the need for defining the number of clusters in advance.

The details of these algorithms are given in Table 2.4. The problem column presents the type of the clustering problem where “U” stands for unsupervised clustering and “SS” for semi-supervised clustering.

Table 2.2: A review of multi-objective metaheuristic algorithms for clustering

Author(s) and Year	Problem[1]	Objectives	# of Clusters[2]	Shape[3]	Approach	Comparison Measure	Compared Algorithms	Experimental Data Sets
Chen & Wang (2005)	U	1. min Compactness 2. max Connectedness	U	A	EA (NSGA-II)	F-measure	k-means	Real
Demirtaş (2011)	U	1. min Compactness 2. max Separation	U	A	EA (SPEA2)	Jaccard Index, Rand Index	k-means, Single-link, DBSCAN, MOEC	Real
Du et al. (2005)	U	1. min Compactness 2. min Number of clusters	U	S	EA (NPGA)	not compared with other algorithms		Real
Garza-Fabre et al. (2017)	U	1. min Compactness 2. min Connectedness	U	A	EA (NSGA-II)	Adjusted Rand Index	k-means, MOCK, MGA in Bandyopadhyay et al. (2007)	Generated
Handl & Knowles (2004)	U	1. min Compactness 2. max Connectedness	K	A	EA (VIENNA based on PESA-II)	F-measure	k-means, Average-link, Single-obj. versions of VIENNA	Generated and Real
Handl & Knowles (2005a)	U	1. min Compactness 2. max Connectedness	U	A	EA (MOCK based on PESA-II)	F-measure	k-means, Average-link, Single-link	Generated
Handl & Knowles (2005b)	U	1. min Compactness 2. max Connectedness	U	A	EA (MOCK based on PESA-II)	Adjusted Rand Index	k-means, Average-link, Single-link, Handl & Knowles (2005a)	Generated
Handl & Knowles (2006a)	SS	1. min Compactness 2. min Connectedness 3. max Adjusted Rand Index	U	A	EA (MOCK based on PESA-II)	Adjusted Rand Index	MOCK	Generated and Real
Handl & Knowles (2007)	U	1. min Compactness 2. min Connectedness	U	A	EA (MOCK based on PESA-II)	Adjusted Rand Index	k-means, Average-link, Single-link, Ensemble	Generated
İnkaya et al. (2015b)	U	1. min Adjusted compactness 2. max Relative separation	U	A	ACO (ACO-C)	Jaccard Index, Rand Index	k-means, Single-link, DBSCAN, NC, NOM	Real
Martínez-Peñaloza et al. (2017)	U	1. min Compactness 2. min Connectedness	U	A	EA (MOCK based on PESA-II, NSGA-II, SPEA-2, MOEA/D)	F-measure, Silhoutte Coefficient	k-means, Meta-clustering, Hierarchical clustering	Real
Matake et al. (2007)	U	1. min Compactness 2. min Connectedness	U	A	EA (MOCK based on PESA-II)	Pareto size	MOCK	Generated
Özyer et al. (2004)	U	1. min Compactness 2. min Number of clusters	U	S	EA (MOKGA)	not compared with other algorithms		Real
Prakash & Singh (2015)	U	1. min Compactness 2. min Connectedness	K	A	PSO (MOPSO)	F-measure	MOPSO, NSGA-II, MABC	Real
Ripon et al. (2006)	U	1. max Intracluster entropy 2. max Intercluster distance	U	S	EA (JGGA)	Dunn's Index, Classification Accuracy	NSGA-II	Generated and Real
Saha & Bandyopadhyay (2010)	U	1. min XB-index 2. max Sym-index	U	PS	SA (AMOSA)	Minkowski Score	MOCK, VGAPS, GCUK	Generated and Real
Saha et al. (2016b)	SS	1. max Sym-index 2. max I-index 3. max Adjusted Rand Index	U	PS	SA (AMOSA)	Adjusted Rand Index, Classification Accuracy	Unsupervised AMOSA, k-means, Average-link, SOM	Real
Santos et al. (2009)	U	1. min Compactness 2. min Connectedness	K	A	ACO (MACC)	F-measure	MOCK, Yang & Kamel Algorithm	Real
Tsai et al. (2012)	U	1. min Compactness 2. min Connectedness	U	A	EA (MOCK based on PESA-II)	Classification Acc., DB Index, PBM Index	MOCK	Real
Won et al. (2008)	U	1. min Compactness 2. min Number of clusters	U	S	EA (NSGA-II) & k-means	Total within cluster variance	k-means	Real

[1] U: Unsupervised, SS: Semi-supervised [2] U: Unknown, K: Known [3] A: Arbitrary, S: Spherical, PS: Point symmetric

Table 2.3: A review of multi-objective metaheuristic algorithms for feature selection

Author(s) and Year	Problem[1]	Objectives	# of Clusters[2]	Shape[3]	Approach	Comparison Measure	Compared Algorithms	Experimental Data Sets
Handl & Knowles (2006b)	U	1. max Silhouette Width 2. max Number of features	U	S	EA (PESA-II) & k-means	Adjusted Rand Index, F-measure	Each objective set is compared with each other	Generated and Real
		1. min DB-index 2. max Number of features						
		1. min Normalized DB-index 2. min Number of features						
		1. min Entropy 2. max Number of features						
Handl & Knowles (2006c)	SS	1. max Silhouette Width 2. max Number of features 3. max Adjusted Rand Index	U	S	EA (PESA-II) & k-means	Adjusted Rand Index	Unsupervised MOCK	Generated and Real
Kim et al. (2002)	U	1. min Compactness 2. max Separation 3. min Number of features 4. min Number of clusters	U	S	EA (ELSA) & k-means	not compared with other algorithms		Generated and Real
Mierswa & Wurst (2006)	U	1. min Normalized DB-index 2. max Number of features	K	S	EA (NSGA-II) & k-means	not compared with other algorithms		Generated and Real
Morita et al. (2003)	U	1. min Normalized DB-index 2. min Number of features	U	S	EA (NSGA) & k-means	not compared with other algorithms		Generated

[1] U: Unsupervised, SS: Semi-supervised
[2] U: Unknown, K: Known
[3] A: Arbitrary, S: Spherical, PS: Point symmetric

Table 2.4: A review of multi-objective metaheuristic algorithms for simultaneous feature selection and clustering

Author(s) and Year	**Problem**[1]	**Objectives**	**# of Clusters**[2]	**Shape**[3]	**Approach**	**Comparison Measure**	**Compared Algorithms**	**Experimental Data Sets**
Dutta et al. (2013)	U	1. min Compactness 2. max Separation	K	S	EA (MOGA) & k-means	DB index	k-means	Real
Saha et al. (2014)	SS	1. min XB-index 2. max Sym-index 3. max Adjusted Rand Index 4. max Number of features	U	PS	SA (AMOSA)	Minkowski score	k-means, Unsupervised AMOSA	Real
Saha et al. (2016a)	SS	1. min XB-index 2. max FCM-index 3. max I-index 4. max Sym-index 5. max Adjusted Rand Index 6. max Number of features	U	PS	SA (AMOSA)	Adjusted Rand, Index, Silhouette Index	Unsupervised AMOSA, MOGA, FCM, SGA, Average-link, SOM	Real

1 U: Unsupervised, SS: Semi-supervised
2 U: Unknown, K: Known
3 A: Arbitrary, S: Spherical, PS: Point symmetric

CHAPTER 3

PROBLEM DEFINITION AND SOLUTION APPROACH

In Chapter 2, the background for the clustering and the feature selection problems were given, and the related literature for our problem was reviewed. In this chapter, we firstly define our problem by referring to the characteristics of the clustering problem stated in Section 2.1. Then, we introduce our proposed solution approach.

The Clustering Problem

As discussed earlier, clustering is the aggregation of similar objects into the same group and separation of dissimilar objects into different groups. The similarity between objects can be found using various similarity measures. In the literature, there are many clustering algorithms proposed for different problem instances. However, there is not a single clustering algorithm applicable to all kinds of data sets.

Clustering is an ill-posed data mining problem due to its challenging properties. Firstly, the similarity measure should be selected based on the data type. Features can be quantitative (numerical) or qualitative (categorical). Data sets may include only quantitative features, only qualitative features, or a mixture of them. The data type affects the choice and the calculation of the similarity measure.

Secondly, the choice of the objective function(s) is not straightforward. Conceptually clusters in a solution should be compact, well-separated, and connected. However, it is difficult to find measures suitable to satisfy all three properties for a specific problem instance. The choice of objective functions may lead to different groupings of the data points. For example, some objective functions use centroid information whereas some use information of all points. Similarly, some objective functions use the explicit information from the entire data set whereas some use aggregated measures (internal validity indices, combination of multiple objective functions, etc.) to calculate the performance of the clustering solution.

Thirdly, as the true cluster labels of data points are not known a priori, comparison of the resulting clustering labels with the true labels is not possible. Hence, identification of a good clustering solution is not simple. The number of clusters is also unknown beforehand. In literature, some of the clustering algorithms take the number of clusters as given but in most real world problems we do not know the true number of clusters.

Moreover, data sets may contain density differences and arbitrary shapes. Most traditional clustering algorithms fail to extract clusters from data having density differences and arbitrary shapes. Connectivity based objective functions and neighborhood definitions may be required. Also, some data sets have outliers and/or noise, which may mask the underlying clustering structure and result in bad clustering solutions. Finally, the dimensionality and size of the data set affects the performance of the clustering algorithm. Some algorithms perform well on low dimensional data sets but show poor performance on high dimensional data sets. The size of the data set increases the computational requirements where the fast extraction of information from the data becomes more important steadily.

Considering all these issues, in this work, we study the clustering problem having the following properties.

- The data sets have numerical features.
- The similarity between data points is measured with Euclidean distance.
- The number of clusters is unknown.
- There may be density differences between clusters.
- There may be density differences within clusters.
- The data set does not contain outliers or noise.
- The data set may be high-dimensional.

- Clusters may have arbitrary shapes.

In our problem we take into account the three objectives of compactness, separation, and connectivity. In order to consider these three objectives simultaneously, we use a multi-objective evolutionary algorithm and a special neighborhood construction algorithm. We handle the compactness and separation objectives by using them as the objective functions of the evolutionary algorithm. Specifically, the objectives are minimizing the intra-cluster distances (compactness) and maximizing the inter-cluster distances (separation). Our proposed approach uses the multi-objective framework of the well-known algorithm NSGA-II (Deb et al. 2002) to simultaneously optimize these two conflicting objectives.

For the third objective, which is connectivity of data points, we use a special Neighborhood Construction (NC) algorithm (İnkaya et al. 2015a). NC uses the underlying connectivity and density information in the data set to construct neighborhoods specific to data points. It then forms closures considering overlaps of these neighborhoods. These closures may also be regarded as subclusters. In our approach, the links between data points within a closure are assumed to be fixed i.e., they are assumed to be connected. The closures are then given as input to the evolutionary algorithm, which further combines them throughout the generations and forms the final clusters. Hence, NC is used as a preprocessor for the proposed evolutionary algorithm.

We call our proposed solution approach for the clustering problem defined above MOCNC, which stands for **M**ulti-**O**bjective Evolutionary **C**lustering with **N**eigborhood **C**onstruction. The algorithm details are provided in Chapter 4 and experimental results are given in Chapter 5.

Using NC closures within the algorithm provides two benefits: (i) connectivity of data points in a closure is ensured and (ii) the problem becomes more scalable as the closures are fixed. The output of MOCNC is a set of nondominated clustering solutions. The nondominated solutions consider different tradeoffs between two conflicting objectives (compactness and separation) and may have different number of clusters.

In Table 2.1, a classification of studies in the literature is presented. In this classification, there are three categories namely clustering, feature selection, and simultaneous feature selection and clustering. Following this classification, MOCNC can be placed in the clustering category.

The Feature Selection Problem

One may notice that there are only a few studies on simultaneous feature selection and clustering. After finalizing experimental work with MOCNC, we extend the MOCNC algorithm for the feature selection problem. Each data point in a data set has a number of features. The number of features defines dimensionality of the data set. The data set may contain both relevant and irrelevant (redundant) features. The redundant features may mask the clustering structure and make it difficult to extract clusters. They may even distort the data such that data points become unseparable. Feature selection methods aim to detect relevant features and reduce the dimensionality of the data set.

In our extension for the feature selection problem, the clustering problem defined above remains the same, but now the data sets may contain redundant features. The extended approach for feature selection is called MOCNC-F. MOCNC-F is again a multi-objective evolutionary algorithm, which searches through different feature subsets. For each feature subset MOCNC finds the nondominated clustering solutions. Fitness of the selected feature subset is defined by the fitness (compactness and separation) of the clustering solution. Then, MOCNC-F analyzes the dominance between different clustering solutions resulting from different feature subsets. The output of MOCNC-F is a set of nondominated clustering solutions each with different compactness and separation values, and possibly with different feature subsets.

The details of MOCNC-F are presented in Chapter 6 and experimental results are given in Chapter 7.

CHAPTER 4

MOCNC : MULTI OBJECTIVE EVOLUTIONARY CLUSTERING WITH NEIGHBORHOOD CONSTRUCTION

In this chapter, we describe our multi-objective evolutionary clustering algorithm MOCNC in detail. As stated in Chapter 3, we use the Neighborhood Construction (NC) algorithm and a multi-objective evolutionary algorithm (NSGA-II) in our proposed approach. Before giving the details of MOCNC, we introduce the NC algorithm in Section 4.1 and the underlying multi-objective evolutionary algorithm, NSGA-II, in Section 4.2. In Section 4.3, we present a flowchart of the proposed algorithm and briefly introduce the algorithm steps. We present the notation used for the clustering problem and in the proposed algorithm, and give the details of the algorithm in a bottom-up approach in Section 4.4.

4.1 The Connectivity-Based Neighborhood Construction (NC) Algorithm

İnkaya et al. (2015a) proposes a special neighborhood construction algorithm based on the density and connectivity information of points in a data set. Most neighborhood construction algorithms require some user specified parameters, and selection of these parameters is not straightforward. The NC algorithm has the advantage that it does not require any such parameters. No specific information about the data set is needed beforehand, and the algorithm uses only the local characteristics of the data set to construct the neighborhood. It uses the Gabriel graph to extract the proximity relations of data points. From the constructed Gabriel graph, NC calculates the density of each data point. Density between two points is defined as the number of points in the closed ball of diameter Euclidean distance between the these points. The data

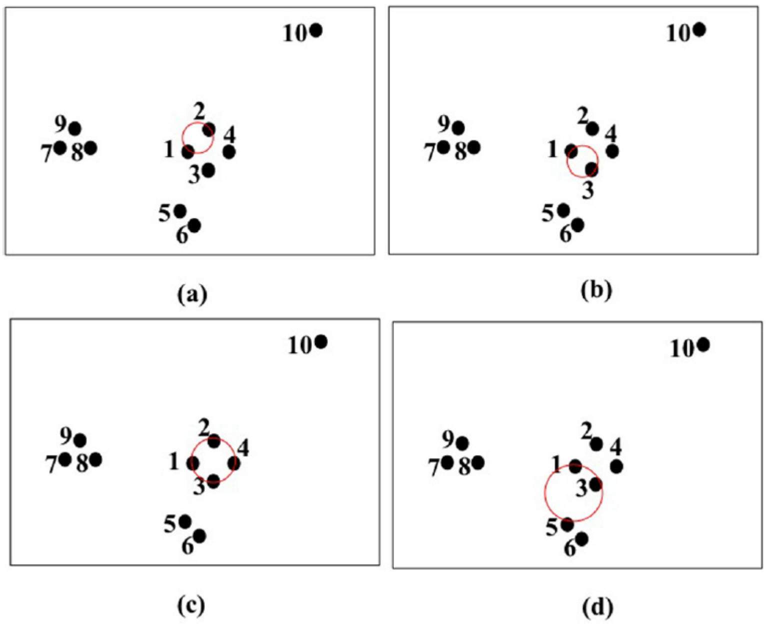


Figure 4.1: Example for the NC algorithm (from İnkaya et al. 2015a)

points are identified as having direct or indirect connectivity with each other based on their density values. The output of the NC algorithm is a unique neighborhood for each data point. Moreover, NC forms closures (subgroups) of data points based on overlapping neighborhoods. Those data points that share the same neighbors in their final neighborhoods belong to the same closure. Since NC uses the connectivity information, the data points in each closure are connected.

An example from İnkaya et al. (2015a) for neighborhood construction part of the NC algorithm is illustrated in Figure 4.1. In the example, there is a data set with ten points. For data point 1, the points are ordered based on the distance as $T_1 = \{2, 3, 4, 5, 6, 8, 9, 7, 10\}$. As can be seen in Figure 4.1(a)-(b), the density with respect to points 2 and 3 are zero since there is no point in the ball of diameter Euclidean distance between points 1 and 2, so they are 'direct' neighbors of point 1. The closest 'indirect' neighbor of point 1 is point 4 with a density of 2 as shown in Figure 4.1(c). Here, the indirect connection is established through points 2 and 3. The density values for the ordered set T_1 are 0, 0, 2, 1, 2, 0, 1, 2, 2. As we move in the ordered set T_1, density should be the same or increase for respective points to be true neighbors of point 1. Hence, point 5 is the break point where the density decreases, as seen in Figure 4.1(d). Up to the break point, the points 2, 3, and 4 are the neighbors of

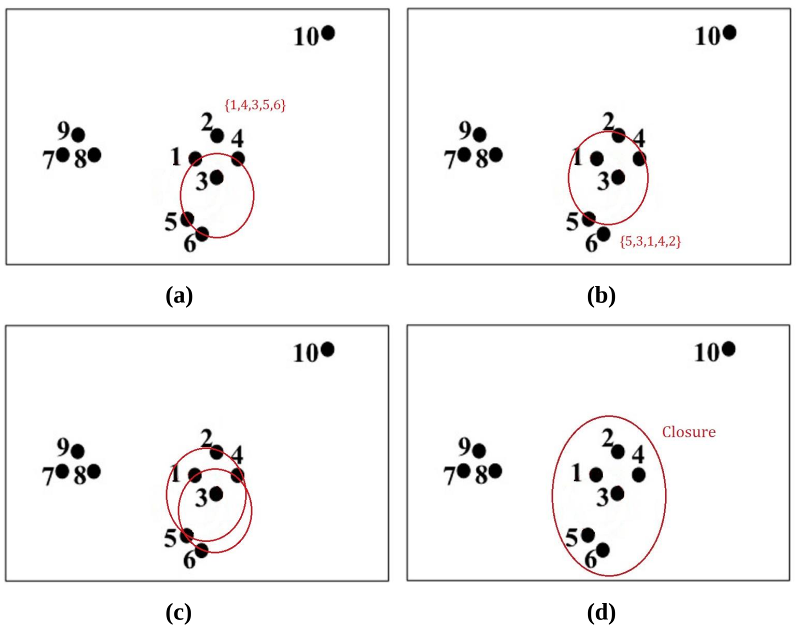


Figure 4.2: Example for the closure construction of the NC algorithm

point 1. In NC, the break points may either indicate a direction change or a different density region. To extend the neighborhood, the two sets including direct and indirect neighbors of points 1 and 5 are compared. If their intersection is not empty, then points 1 and 5 are indirectly connected. Otherwise, they do not belong to each others' neighborhoods. The direct and indirect neighbors of point 5 are 6, 3, 1, 4, 2. Points 1 and 5 have 3 neighbors in common, so point 5 is in the neighborhood of point 1. Going back to point 1, since the density of the next member in T_1 increases, point 6 is also in its neighborhood. Hence, the neighborhood of point 1 begins with its direct and indirect neighbors, and then extends. In the last step, NC applies a mutual connectivity test to members of the neighborhoods to find the final neighborhood for each point. The final neighborhood of point 1 is $\{2, 3, 4, 5, 6\}$.

The closure construction step of NC algorithm is illustrated in Figure 4.2. The final neighborhood of point 2 is $\{1, 3, 4, 5, 6\}$ as can be seen in Figure 4.2 (a), and the neighborhood of point 6 is $\{1, 2, 3, 4, 5\}$ as in Figure 4.2 (b). Since the neighborhoods have common points Figure 4.2 (c), these neighborhoods are merged and a closure is obtained as in Figure 4.2 (d).

The main advantage of the NC algorithm is to be a parameter-free neighborhood construction algorithm. NC works well on spatial data sets having arbitrary shapes, density differences within and between clusters. Our proposed evolutionary algorithm uses closures as input. We assume that, as the points in a closure are connected, they belong to the same cluster. Then, instead of individual points, the evolutionary algorithm links the closures to find the clusters. There are two main contributions that NC provides to our proposed algorithm. Firstly, since NC uses connectivity information of data points to construct closures, the resulting clustering solutions are also connected. Therefore, as one of the multiple objectives of clustering, the connectivity objective is achieved by using NC as a preprocessor for MOCNC. Secondly, the problem becomes more scalable since the number of closures is much less than the number of data points.

Further details of NC including its pseudocode can be found in İnkaya (2011) and İnkaya et al. (2015a).

4.2 The Nondominated Sorting Genetic Algorithm II (NSGA-II)

NSGA-II (Deb et al. 2002) is an improved version of Nondominated Sorting Genetic Algorithm (NSGA) proposed by Srinivas and Deb (1994). NSGA-II is a multi-objective evolutionary algorithm, which finds the nondominated solutions. In each generation, a mating pool is constructed from the current population. Then, offsprings are generated by applying crossover to the mating pool, followed by mutation of offsprings. The algorithm evaluates the parent and child populations together. It finds subsets of solutions that belong to a number of successive nondominated fronts and ranks them accordingly (hence the name nondominated sorting). The rank of an individual corresponds to the rank of the front it belongs. Moreover, NSGA-II uses a kind of density measure of the front. The crowding distance of each individual solution is the average of the distances between the individual and its closest neighbors in the front. Individuals that have better ranks are transferred to the next generation until the population is filled. For the final front, in case of equal ranks, the individual that has a larger crowding distance enters the population. The algorithm uses the crowding distance to provide diverse solutions in the front.

4.3 Overview of the MOCNC Algorithm

The MOCNC algorithm has six steps. The flowchart of the algorithm is presented in Figure 4.3. In Step0, the data set is read and the algorithm parameters are initialized. Then, the NC closures are generated and their compactness values are calculated in Step1. In MOCNC the genes in a chromosome (decision variables) are the links connecting pairs of closures instead of pairs of points. The points in closures are assumed to form subclusters, which are fixed. Hence, the compactness values of closures are also fixed and can be used to calculate compactness of the clustering solution. Step2 is the preprocessing step for the main loop of MOCNC. The pairwise distances between closures are calculated and the nearest neighbors of each closure are found. In Step3 the initial population is generated. Step4 is the evolution step of the algorithm. In this step, for each generation the following evolutionary algorithm operations are applied. The mating pool is formed. Crossover is applied to the mating pool and the offspring population is generated. Then, mutation is applied to offsprings. The fitness calculations are made for the two objectives, namely compactness and separation. Then, the nondominated sorting is applied to the union of parent and child populations to obtain the new population. Finally, in Step5 a simple improvement process is applied to the final population.

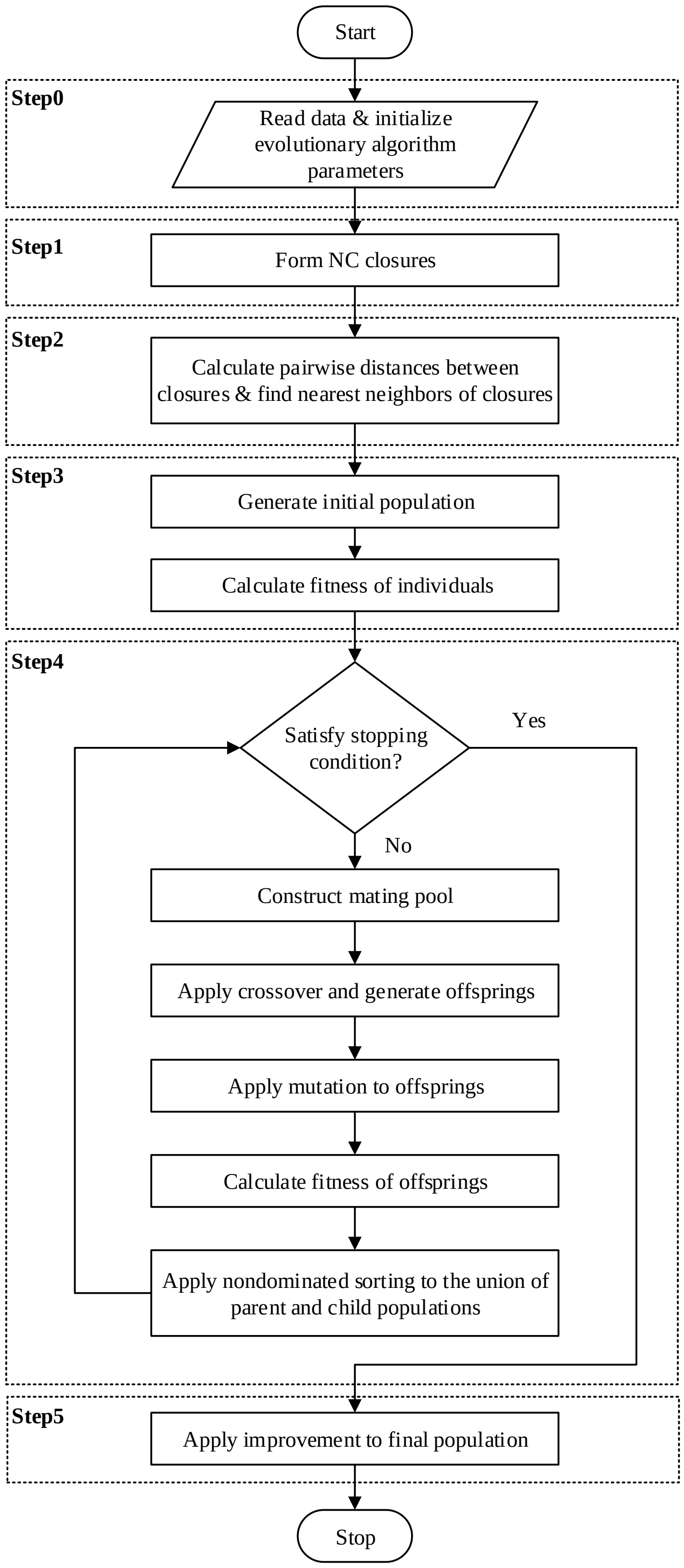

Start
Step0
Read data & initialize evolutionary algorithm parameters
Step1
Form NC closures
Step2
Calculate pairwise distances between closures & find nearest neighbors of closures
Step3
Generate initial population
Calculate fitness of individuals
Step4
Satisfy stopping condition?
Yes
No
Construct mating pool
Apply crossover and generate offsprings
Apply mutation to offsprings
Calculate fitness of offsprings
Apply nondominated sorting to the union of parent and child populations
Step5
Apply improvement to final population
Stop

Figure 4.3: Flowchart of the MOCNC algorithm

4.4 Description of the MOCNC Algorithm

In this section, first the notation used for the clustering problem and the MOCNC algorithm are presented. Then, the steps summarized in the overview of the algorithm will be explained in detail.

4.4.1 Notation for the Clustering Problem and the Objective Functions

We use the following notation to represent the clustering problem and the objective functions.

D	set of data points
C_k	set of points in cluster k
B	set of closures
B_k	set of closures in cluster k
E_m	set of points in closure m
F	set of features
N	number of data points in set D
$numK$	number of clusters found in a solution
$numB$	number of closures in set D
$numF$	number of features in set D
i,j	indices for data points, $i, j = 1, ..., N$
k,l	indices for clusters, $k, l = 1, ..., numK$
m,n	indices for closures, $m, n = 1, ..., numB$
f	indice for features, $f = 1, ..., numF$
d_{ij}	Euclidean distance between points i and j
c_k	centroid of cluster k
c_m	centroid of closure m
n_m	number of points in closure m
$comp_m$	compactness of closure m
$comp_k$	compactness of cluster k

4.4.2 Notation for the Evolutionary Algorithm

We use the following notation to describe our evolutionary algorithm.

$cpop$	population size of clustering solutions
$cgen$	number of generations in evolution
s	indice for individual solutions, $s = 1, ..., cpop$
t	indice for generations, $t = 1, ..., cgen$
P^t	parent population at generation t
P^t_s	solution (chromosome) s of parent population at generation t
$P^t_s(m)$	mth gene in solution s of parent population at generation t
M^t	mating pool at generation t
O^t	offspring population at generation t
O^t_s	solution (chromosome) s of offspring population at generation t
$O^t_s(m)$	mth gene in solution s of offspring population at generation t
ind_s	individual solution (chromosome) s
$cpcross$	crossover probability for clustering solutions
$cpmut$	mutation probability for clustering solutions
$fcomp_s$	compactness of clustering solution s
$fsep_s$	separation of clustering solution s

4.4.3 Fitness Functions Used for the Clustering Problem

For the clustering problem, we select the two fundamental objectives of clustering, compactness and separation of clusters. The selected compactness measure is a representative based compactness measure. It is computed as the overall average of distances between data points in clusters and the corresponding cluster centroids. The compactness measure is referred to as $CCrA$ throughout the study where C stands for compactness, Cr for centroid-based, and A for the average. The compactness measure is formulated as follows.

$$CCrA = \frac{1}{N} \sum_{k=1}^{numK} \sum_{i \in C_k} d^2_{ic_k} \tag{4.1}$$

The closures and their compactness values are fixed throughout the algorithm. Therefore, the original compactness measure in Equation 4.1 can be calculated using Equation 4.2. The proof of equivalence of the two Equations (4.1 and 4.2) is given in Appendix A.

$$CCrA = \frac{1}{N}\left(\sum_{k=1}^{numK}\sum_{m\in B_k} n_m * (comp_m + d^2_{c_k c_m})\right) \tag{4.2}$$

where $comp_m$ denotes the compactness of closure m and is calculated as follows.

$$comp_m = \frac{1}{n_m}\sum_{i=1}^{n_m} d^2_{ic_m}$$

Computation of compactness of a clustering solution has two steps: (i) the compactness values of closures are calculated, and (ii) the overall compactness of the clustering solution is calculated by using the compactness of closures and the distances between pairs of closures. The calculation steps of closure compactness are shown in Algorithm 1. The data points in closures are given. The distance between each data point and the closure centroid is calculated and summed (lines 2-3). Then the summed distance is divided by the number of points in the closure. The output is the compactness of a closure.

Algorithm 1: Computation of closure compactness

```
Procedure pre_CCrA()
    input  : $c_m$, $n_m$
    output: $comp_m$
1   for $i = 1, ..., n_m$
2       Calculate $d^2_{ic_m}$
3       $comp_m = comp_m + d^2_{ic_m}$
4   end for
5   $comp_m = comp_m / n_m$
end
```

To calculate the overall compactness, the number of clusters in the solution, the set of closures in clusters, and closure compactness values are given as input to Algorithm 2. For each cluster, firstly the cluster centroid is calculated (lines 2-5), then the compactness of the cluster is calculated by using the compactness of closures in that cluster and the distances between cluster and closure centroids (lines 6-8). Compactness values of all clusters are summed and divided by the total number of data points. The output is the compactness measure for a given clustering solution.

Algorithm 2: Computation of compactness objective function

```
   Procedure CCrA()
      input  : $numK$, $B_k \; \forall k$, $c_m, n_m, comp_m \; \forall m$
      output : $fcomp$
1     for $k = 1, ..., numK$
2         for $m = 1, ..., |B_k|$
3             $c_k = c_k + n_m * c_m$
4         end for
5         $c_k = c_k / |C_k|$
6         for $m = 1, ..., |B_k|$
7             $comp_k = n_m * (comp_m + d^2_{c_k c_m})$
8         end for
9         $fcomp = fcomp + comp_k$
10    end for
11    $fcomp = fcomp / N$
   end
```

The separation objective function is the single-link linkage metric. It calculates the minimum distance between the closest points belonging to two different clusters. The separation measure is named as SPM where S stands for separation, P for pointwise and M for minimum. Similar to the compactness measure, separation is also calculated using closures in MOCNC. The separation measure is formulated as follows.

$$SPM = min\{d_{ij} : \forall i \in E_m, \forall j \in E_n, \forall m \in B_k, \forall n \in B_l, k \neq l\} \tag{4.3}$$

The calculation steps of separation measure are shown in Algorithm 3. The number of clusters in a solution and the set of closures in each cluster are given as input. For each cluster pair, the distance between points of closures in those clusters is calculated. The minimum of these distances is the separation for the clustering solution. The output of the separation function is the separation value and the cluster pair that gives the separation value. The cluster pair will then be used in the mutation operator.

Algorithm 3: Computation of separation objective function

```
Procedure SPM()
    input : $numK, B_k \; \forall k$
    output: $fsep, k^*, l^*$
    Let $min$ = a very large number
    for $k = 1, ..., numK - 1$
        for $l = k, ..., numK$
            for $m = 1, ..., |B_k|$
                for $n = 1, ..., |B_l|$
                    Calculate $d_{mn}$
                    if $d_{mn} < min$ then
                        $min = d_{mn}$
                        $k^* = k$
                        $l^* = l$
                    end
                end for
            end for
        end for
    end for
    $fsep = min$
end
```

4.4.4 Chromosome Representation

Our chromosome structure consists of two parts: (i) the decision variables (genes) and (ii) the fitness values (compactness and separation). In our chromosomes, the genes correspond to the links between pairs of NC closures. We use the adjacency-based representation (Park and Song 1998). The number of genes is the number of NC closures in the data set to begin with. The mth gene corresponds to the mth closure and it can take a value between 1 and $numB$. Let the value of the mth gene be n. Then, the mth closure is linked (or connected) to the nth closure, and these closures are in the same cluster. The adjacency-based representation is decoded to obtain the clustering solution. In the resulting clustering solution, all connected closures will be in the same cluster. The unconnected groups of closures form different clusters. An example of adjacency-based representation decoding is illustrated in Figure 4.4.

In the example, there is a chromosome with six genes representing closures 1-6. In Figure 4.4(a) the genotype of the chromosome is represented. The first closure is linked to closure 2, the second closure to closure 3, and so on. The phenotype of the

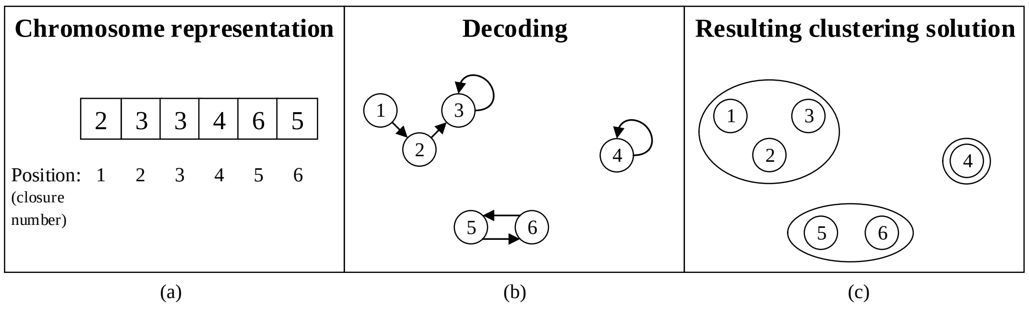


Figure 4.4: Illustration of the adjacency-based representation

chromosome is represented in Figure 4.4(b). After the decoding process, the resulting clustering solution is shown in 4.4(c). In the solution, closures 1, 2 and 3 are in the same cluster. Similarly, closures 5 and 6 are in the same cluster. Closure 4 forms a cluster on its own.

This representation is advantageous for the clustering problem since it does not require the number of clusters a priori. During the decoding process, without knowing the number of clusters the resulting clustering solution is constructed. Moreover, this enables to compare different clustering solutions with different number of clusters at the end of the algorithm.

4.4.5 Initial Population Generation

Our initialization routine consists of two types of solutions, random solutions and seed solutions. We generate almost all the initial solutions randomly, and to improve the performance of the algorithm, we generate one or two seed solutions depending on the dimension of the data set.

In all data sets, we generate a seed solution in which each closure is connected to its closest neighbor closure. When the dimensionality of the data set is greater than or equal to 10, we generate an additional seed solution. In this seed solution, all closures are connected to themselves.

The main aim of these seed solutions is to give the algorithm the chance to visit every solution in the search space, i.e. increase the reachability. Since the mutation operator of MOCNC behaves towards merging the closest clusters, the seed solutions enable to increase the percentage of searched space.

The steps of the initialization routine is shown in Algorithm 4. If the number of features is less than 10, the first solution of the population is generated by connecting each closure to its nearest neighbor (lines 2-4). Remaining solutions of the initial population are generated randomly (lines 5-10). Otherwise, if the dimensionality is greater than or equal to 10, the first solution of the initial population is generated by assigning each closure to itself (lines 12-14). The second solution is generated by assigning each closure to its nearest neighbor (lines 15-17). Finally, the remaining ones are generated randomly (lines 18-23).

Algorithm 4: Initial population generation

```
   Procedure generate_initial_population()
       input  : $numF, numB, cpop$
       output: $P^0$
1      if $numF < 10$ then
2          for $m = 1, ..., numB$
3              $P_1^0(m)$ = nearest neighbor of closure $m$
4          end for
5          for $s = 2, ..., cpop$
6              for $m = 1, ..., numB$
7                  $P_s^0(m)$ = discrete uniform $[1, numB]$
8              end for
9          end for
10     else
11         for $m = 1, ..., numB$
12             $P_1^0(m) = m$
13         end for
14         for $m = 1, ..., numB$
15             $P_2^0(m)$ = nearest neighbor of closure $m$
16         end for
17         for $s = 3, ..., cpop$
18             for $m = 1, ..., numB$
19                 $P_s^0(m)$ = discrete uniform $[1, numB]$
20             end for
21         end for
22     end
   end
```

4.4.6 Parent Selection and Mating Pool Construction

In NSGA-II the selection of parents to construct a mating pool is based on ranks and crowding distances of individual solutions. The rank of an individual in the population is the rank of the nondominated front it belongs to. Moreover, the individuals have crowding distance values. Crowding distance is a kind of a density measure for a given solution in its front. As in NSGA-II, MOCNC uses binary tournament selection to form the mating pool. Algorithm 5 represents the pseudocode of the selection process. From the current population, two individuals are selected randomly and compared firstly using their ranks. If the rank of one of the individuals is smaller than the other (indicating a better front), then this solution is put into the mating pool. If the ranks are equal, then the crowding distances of individuals are compared. The individual with the larger crowding distance is placed in the mating pool.

Algorithm 5: Parent selection and mating pool construction

```
Procedure binary_tournament_selection()
    input  : $P^t, cpop$
    output: $M^t$
    for $s = 1, ..., cpop$
        Select $ind_1$ randomly from $P^t$
        Select $ind_2$ randomly from $P^t$
        if rank of $ind_1$ < rank of $ind_2$ then
            $M^t = M^t \cup \{ind_1\}$
        else if rank of $ind_2$ < rank of $ind_1$ then
            $M^t = M^t \cup \{ind_2\}$
        else if crowding distance of $ind_1$ < crowding distance of $ind_2$ then
            $M^t = M^t \cup \{ind_2\}$
        else
            $M^t = M^t \cup \{ind_1\}$
        end
    end for
end
```

4.4.7 The Crossover Operator

As the crossover operator, we choose the uniform crossover. Uniform crossover is an unbiased operator since it is not affected from the ordering of genes in the chromosome. It randomly generates two offsprings from a pair of selected parents. The pseudocode of uniform crossover is provided in Algorithm 6. Crossover is applied with a certain crossover probability. If crossover is applied, two parents are selected from the mating pool and a binary mask for reproduction is generated randomly (lines 5-7). According to the mask, two offsprings are generated and placed in the offspring population (lines 8-16). Otherwise, if crossover is not applied, then the selected two parents will be copied as the offsprings (lines 18-19). The selected crossover operator randomly searches the solution space which has $numB^{numB}$ solutions for a data set with $numB$ closures.

Algorithm 6: Crossover

```
    Procedure uniform_crossover()
        input  : $M^t, cpop$
        output: $O^t$
1       $s = 1$
2       while $s < cpop$ do
3           $rnd = U[0, 1]$
4           if $rnd \leq cpcross$ then
5               Create a mask of size $numB$ consisting of randomly generated 0s
                and 1s
6               for $m = 1, ..., numB$
7                   if $mask(m) = 0$ then
8                       $O^t_s(m) = P^t_s(m)$
9                       $O^t_{s+1}(m) = P^t_{s+1}(m)$
10                  else
11                      $O^t_s(m) = P^t_{s+1}(m)$
12                      $O^t_{s+1}(m) = P^t_s(m)$
13                  end
14              end for
15          else
16              $O^t_s = P^t_s$
17              $O^t_{s+1} = P^t_{s+1}$
18          end
19          $s = s + 2$
20      end
    end
```

4.4.8 The Mutation Operator

In our algorithm, we propose a mutation operator for the clustering problem, which is named as $mutation\_SPM$. The main objective of the mutation operator is to search for a nondominated solution by merging the two closest closures (or subclusters obtained by previous merging of closures). With the uniform crossover new clustering solutions are obtained randomly by combining or dividing subclusters. The mutation operator is used to selectively combine the subclusters, providing the convergence of the algorithm.

Mutation is applied to an offspring with a mutation probability. The proposed mutation operator is deterministic. If the mutation is to be applied, the two closest subclusters are merged. The pair of subclusters to be merged is determined by the separation objective function.

The mutation steps are outlined in Algorithm 7. The closest subclusters are given as input to the mutation operator. If the mutation probability is realized, the given subclusters are merged by constructing a link between two closures in the corresponding subclusters. The important point here is to not divide the subclusters while constructing a new link between two closures from two different subclusters. To avoid unintentional division of clusters by the mutation operator, during the decoding process, in each cluster one of the closures is linked to itself. Then, in mutation the closure that is linked to itself from the first subcluster is connected to a closure in the second subcluster. This way, merging two subclusters without dividing them is ensured.

Algorithm 7: Mutation

```
Procedure mutation_SPM()
    input  : $O_s^t, k^*, l^*, B_{k^*}, B_{l^*}$
    output: mutated $O_s^t$
    for $m = 1, ..., |B_{k^*}|$
        if Closure $m$ is linked to itself then
            Connect closure $m$ to closure $n \in B_{l^*}$
            break
        end
    end for
end
```

4.4.9 Improvement

Compactness objective function improves as the number of clusters increases. Hence, it has a bias towards the larger number of clusters. On the other hand, separation objective function is not directly affected from the number of clusters. However, it has a tendency to merge the closest clusters to improve its value. Therefore, two clustering solutions may have the same separation value but slightly different compactness values due to different number of clusters. In such a case the solution with fewer clusters is dominated. Moreover, sometimes the additional clusters contain only a few data points. To merge these small clusters to their nearest clusters, we apply an improvement process to the population at the last generation of the algorithm. This improvement is useful in counteracting the bias of the compactness objective.

The improvement process is presented in Algorithm 8. The final population and the closure centroids are given as input. If a solution has a cluster with fewer than three data points, the closures inside this cluster are assigned to their nearest clusters. The distances between the centroid of the closure and the other clusters are calculated. The closure is then assigned to its nearest cluster.

4.4.10 The MOCNC Algorithm

The pseudocode of MOCNC is given in Algorithm 9. The steps are named as initialization, finding NC closures and their compactness values, pairwise distance calculation and finding nearest neighbors, initial population generation, evolution, and improvement. The step numbers are the same as in Figure 4.3.

Before starting the main loop of the algorithm in Step0, the data set is given and the algorithm parameters (the population size, the number of generations, probability of crossover and probability of mutation) are initialized (lines 1-3).

After the initialization process, the next step is finding NC closures and their compactness values (Step1). The NC algorithm takes the data set and generates the NC closures. After finding the closures, their compactness values are calculated based on the compactness objective function, $CCrA$. Since the closures are fixed throughout

Algorithm 8: Improvement

```
Procedure Improvement()
    input  : $P^{cgen}, c_m \; \forall m$
    output: $P^{cgen}$
 1  Let $min$ = a very large number
 2  for $s = 1, ..., cpop$
 3      $numK$ = number of clusters in $P_s^{cgen}$
 4      for $k = 1, ..., numK$
 5          if $|C_k| < 3$ then
 6              for $m = 1, ..., |B_k|$
 7                  for $l = 1, ..., numK (k \neq l)$
 8                      Find $c_l$
 9                      Calculate $d_{c_m c_l}$
10                      if $d_{c_m c_l} < min$ then
11                          $min = d_{c_m c_l}$
12                          $l^* = l$
13                      end
14                  end for
15                  Connect closure $m$ to cluster $l^*$
16              end for
17          end
18      end for
19  end for
end
```

the algorithm, their compactness values are also fixed. Hence, to increase the efficiency of the algorithm we use the compactness of closures in calculating the overall compactness of a clustering solution instead of using individual points.

The next step (Step2) is a preprocessing step for the evolutionary algorithm. The pairwise distances between closures are calculated to be used afterwards in the separation objective function calculations. The nearest neighbor for each closure is also found to be used in the initial population generation.

Step3 and Step4 are the evolutionary algorithm steps. In Step3 the initial population is generated (line 14). The initial population consists of mostly randomly generated solutions and oner or two seed solutions. The seed solutions are generated to increase the reachability for the evolutionary algorithm. After initial population generation, the fitness values of each individual is calculated (lines 15-18).

In Step4 the generations of the algorithm evolve starting with the initial population.

In each generation, firstly the parent selection procedure is applied. We use the binary tournament selection as the selection operator (line 21). The output of the selection process is the mating pool. Next, the uniform crossover is applied to the parents in the mating pool and the offsprings are generated (line 22). The output of the crossover process is the offspring population. For each individual in the this population the separation objective function is calculated to find the cluster pair that gives the separation value of the clustering solution (line 24). The offsprings is then subjected to the mutation process. The mutation operator merges the clusters that give the separation value (line 25). After the mutation, compactness and separation of the offspring are recalculated (lines 26-27). After fitness values are found for all offsprings, nondominated sorting is applied to the union of parent and offspring populations (line 29). At the end of nondominated sorting a new population is generated and the next generation starts.

The last step (Step5) of the proposed algorithm is the improvement step. The improvement process is applied to the final population. The clustering solutions having clusters with fewer than three data points are connected to their nearest clusters (line 32).

Algorithm 9: The MOCNC Algorithm

```
Procedure MOCNC()
    input  : Data set $D$
    output: Nondominated clustering solutions
    Step0. Initialization
    Set $cpop, cgen, cpcross, cpmut$
    Read data set
    Step1. Find NC closures and their compactness
    Call NC() $\rightarrow B$
    for $m = 1, ..., numB$
        Find $c_m, n_m$
        Call pre_CCrA() $\rightarrow comp_m$
    end for
    Step2. Find pairwise distances and nearest neighbors of closures
    Calculate pairwise distances between closures
    Find the nearest neighbor of each closure
    Step3. Initial population generation
    Call generate_initial_population() $\rightarrow P^0$
    for $s = 1, ..., cpop$
        $fcomp_s \leftarrow$ CCrA()
        $fsep_s \leftarrow$ SPM()
    end for
    Step4. Evolution
    for $t = 1, ..., cgen$
        Call binary_tournament_selection() for $P^{t-1} \rightarrow M^t$
        Call uniform_crossover() for $M^t \rightarrow O^t$
        for $s = 1, ..., cpop$
            Call SPM() for $O_s^t \rightarrow (fsep_s, k^*, l^*)$
            Apply mutation_SPM() to $O_s^t$ using $k^*$ and $l^*$
            $fcomp_s \leftarrow$ CCrA()
            $fsep_s \leftarrow$ SPM()
        end for
        Apply nondominated sorting to $(P^{t-1} \cup O^t)$ to obtain $P^t$
    end for
    Step5. Improvement
    Call improvement() for $P^{cgen} \rightarrow P^{cgen}$
end
```

CHAPTER 5

COMPUTATIONAL RESULTS OF MOCNC

In this chapter, we give the empirical results obtained with MOCNC. In Section 5.1, we describe the data sets used in the computational study. Section 5.2 gives the parameter settings and performance measures. We introduce the benchmark approaches used for comparison in Section 5.3. Finally, the experimental results and related discussion are given in Section 5.4.

5.1 Data Sets

In our computational experiments, we used two different groups of data sets. For the first group we generated 2- or higher dimensional data sets using the Gaussian data set generator proposed in Handl and Knowles (2005b). We preserved the main principles of the data generator but except modifying the trial-and-error scheme considering our problem. In the original data set generator, the data sets are generated from multivariate normal distributions. Given the number of clusters, the mean and the size of each cluster are generated uniformly in a user defined range. Then, the variance-covariance matrix for feature values of each cluster is constructed, which needs to be symmetrical and positive definite. By using the mean, the size of the cluster, and the variance-covariance matrix, the data points for a cluster are generated from the multivariate normal distribution. The generator iteratively generates clusters until reaching the user defined number of clusters, and rejects a newly generated cluster if it has an overlap with previously generated clusters.

We preserved the random generation principles of the mean, the size of the cluster, and the variance-covariance matrix. However, we changed the rejection condition of a newly generated cluster. Since we use the NC closures in our algorithm as an input, data sets should fit the principle of the NC algorithm, namely the clusters should be crisply separated. In the NC algorithm, by using the ordered set of points for a data point a unique neighborhood for this point is formed. If this neighborhood includes points from different clusters, then the resulting closures may also contain points from different clusters. Hence, the clustering solutions of the MOCNC algorithm also have the characteristics of the closures. Therefore, the data generator checks clusters for their compatibility with the NC algorithm. To guarantee pure NC closures, each direct neighbor of a point should be from the same cluster. Hence, when a cluster is generated, firstly the data points in this cluster are checked to see if their direct neighbors belong to the same cluster. If this condition is not satisfied, then the cluster is rejected and a new cluster generation begins. Otherwise, if the condition for the new cluster holds, the previously generated clusters are checked for this condition. In the care of the condition does not hold for any of the previously generated clusters, then the newly generated cluster is again rejected and a new cluster generation begins. Generation continues until the valid set of clusters are generated.

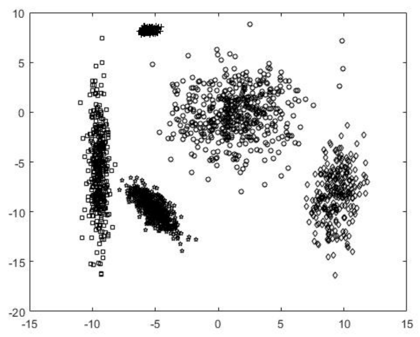
Using this data generator, 150 different data sets were generated with 15 different problem settings, each problem setting has 10 different instances. The problem settings are various combinations of the number of clusters and the number of features. Data sets have 5, 10, or 20 clusters. The number of features is 2, 5, 10, 20, or 40. The number of points in each cluster is generated from Uniform $[50, 500]$. The cluster means are generated within a range that is proportional with the number of features and the number of clusters.

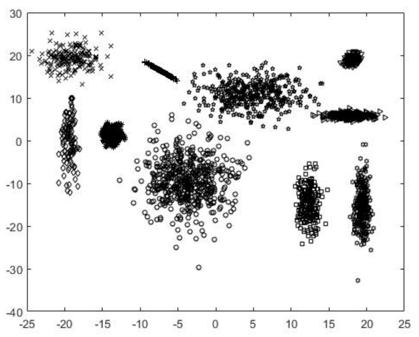
The data set properties are summarized in Table 5.1. Each row in the table corresponds to a problem setting for a combination of dimensionality and the number of clusters. The third column of the table presents the range for cluster means. The last three columns are for the minimum, the average, and the maximum number of data points across 10 instances for a problem setting.

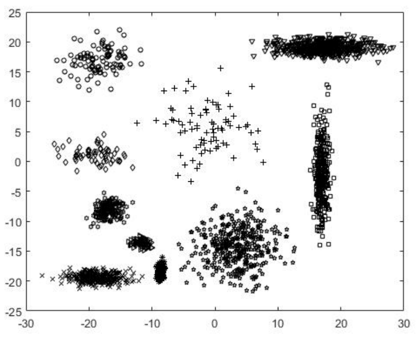
Some examples from the first group of data sets are given in Figure 5.1. For example, 2d-10c-7 refers to the 7^{th} data set instance having 2 features and 10 clusters. It is shown Figure 5.1(b).

Table 5.1: Summary of randomly generated data sets

Number of Features	**Number of Clusters**	**Range for Cluster Means**	**Number of Data Points**		
			Min	**Avg**	**Max**
2	5	[-10, 10]	942	1467.1	1921
2	10	[-20, 20]	2305	3095.2	3627
2	20	[-40, 40]	4235	5657.5	6810
5	5	[-10, 10]	1097	1561.2	1999
5	10	[-20, 20]	2403	2822.8	3687
5	20	[-40, 40]	4615	6073.2	6947
10	5	[-10, 10]	725	1460.4	2070
10	10	[-20, 20]	2285	3114.5	4121
10	20	[-40, 40]	4634	5964.0	7039
20	5	[-15, 15]	804	1405.3	2402
20	10	[-20, 20]	2490	2971.5	3609
20	20	[-40, 40]	4757	5752.1	6582
40	5	[-15, 15]	947	1557.0	2039
40	10	[-20, 20]	2253	3098.1	3878
40	20	[-40, 40]	5283	5973.1	6746


(a) 2d-5c-5


(b) 2d-10c-7


(c) 2d-10c-8

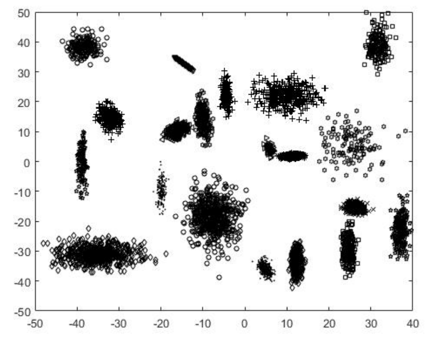

(d) 2d-20c-10

Figure 5.1: Examples of randomly generated data sets

Table 5.2: Parameter setting of the MOCNC algorithm

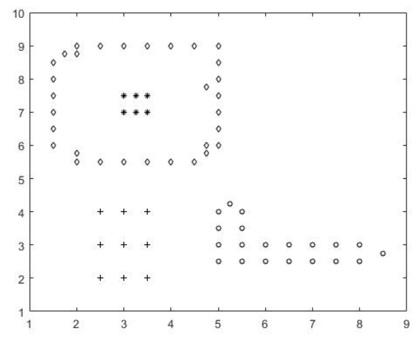
Parameter	**Value**	
Dimension	{2, 5}	{10, 20, 40}
Population size	100	100
Number of generations	100	200
Crossover probability	1	1
Mutation probability	1	1

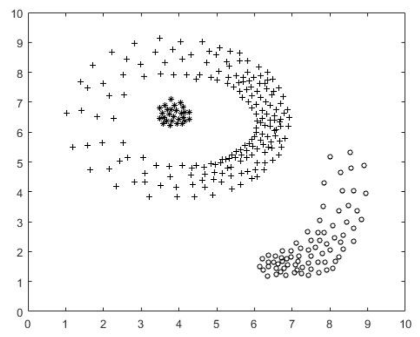
The second group of data sets contains 2-dimensional data sets. These include clusters with inter-cluster and intra-cluster density differences, and arbitrary shapes. These data sets are from the literature that were also used in İnkaya (2011), İnkaya et al. (2015a), and Handl and Knowles (2007). Some of the data sets from this group are shown in Figure 5.2.

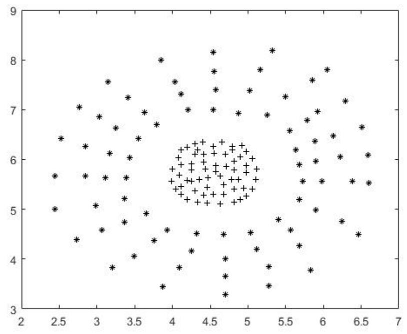
5.2 Parameter Settings and Performance Measures

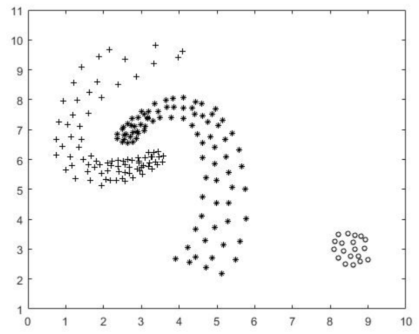
The parameter list of MOCNC are the same as those of NSGA-II (Deb et al. 2002). We give the parameter settings used for MOCNC in Table 5.2. These final values are determined based on numerous pilot runs. To set the number of generations, we ran the algorithm up to 400 generations, and at every 50 generations we analyzed the convergence behavior of the algorithm. We observed that the dimensionality hence the number of seed solutions affected the convergence of the algorithm. Therefore, we set different number of generations for different dimensions.

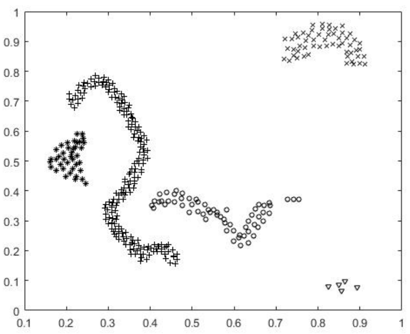
To evaluate the clustering performance of our algorithm, we use the Rand Index (RI) and the Adjusted Rand Index (ARI), which are widely used external validity indices that measure the clustering quality. These two measures mainly penalize the mixing and the division of clusters. The RI takes two partitionings as input, which are the resulting clustering solution and the true cluster labels, and evaluates the similarity between them. In ARI a statistical correction is introduced to RI to exclude the effect of randomness in successful clustering. The RI can take values in the range $[0, 1]$, whereas the ARI can also take negative values. The larger the RI and ARI values, the closer the solution to the target clustering. As the ARI value gets close to 0, the clustering solution essentially becomes a random partition. A negative value of ARI

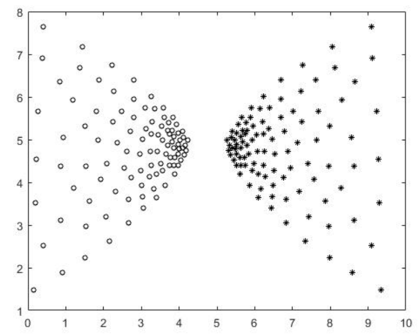

(a) data_66

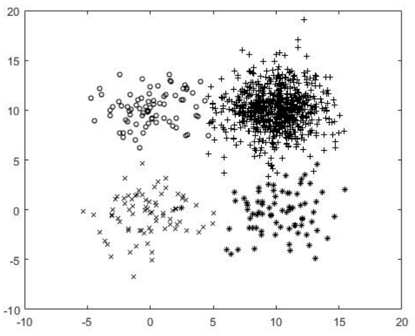

(b) data-c-cc-nu-n_v2


(c) data-oo_v2


(d) data-uc-cc-nu-n_v2


(e) train3_v1


(f) dataX_v2


(g) size5

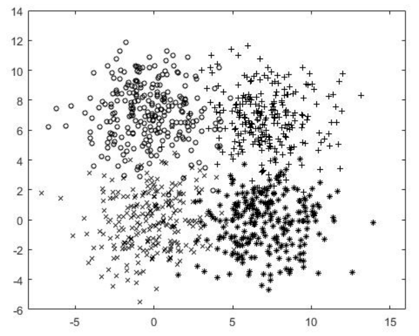

(h) square4

Figure 5.2: Examples of benchmark data sets

means that the solution is worse than a random partition. The RI is defined as in Equation 5.1, following the notation.

Let A be the resulting clustering solution and B be the true cluster labels of the data set.

a: number of point pairs that are in the same cluster in A and the same cluster in B
b: number of point pairs that are in different clusters in A and different clusters in B
c: number of point pairs that are in the same cluster in A but different clusters in B
d: number of point pairs that are in different clusters in A but the same cluster in B

$$RI = \frac{a+b}{a+b+c+d} \tag{5.1}$$

The ARI is defined as in Equation 5.2 based on the contingency table of the two partitions A and B.

$$ARI = \frac{\sum_{kl}\binom{n_{kl}}{2} - \left[\sum_k \binom{n_{k.}}{2} * \sum_l \binom{n_{.l}}{2}\right] / \binom{N}{2}}{\frac{1}{2}\left[\sum_k \binom{n_{k.}}{2} + \sum_l \binom{n_{.l}}{2}\right] - \left[\sum_k \binom{n_{k.}}{2} * \sum_l \binom{n_{.l}}{2}\right] / \binom{N}{2}} \tag{5.2}$$

where n_{kl} is the number data points that are in common in cluster k of partition A and cluster l of partition B, and N is the number of data points in the data set. A period used as a subscript means that the numbers of points are summed up across that index.

5.3 Benchmark Clustering Methods Used for Comparison

In our experiments, we compare our algorithm with two traditional single-objective clustering algorithms, *k-means* and *single-link* clustering, and a multi-objective genetic algorithm for clustering, Δ-MOCK, which was proposed by Garza-Fabre et al. (2017).

k-means (MacQueen et al. 1967) and *single-link* clustering (Voorhees 1985) are well-known single-objective clustering algorithms. *k-means* is a partitional clustering algorithm that aims to minimize the sum of intra-cluster distances. *Single-link* clustering is a hierarchical clustering algorithm aiming to maximize the minimum distance between the closest points from two different clusters in every iteration of the algorithm. These two clustering algorithms are selected intentionally for comparison since their objectives are one of the objective functions of our algorithm. Comparing MOCNC with these two clustering algorithms shows the advantages of multi-objective clustering.

k-means is run for a range of different number of clusters. Similarly, the resulting dendrogram of *single-link* clustering is cut at different number of clusters. The range for the number of clusters for both algorithms is set as $\{1, 2, ..., 2k^*\}$ where k^* is the true number of clusters. For each data set, the clustering solution that has the best RI and ARI values and the corresponding number of clusters are reported.

Δ-MOCK is an improved version of MOCK proposed by Handl and Knowles (2007). Similar to our algorithm, Δ-MOCK uses the nondominated sorting of NSGA-II, but with different objective functions for clustering, namely compactness and connectivity. Their compactness objective function is the same as the one used in this thesis. As the second objective, they use a penalization based connectivity objective function as in Equation 5.3. In this objective function, a user-defined neighborhood parameter, L is used. If a point and its lth nearest neighbor, where $l \leq L$, are in different clusters, then the amount of penalty added to the connectivity objective function is $\frac{1}{l}$.

$$\sum_{i=1}^{N} \sum_{l=1}^{L} x_{inn_l^i} \tag{5.3}$$

where nn_l^i is the lth nearest neighbor of point i, and

$$\mathrm{x}_{inn_l^i} = \begin{cases} \frac{1}{l} & if\ i \text{ and } nn_l^i \text{ are in different clusters} \\ 0 & otherwise. \end{cases}$$

Δ-MOCK uses adjacency-based chromosome representation to represent the clustering solutions as in MOCNC, but their genes correspond to individual data points. To decrease the computation time, it uses a reduced length chromosome throughout the genetic algorithm. A user-defined parameter, δ determines the length of the chromosome. As a preprocessing step, a minimum spanning tree (MST) is constructed for the entire data set, and the resulting links are sorted in descending order of their 'interestingness' values. Then, the top $(1-\delta)\%$ of the links are assumed to be relevant links, and the remaining $\delta\%$ of the links are assumed to be non-relevant and fixed throughout the algorithm. The relevant links form the chromosomes, and the genetic operators are only applied to these links. This reduced chromosome length also decreases the computation time of the objective function calculations.

This algorithm uses a MST based initial population generation scheme. In the initial population, individual clustering solutions are generated by randomly breaking $(k-1)$ relevant links from the constructed full MST, where $k = 1, ..., k_{max}$ and k_{max} is also a user-defined parameter. The value of k_{max} is taken as twice the true number of clusters. This may be regarded as the major disadvantage of Δ-MOCK. For parent selection and crossover operators, Δ-MOCK uses the same operators as in MOCNC. It has a neighborhood-biased mutation operator, in which neighborhood parameter, L, needs to be defined. In the mutation, if a link is broken with the mutation probability, then the point is linked to a point within its L nearest neighborhood.

Since both Δ-MOCK and MOCNC are based on the NSGA-II algorithm, they are comparable in terms of different objective function pairs, chromosome representations, initialization routines, and genetic operators. However, Δ-MOCK has more parameters to be set by the user. The parameters used for Δ-MOCK are set as in Table 5.3 as proposed in Garza-Fabre et al. (2017).

5.4 Computational Results for Generated Data Sets

The NC and MOCNC algorithms were coded in C and all the experiments were conducted on a PC with a 3.6 GHz 4-Core Intel Core i7-4790 processor and 8GB of RAM.

MOCNC, Δ-MOCK, *k-means*, and *single-link* clustering algorithms are applied to

Table 5.3: Parameter setting of the Δ-MOCK algorithm

Parameter	**Value**
Population size	100
Number of generations	100
Crossover probability	1
Mutation probability	$\frac{1}{\Gamma} + (\frac{nn_l^i}{\Gamma})^2$
k_{max}	$2k^*$
Neighborhood parameter (L)	10
Encoding-length parameter (δ)	80

k^* is the true number of clusters in the data set.
Γ is the number of relevant links hence the chromosome length.

both groups of data sets described in Section 5.1. The detailed results for generated data sets are given in Tables B.1 - B.5 see Appendix B. The summary of computational results for generated data sets is given in Table 5.4.

The first column of the Table 5.4 presents the problem settings. The problem settings are named based on the number of features and the number of clusters in the data set. For example, 2d-5c refers to the data sets having 2 features and 5 clusters. For each problem setting the values in the respective rows are the averages of 10 problem instances. For each problem instance, from the final population of nondominated solutions, the one that gives the best RI and ARI values is taken as the solution. For MOCNC and Δ-MOCK algorithms, the averages of the RI and ARI values in the final population across 10 instances are reported, as well as the corresponding average number of clusters (k). Since *k-means* and *single-link* clustering algorithms require the number of clusters to be known, a range of number of clusters are run for each data set. Then, the clustering solution that gives the best RI and ARI values is taken as the solution for the data set, and the average of 10 problem instances for each problem setting is reported.

As we expect, *k-means* and *single-link* clustering could not find the target clustering solutions in most of the problem settings. Out of 15 problem settings, *single-link* clustering has the worst RI and ARI performance in 9 settings, and *k-means* shows the worst performance in the remaining 6 settings. Although RI and ARI values of *k-means* and *single-link* clustering are not very low, the number of clusters found are not close to the target number of clusters. Mostly, the resulting number of clusters are higher than the target number of clusters. In all the problem settings, the best performers are the MOCNC and Δ-MOCK algorithms (in 2 problem settings *single-link* clustering also finds the target clustering as MOCNC and Δ-MOCK). We performed the 2-sample t-test at 5% significance level to see if the differences in average RI and ARI values between the two algorithms are statistically significant. When we compare the RI and ARI values of 150 data sets, Δ-MOCK shows better performance compared to MOCNC. Moreover, we analyzed the individual problem settings for the significance of difference between mean values of RI and ARI values (10 instances for each problem setting are compared for statistical significance). MOCNC and Δ-MOCK find the target clusters in 9 and 13 problem settings, respectively. The problem settings that Δ-MOCK finds target clusters covers those that MOCNC find. Out of the remaining 6 problem settings for which MOCNC could not find the target clusters, except the problem settings 5d-10c and 5d-20c, there is no statistically significant difference between the two algorithms. The Δ-MOCK algorithm performs slightly better in 5d-10c and 5d-20c problems.

The computational times of the MOCNC algorithm and the clustering methods used for comparison are summarized in Table 5.5. We present both the total execution time of MOCNC and its components which are NC and the genetic algorithm (GA). As the dimension increases, the shares of the two components in the total time begin to change. In 2, 5 and 10 dimensional data sets more than 90% of the total computing time is for the NC execution whereas in 20 dimensional data sets almost 60% of the total time is for the NC execution and the remaining computing time is for the GA execution. In 40 dimensional data sets, the NC constitutes only 13% of the total computing time whereas the remaining is due to GA. The NC execution times increase proportional to the size of the data set and the dimensionality. The GA times are mainly affected by the number of closures since the chromosome length is determined by the number of closures. Moreover, as the dimensionality increases, the GA execution times increase due to the number of features used in fitness calculations. The contribution of the final improvement process is negligible.

In problem settings, the execution time of Δ-MOCK is less than the computing time of MOCNC. The Δ-MOCK algorithm uses two significant user-defined parameters to increase the efficiency and reduce the computation time, which are encoding-length

Table 5.4: Solution quality of MOCNC, Δ-MOCK, *k*-means and Single-Link for generated data sets[1]

Problem	**Number of Points**	**Number of NC Closures**	**MOCNC**						**Δ-MOCK**			***k*-means**			**Single-Link**		
			Before Improvement			**After Improvement**											
			RI	*ARI*	k^2	*RI*	*ARI*	k^2	*RI*	*ARI*	k^2	*RI*	*ARI*	k^2	*RI*	*ARI*	k^2
2d-5c	1467.1	60.7	1.000	1.000	5.3	1.000	0.999	5.0	1.000	1.000	5.0	0.962	0.893	5.8	0.931	0.841	7.1
2d-10c	3095.2	132.2	1.000	1.000	10.2	1.000	1.000	10.1	1.000	0.999	10.1	0.987	0.940	9.7	0.976	0.903	14.7
2d-20c	5657.5	246.8	0.999	0.995	19.8	0.999	0.995	19.8	0.999	0.995	19.9	0.992	0.928	21.0	0.990	0.925	28.2
5d-5c	1561.2	31.3	0.999	0.998	6.7	0.999	0.997	5.0	1.000	1.000	5.0	0.979	0.942	5.1	0.811	0.616	8.6
5d-10c	2822.8	59.4	0.993	0.972	9.6	0.992	0.965	9.3	1.000	1.000	10.0	0.991	0.958	9.6	0.918	0.766	16.2
5d-20c	6073.2	133.7	0.998	0.984	18.8	0.998	0.984	18.8	1.000	1.000	20.0	0.996	0.964	19.5	0.998	0.984	26.9
10d-5c	1460.4	24.9	0.998	0.994	11.4	1.000	1.000	5.0	1.000	1.000	5.0	0.986	0.961	5.1	0.601	0.348	9.0
10d-10c	3114.5	54.1	1.000	0.999	13.4	1.000	1.000	10.0	1.000	1.000	10.0	0.989	0.947	10.0	0.934	0.803	12.8
10d-20c	5964.0	104.0	1.000	1.000	21.3	1.000	1.000	20.0	1.000	1.000	20.0	0.996	0.968	20.3	1.000	0.997	20.2
20d-5c	1405.3	50.4	0.999	0.999	6.4	1.000	1.000	5.0	1.000	1.000	5.0	0.980	0.946	5.2	0.879	0.762	6.4
20d-10c	2971.5	110.5	1.000	0.999	11.1	1.000	1.000	10.0	1.000	1.000	10.0	0.989	0.950	10.1	0.996	0.981	12.8
20d-20c	5752.1	220.7	1.000	1.000	20.5	1.000	1.000	20.0	1.000	1.000	20.0	0.995	0.961	20.6	1.000	1.000	20.0
40d-5c	1557.0	150.0	0.999	0.998	6.9	1.000	1.000	5.0	1.000	1.000	5.0	0.967	0.916	4.6	0.847	0.715	6.1
40d-10c	3098.1	284.0	1.000	1.000	10.4	1.000	1.000	10.0	1.000	1.000	10.0	0.988	0.942	11.1	0.996	0.984	10.5
40d-20c	5973.1	549.9	1.000	1.000	20.0	1.000	1.000	20.0	1.000	1.000	20.0	0.994	0.946	19.2	1.000	1.000	20.0

[1] Averages across 10 instances for each problem setting
[2] Average number of clusters found in 10 problem instances

parameter, δ, and neighborhood size parameter, L. The chromosome length is determined by the δ value. As δ increases, the chromosome length hence the computation time significantly decreases, because the number of genes that are crossed and mutated decreases. The decrease in time depending on the selection of δ, is demonstrated in Garza-Fabre et al. (2017). The neighborhood size shows its affect on the connectivity computations. The connectivity function for a solution is updated only if the two points are in different clusters and the intersection of their neighborhoods is not empty. Hence, as the neighborhood size decreases, the connectivity computations require less time. As opposed to the Δ-MOCK algorithm, MOCNC does not need any user-defined parameters. It is evident that the choice of the parameters defined above decreases the computation time of Δ-MOCK. Furthermore, using a fixed value for L and δ, Δ-MOCK times do not increase much as the data set size and dimensionality increase. However, choice of these parameters requires a preliminary study. On the other hand, we propose a parameter-free algorithm, in return for the higher computation time.

k-means and *single-link* clustering algorithms are simple and fast clustering algorithms. Their computational times are less than MOCNC and Δ-MOCK. However, their clustering solution qualities are poorer than the multi-objective clustering algorithms.

5.5 Computational Results for Benchmark Data Sets

We also experimented our algorithm on 20 arbitrary shaped data sets described in Section 5.1. The results for benchmark data sets are presented in Table 5.6. In most of the data sets, *k-means* does not result in good partitions; it results in mixes the clusters. *Single-link* clustering finds the target clusters for 14 problem settings in which the clusters are well-separated. In 19 data sets, except the *square4* data set, MOCNC and Δ-MOCK find good clustering solutions. However, in *square4* data set, the clusters are overlapped. Hence, the separation and connectivity objectives fail to detect the clustering. MOCNC and Δ-MOCK find the target clusters for 15 and 12 data sets, respectively. We perform the 2-sample t-test on 20 data sets at 5% significance level to test if the differences in average RI and ARI values are statistically significant. The

Table 5.5: Computational times (in seconds) of MOCNC, Δ-MOCK, *k*-means and Single-Link for generated data sets

Problem	**Number of Points**	**Number of NC Closures**	**MOCNC**				**Δ-MOCK**	***k-means***	**Single-Link**
			NC	*GA*	*Improvement*	*Total*			
2d-5c	1467.10	60.70	5.26	0.52	0.00	5.79	0.64	0.04	0.15
2d-10c	3095.20	132.20	35.17	3.37	0.02	38.55	3.50	0.11	1.66
2d-20c	5657.50	246.80	185.40	20.57	0.13	206.10	3.51	0.52	13.64
5d-5c	1561.20	31.30	6.35	0.26	0.00	6.62	0.76	0.04	0.15
5d-10c	2822.80	59.40	29.41	0.67	0.00	30.08	2.77	0.13	1.24
5d-20c	6073.20	133.70	247.92	4.67	0.02	252.61	17.18	0.58	16.63
10d-5c	1460.40	24.90	6.82	0.90	0.00	7.71	0.75	0.04	0.15
10d-10c	3114.50	54.10	44.53	4.50	0.01	49.05	3.30	0.14	1.79
10d-20c	5964.00	104.00	253.74	17.66	0.05	271.45	16.84	0.62	17.32
20d-5c	1405.30	50.40	7.64	5.26	0.04	12.93	0.79	0.04	0.16
20d-10c	2971.50	110.50	44.53	31.00	0.13	75.65	3.00	0.16	1.68
20d-20c	5752.10	220.70	250.10	160.40	0.55	411.05	14.33	0.72	16.94
40d-5c	1557.00	150.00	13.32	97.31	0.34	110.97	1.06	0.05	0.27
40d-10c	3098.10	284.00	64.42	446.67	1.52	512.61	3.62	0.22	2.38
40d-20c	5973.10	549.90	332.14	1912.20	5.26	2249.59	15.69	1.11	20.17

performances of the algorithms are not significantly different from each other.

Δ-MOCK fails to detect true clusters in *data_66*, *data-oo_v2*, and *train3_v1* since the user-defined neighborhood size causes to include points from different clusters in the neighborhoods. For example, two of the clusters in *data_66* include 6 and 9 points. The default neighborhood size for Δ-MOCK proposed in Garza-Fabre et al. (2017) is $L = 10$. Hence, the points in these small clusters include neighbors from a different cluster, which leads to mixed clusters.

On the other hand, MOCNC fails on *size5* and *square4* data sets since these data sets are not well-separated, e.g. MOCNC combines the right two clusters of *size5* data set into one cluster. The reason behind this is that the NC closures include points from different clusters. For data sets named as *twenty* and *fourty*, the closures produced by NC are already the clusters themselves. Hence, these data sets are solved by including a seed chromosome that links each closure to itself.

For benchmark data sets, the computational times of the MOCNC algorithm and the clustering methods used for comparison are summarized in Table 5.7. All four algorithms find clustering solutions in a reasonable time.

Table 5.6: Solution quality of MOCNC, Δ-MOCK, *k*-means and Single-Link for benchmark data sets

Problem	Number of Points	Number of Features	Number of Clusters	Number of NC Closures	MOCNC			Δ-MOCK			k-means			Single-Link		
					RI	*ARI*	*k*	*RI*	*ARI*	*k*	*RI*	*ARI*	*k*	*RI*	*ARI*	*k*
data_60	60	2	3	3	1.000	1.000	3	1.000	1.000	3	0.898	0.796	2	1.000	1.000	3
data_66	66	2	4	4	1.000	1.000	4	0.913	0.816	3	0.829	0.659	2	1.000	1.000	4
data-c-cc-nu-n2_v2	285	2	3	3	1.000	1.000	3	1.000	1.000	3	0.651	0.317	6	1.000	1.000	3
data-c-cc-nu-n_v2	192	2	3	4	1.000	1.000	3	1.000	1.000	3	0.878	0.757	2	1.000	1.000	3
data-c-cv-nu-n_v2	73	2	3	3	1.000	1.000	3	1.000	1.000	3	0.755	0.533	2	1.000	1.000	3
data-c-cv-u-n_v2	78	2	2	2	1.000	1.000	2	1.000	1.000	2	1.000	1.000	2	1.000	1.000	2
data-oo_v2	140	2	2	2	1.000	1.000	2	0.845	0.692	3	0.654	0.311	4	1.000	1.000	2
data-uc-cc-nu-n_v2	188	2	3	3	1.000	1.000	3	1.000	1.000	3	0.723	0.408	4	0.594	0.270	2
data-uc-cv-nu-n	127	2	6	6	0.980	0.960	4	0.983	0.966	3	0.816	0.618	4	0.990	0.980	5
dataX_v2	200	2	2	2	1.000	1.000	2	1.000	1.000	2	1.000	1.000	2	1.000	1.000	2
train1_v1	306	2	5	8	1.000	1.000	5	0.999	0.998	4	0.999	0.998	4	1.000	1.000	5
train2	287	2	4	6	1.000	1.000	4	1.000	1.000	4	0.853	0.618	6	1.000	1.000	4
train3_v1	361	2	5	6	1.000	1.000	5	0.995	0.990	4	0.745	0.429	4	1.000	1.000	5
fourty	1000	2	40	90	1.000	1.000	40	1.000	1.000	40	0.999	0.973	45	1.000	1.000	40
long1	1000	2	2	43	0.998	0.996	2	1.000	1.000	2	0.640	0.279	4	0.997	0.994	4
size5	1000	2	4	25	0.869	0.710	5	0.990	0.978	6	0.962	0.920	4	0.755	0.422	8
spiral	1000	2	2	2	1.000	1.000	2	1.000	1.000	2	0.526	0.051	4	1.000	1.000	2
square1	1000	2	4	33	0.987	0.966	7	0.991	0.976	4	0.990	0.973	4	0.254	0.000	6
square4	1000	2	4	27	0.768	0.440	8	0.921	0.789	4	0.939	0.837	4	0.253	0.000	4
twenty	1000	2	20	66	1.000	1.000	20	1.000	1.000	20	0.999	0.986	21	1.000	1.000	20

* Correct clusters are found with NC

** MOCNC is initialized with the two seed solutions since some of the NC closures are clusters itself

Table 5.7: Computational times (in seconds) of MOCNC, Δ-MOCK, *k*-means and Single-Link for benchmark data sets

Problem	**Number of Points**	**Number of Clusters**	**Number of NC Closures**	**MOCNC**			**Δ-MOCK**	***k-means***	**Single-Link**
				NC	*GA*	*Total*			
data_60	60	3	3	0.000	0.156	0.156	0.046	0.855	0.241
data_66	66	4	4	0.015	0.140	0.155	0.046	0.030	0.007
data-c-cc-nu-n2_v2	285	3	3	0.093	0.296	0.389	0.078	0.015	0.006
data-c-cc-nu-n_v2	192	3	4	0.047	0.156	0.203	0.078	0.015	0.008
data-c-cv-nu-n_v2	73	3	3	0.016	0.187	0.203	0.061	0.010	0.005
data-c-cv-u-n_v2	78	2	2	0.000	0.140	0.140	0.046	0.007	0.002
data-oo_v2	140	2	2	0.016	0.125	0.141	0.078	0.009	0.004
data-uc-cc-nu-n_v2	188	3	3	0.046	0.171	0.217	0.094	0.012	0.005
data-uc-cv-nu-n	127	6	6	0.031	0.187	0.218	0.078	0.027	0.009
dataX_v2	200	2	2	0.047	0.140	0.187	0.093	0.006	0.004
train1_v1	306	5	8	0.110	0.218	0.328	0.109	0.024	0.011
train2	287	4	6	0.094	0.156	0.250	0.093	0.017	0.008
train3_v1	361	5	6	0.156	0.156	0.312	0.109	0.026	0.013
fourty	1000	40	90	1.625	2.968	4.593	0.296	0.563	0.515
long1	1000	2	43	1.703	0.703	2.406	0.312	0.011	0.024
size5	1000	4	25	1.844	0.328	2.172	0.343	0.023	0.045
spiral	1000	2	2	1.749	0.218	1.967	0.297	0.014	0.022
square1	1000	4	33	1.656	0.484	2.140	0.312	0.023	0.045
square4	1000	4	27	1.749	0.468	2.217	0.296	0.021	0.045
twenty	1000	20	66	1.578	1.374	2.952	0.312	0.176	0.244

CHAPTER 6

MOCNC-F : MOCNC WITH FEATURE SELECTION

In this chapter, we present MOCNC with Feature Selection (MOCNC-F). In Section 6.1, we present the flowchart of the proposed approach and briefly introduce the algorithm steps. In Section 6.2, the additional notation used in MOCNC-F is introduced and the algorithm is presented in detail.

6.1 Overview of MOCNC-F

A data set has a number of features, which may be relevant or irrelevant (redundant). Redundant features may mask the clustering structure and make it difficult to extract clusters, or they may just be noninformative. They may even distort the data such that data points become unseparable. MOCNC-F aims to detect the relevant features in a data set.

MOCNC-F is a multi-objective evolutionary algorithm, which searches through different feature subsets. MOCNC-F makes use of MOCNC in feature selection. Specifically, using the features selected according to a MOCNC-F chromosome, MOCNC generates the clustering solutions. For each feature subset MOCNC finds the nondominated clustering solutions. The compactness and separation objective values of these clustering solutions are also used to assess the fitness of the respective MOCNC-F chromosome. Then, MOCNC-F evaluates the dominance among different clustering solutions resulting from different feature subsets. The output of MOCNC-F is a set of nondominated clustering solutions each with different compactness and separation values, and possibly with different feature subsets.

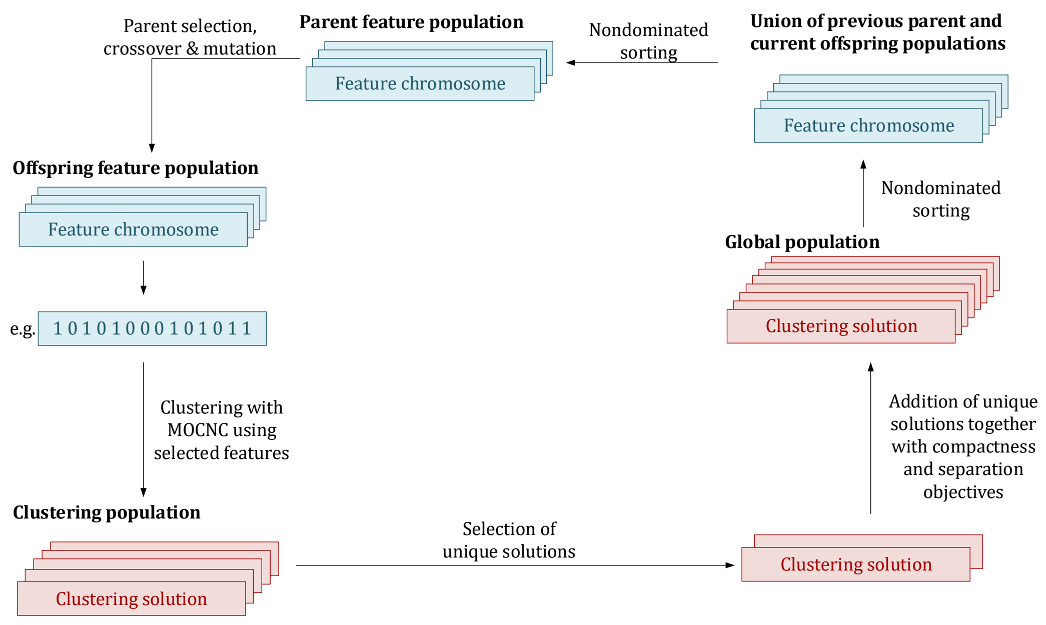

Parent selection, crossover & mutation
Parent feature population
Feature chromosome
Nondominated sorting
Union of previous parent and current offspring populations
Feature chromosome
Offspring feature population
Feature chromosome
Nondominated sorting
Global population
e.g. 1 0 1 0 1 0 0 0 1 0 1 0 1 1
Clustering solution
Clustering with MOCNC using selected features
Addition of unique solutions together with compactness and separation objectives
Clustering population
Selection of unique solutions
Clustering solution
Clustering solution

Figure 6.1: Interaction of feature selection and clustering stages

The interaction between the feature selection and clustering stages is illustrated in Figure 6.1. For each offspring feature chromosome generated from the parent feature population, MOCNC is applied and a clustering population is obtained. The unique set of clustering solutions selected from this population are added to the global population together with their corresponding compactness and separation values. Nondominated sorting is applied to the global population to construct a new feature population, which is then mixed with the previous parent population. Finally, nondominated sorting is applied to the union to form the next generation's parent feature population.

The MOCNC-F algorithm includes four main steps. The flowchart of the algorithm is presented in Figure 6.2. In Step0, the data set is read and the algorithm parameters for MOCNC and MOCNC-F are initialized. Then, to be able to compare fitness of different selected feature subsets, values of each feature are normalized between 0 and 1 as a preprocessing in Step1. Step2 is the initial population generation process for MOCNC-F. Initially, a fully random population for features is generated. Each individual feature chromosome is evaluated by calling MOCNC. After the fitness calculations, duplicating clustering solutions in the final MOCNC population are eliminated and unique solutions are gathered in a global population of clustering solutions. Nondominated sorting (based on compactness and separation) is applied to the global population to obtain the initial feature population. Then, MOCNC-F evolution starts with the initial population in Step3. For each generation, firstly the mating pool for feature chromosomes is formed. Then, offsprings are generated from the mating pool and inserted into the child population. After crossover, mutation is applied to the child population. Each offspring is evaluated by MOCNC. The resulting set of unique clustering solutions for each feature chromosome enter the global population. Then, nondominated sorting is applied to the global population and the new feature population for the next generation is obtained.

6.2 Description of the MOCNC-F Algorithm

In this section, the steps of MOCNC-F algorithm are presented.

6.2.1 Additional Notation Used in MOCNC-F

While describing MOCNC-F we use the same notation given in Section 4.4.1 and Section 4.4.2 for the clustering problem and the evolutionary algorithm, respectively. The following additional notation is used in MOCNC-F.

$fpop$	population size of feature solutions
$fgen$	number of generations in evolution in MOCNC-F
$fpcross$	crossover probability for feature solutions
$fpmut$	mutation probability for feature solutions
$globP$	set of clustering solutions in global population

6.2.2 Fitness Functions Used for the Feature Selection Problem

The fitness functions of MOCNC are the compactness and separation of a clustering solution. In MOCNC-F, we basically use the same fitness functions, but now we take the average of the fitness values by dividing the original fitness values with the number of features selected. The compactness and separation measures are formulated as

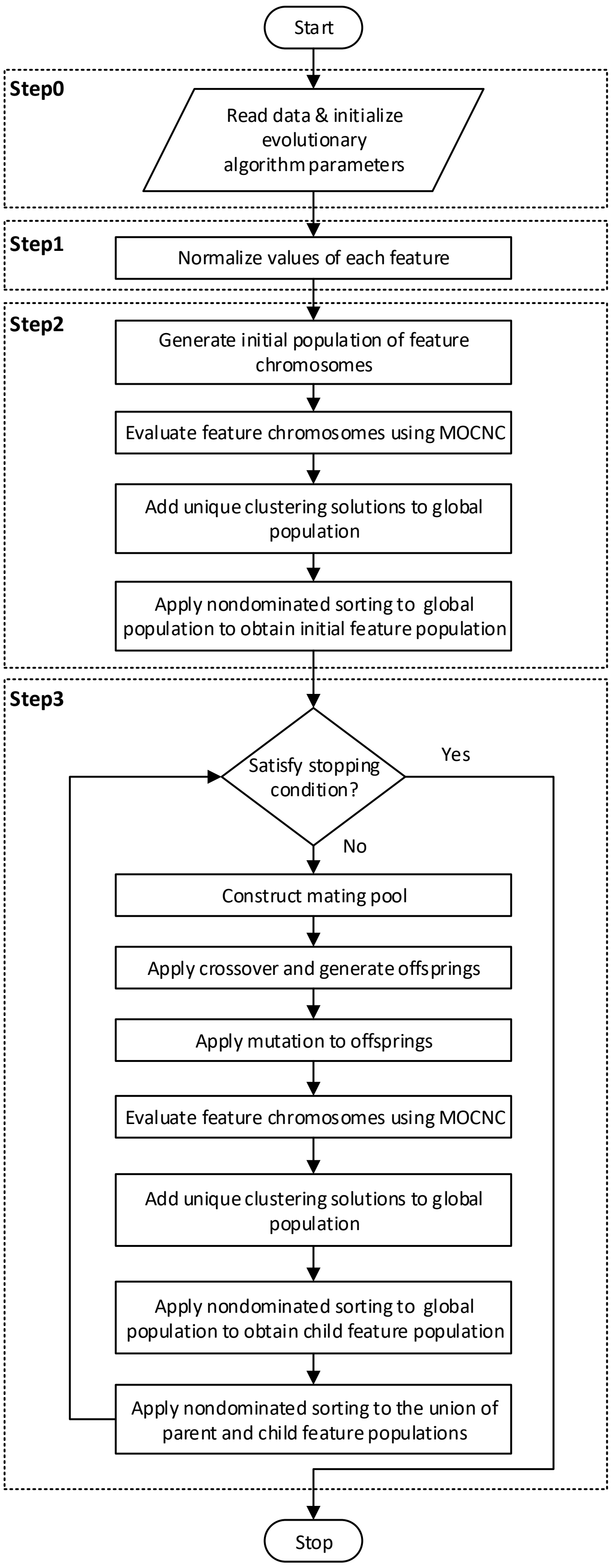

Start
Step0
Read data & initialize evolutionary algorithm parameters
Step1
Normalize values of each feature
Step2
Generate initial population of feature chromosomes
Evaluate feature chromosomes using MOCNC
Add unique clustering solutions to global population
Apply nondominated sorting to global population to obtain initial feature population
Step3
Satisfy stopping condition?
Yes
No
Construct mating pool
Apply crossover and generate offsprings
Apply mutation to offsprings
Evaluate feature chromosomes using MOCNC
Add unique clustering solutions to global population
Apply nondominated sorting to global population to obtain child feature population
Apply nondominated sorting to the union of parent and child feature populations
Stop

Figure 6.2: Flowchart of MOCNC-F

follows.

$$avgCCrA = \frac{CCrA}{d}$$

$$avgSPM = \frac{SPM}{d}$$

where d is the number of features selected in a solution.

6.2.3 Chromosome Representation

In MOCNC-F, the feature chromosomes consist of three types of information: (i) the feature selection decision variables, (ii) the corresponding clustering solution, and (iii) the fitness values. The feature selection is represented with binary genes. If the fth feature is selected, then 1 is assigned to the fth gene of the chromosome. Otherwise, 0 is assigned. The chromosome structure is illustrated in Figure 6.3. This is a chromosome with eight features and six closures. The selected features in this chromosome are features 1, 4, 6, and 7. In the clustering solution part, there are six closures linked to each other and resulting in three clusters, $C_1 = \{1, 2, 3\}$, $C_2 = \{4\}$, $C_3 = \{5, 6\}$.

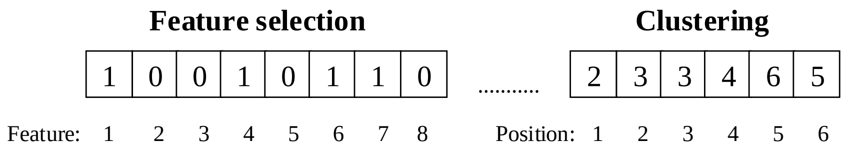


Figure 6.3: Illustration of the MOCNC-F chromosome structure

6.2.4 Initial Population Generation

In our algorithm, the initial feature population is generated randomly. No special technique is applied for initial population generation. With equal probability, a feature is either selected in a chromosome or not. The pseudocode of the initial population generation is given in Algorithm 10.

Algorithm 10: Initial population generation for features

```
Procedure generate_initial_population_for_features()
    input  : $numF, fpop$
    output: $P^0$
    for $s = 1, ..., fpop$
        for $f = 1, ..., numF$
            $rnd = U[0, 1]$
            if $rnd < 0.5$ then
                $P_s^0(f) = 1$
            else
                $P_s^0(f) = 0$
            end
        end for
    end for
end
```

6.2.5 Genetic Operators

The selection and crossover operators of MOCNC-F are the same as the ones in MOCNC. The binary tournament selection is used for mating pool construction and the uniform crossover is applied. The mutation operator for feature chromosomes is naturally different from the mutation operator in MOCNC. The bit-flip mutation operator is used. This mutation operator chooses a gene randomly and changes its current binary value with a certain mutation probability. The bit-flip mutation operator is presented in Algorithm 11.

6.2.6 The MOCNC-F Algorithm

The MOCNC-F has four main steps. The steps are initialization of parameters, normalization of the feature values, initial population generation, and evolution. The pseudocode of the MOCNC-F algorithm is presented in Algorithm 12.

In the initialization step, the data set is read, and the evolutionary algorithm parameters, which are the population size, the number of generations, the probability of crossover, and the probability of mutation, for MOCNC-F are initialized.

Algorithm 11: Mutation for features

```
Procedure bit-flip_mutation()
    input  : $numF, O_s^t$
    output: mutated $O_s^t$
    $rnd = U[0,1]$
    if $rnd < fpmut$ then
        $f$ = discrete uniform $[1, numF]$
        if $O_s^t(f) = 1$ then
            $O_s^t(f) = 0$
        else
            $O_s^t(f) = 1$
        end
    end
end
```

Step1 is a preprocessing step before the main loop of MOCNC-F. In this step the data set is normalized. Features may have different ranges and distributions. In order to compare different feature subsets, feature values should be in the same range. Hence, values of each feature are normalized between 0 and 1 by using the maximum and the minimum value of that feature across the data set.

In Step2, initial population for feature chromosomes are generated. For each individual of the initial feature population, MOCNC is applied to evaluate the corresponding feature subset, (see line 11). The output of the MOCNC is a population of non-dominated clustering solutions for the given feature subset. However, this population may contain identical clustering solutions which may cause premature convergence for MOCNC-F. To avoid this, identical solutions are eliminated and only the unique clustering solutions are kept for each feature chromosome. In the elimination process, it is assumed that the clustering solutions with equal fitness values are the same. The set of unique solutions obtained from MOCNC for each feature chromosome are added to the global population of MOCNC-F, (see lines 12-13). In the global population there can be at most $fpop * cpop$ solutions. Each chromosome in the global population includes a feature subset, a corresponding clustering solution, and fitness values. Nondominated sorting is applied to the global population to construct the initial feature population having $fpop$ individuals, (see line 15).

Step3 is the evolution process of the MOCNC-F. In each generation, firstly the mating pool is constructed with binary tournament selection. Then, uniform crossover is applied to the parent population to generate the child population. Next, the bit-flip mutation is applied to each offspring in the child population. The child population consists of $fpop$ solutions. To evaluate each feature offspring, MOCNC is called, (see line 22). The unique clustering solutions obtained for each feature offspring are added to the global population, (see lines 23-24). Then, nondominated sorting is applied to the global population and the child feature population is generated in line 26. Then, the new feature population is constructed by applying nondominated sorting to the union of the parent and child populations in line 27. At the end of $fgen$ generations the final feature population, which has at most $fpop$ nondominated clustering solutions with different feature subsets is obtained.

Algorithm 12: The MOCNC-F Algorithm

```
Procedure MOCNC-F()
    input  : Data set $D$
    output: Nondominated clustering solutions obtained with different feature
            subsets
    Step0. Initialization
    Set $fpop, fgen, fpcross, fpmut, cpop, cgen, cpcross, cpmut$
    Read data set
    Step1. Normalization of the data set
    for $f = 1, ..., numF$
        Normalize feature $f$ between 0 and 1
    end for
    Step2. Initial population generation for features
    Call generate_initial_population_for_features() $\rightarrow P^0$
    for $s = 1, ..., fpop$
        Call MOCNC() for feature subset in $P^0_s$ to obtain nondominated
        clustering solutions
        Eliminate identical clustering solutions
        Add unique clustering solutions to $globP$
    end for
    Apply nondominated sorting to $globP$ to obtain $P^0$
    Step3. Evolution
    for $t = 1, ..., fgen$
        Call binary_tournament_selection() for $P^{t-1} \rightarrow M^t$
        Call uniform_crossover() for $M^t \rightarrow O^t$
        for $s = 1, ..., fpop$
            Apply bit-flip_mutation() for $O^t_s$
            Call MOCNC() for feature subset in $O^t_s$ to obtain nondominated
            clustering solutions
            Eliminate repeated clustering solutions
            Add unique clustering solutions to $globP$
        end for
        Apply nondominated sorting to $globP$ to obtain $O^t$
        Apply nondominated sorting to $(P^{t-1} \cup O^t)$ to obtain $P^t$
    end for
end
```

CHAPTER 7

COMPUTATIONAL RESULTS OF MOCNC-F

In this chapter, we report the computational results of MOCNC-F. In Section 7.1, we describe the generation of redundant features and give the data set properties. In Section 7.2 we give the parameter settings of the algorithm and performance measures used for evaluation. Then, the benchmark method used for comparison is introduced in Section 7.3. Finally, we present the computational results and discuss them in Section 7.4.

7.1 Data Sets

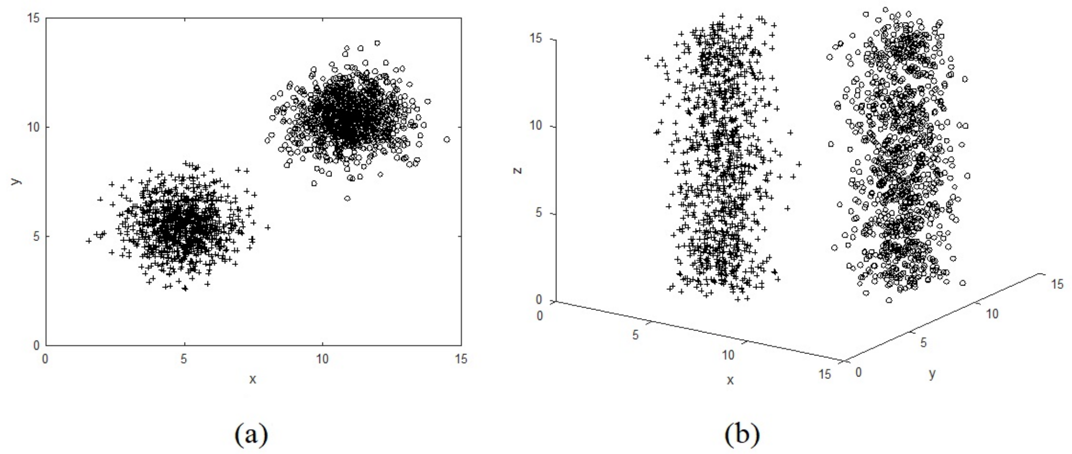
In our computational experiments, we used the generated data sets previously described in Section 5.1. There were 15 different problem settings with 2, 5, 10, 20, 40 features and 5, 10, 20 clusters. We generated 10 data sets for each problem setting. In new experimental study for feature selection, extra features called redundant features are generated in addition to the original features. If the original dimensionality (number of features) is d, $0.2d$, $0.5d$, and d redundant features are added to the data sets and new data sets are generated. Hence, with each original data set, three new data sets are generated having a different number of redundant features. The data set properties are summarized in Table 7.1. 2-dimensional data sets are excluded from our experiments. Since the number of possible feature selection solutions with 2 original and 2 redundant features is only 2^4, the correct selection can be found by simple enumeration. In total, there are 360 data sets consisting of 4 x 3 x 3 = 36 different problem settings each having 10 different instances.

Values of redundant features are generated from uniform distribution using the ranges

Table 7.1: Summary of feature selection data sets

Number of Original Features	**Number of Clusters**	**Range of Values of Redundant Features**	**Number of Redundant Features**
5	5	[-10,10]	{1,3,5}
5	10	[-20,20]	{1,3,5}
5	20	[-40,40]	{1,3,5}
10	5	[-10,10]	{2,5,10}
10	10	[-20,20]	{2,5,10}
10	20	[-40,40]	{2,5,10}
20	5	[-15,15]	{4,10,20}
20	10	[-20,20]	{4,10,20}
20	20	[-40,40]	{4,10,20}
40	5	[-15,15]	{8,20,40}
40	10	[-20,20]	{8,20,40}
40	20	[-40,40]	{8,20,40}

of original features. We assume that the feature values generated from uniform distribution behave as noise in the data sets and do not contribute to the clustering process. To show the redundancy of uniformly distributed values, an example is provided in Figure 7.1. In Figure 7.1(a), an original 2-dimensional data set having 2 clusters is shown. Both of the features are required to separate the clusters. In Figure 7.1(b), a new feature generated from uniform distribution is added to the original features. The new feature does not provide any additional information for detecting the clusters, so it is a redundant feature.



(a) (b)

Figure 7.1: An example for a redundant feature

Table 7.2: Parameter setting of the MOCNC-F algorithm

Parameter	Value
Population size for features	20
Number of generations for features	15
Crossover probability for features	1
Mutation probability for features	0.5

7.2 Parameter Settings and Performance Measures

The parameters of MOCNC-F are those that are used in NSGA-II. We give the additional parameter settings used in MOCNC-F in Table 7.2. The MOCNC parameters are kept as the same, and the final parameter values for MOCNC-F are set after pilot runs. To set the feature population size, we analyze the number of nondominated clustering solutions in each generation, and set its value large enough to cover all nondominated solutions. To set the number of generations for feature solutions, we run the algorithm up to 20 generations, and we analyze the convergence behavior of the algorithm at each 5 generation interval. For the mutation probability, first two extreme values are tried. When the mutation probability is 0, the exploration capabilities of the algorithm decrease. On the other hand, when the mutation probability is 1, a good feature solution that may be found with crossover is changed and finding this solution again may take along time. Hence, the mutation probability is set as 0.5 after trying some other levels between 0.5 and 1.

To evaluate the clustering performance for different feature subsets, we again use the Rand Index (RI) and Adjusted Rand Index (ARI) as described by Equations 5.1 and 5.2, respectively.

7.3 Benchmark Method Used for Comparison

In Section 5.4, we presented the experimental results of MOCNC and the methods used for comparison. Among these methods, Δ-MOCK shows slightly better performance than MOCNC on a few problem settings. However, in most of the problem settings, there is no significant difference between MOCNC and Δ-MOCK. For this reason, we use Δ-MOCK as a benchmark algorithm to MOCNC in the feature selection algorithm. In the MOCNC-F algorithm, we replace MOCNC with Δ-MOCK, hence we evaluate the clustering performance for a selected feature subset by Δ-MOCK. To make it easier to follow, this adapted algorithm is referred to as Δ-MOCK-F.

As mentioned earlier, MOCNC and Δ-MOCK have one objective in common, which is the compactness measure. Their second objectives are different; MOCNC uses the separation whereas Δ-MOCK uses the connectivity. Both the compactness and separation objectives are distance-based measures, so they are affected by the number of features selected in the solution. However, the connectivity objective of Δ-MOCK is based on penalizing neighbors and is not affected by the number of features selected since the neighborhoods are constructed from scratch for each feature subset.

In MOCNC-F, to remove the effect of the number of features, we divide the original compactness and separation values by the number of features selected. In Δ-MOCK-F, we also apply this but only for the compactness objective. The connectivity objective is calculated as in Δ-MOCK.

7.4 Computational Results

To test the performance of the MOCNC-F algorithm, we ran our algorithm on 360 data sets, in which there are 36 different problem settings each having 10 instances. The MOCNC-F algorithm was coded in C and all the experiments are run on a PC with a 3.6 GHz 4-Core Intel Core i7-4790 processor and 8GB of RAM.

MOCNC-F and Δ-MOCK-F are applied to the data sets described in Section 7.1. We use the same parameter settings for both algorithms to ensure that the number of solutions visited are basically the same. The detailed results for individual problem instances are given in Tables C.1 - C.12 in Appendix C. The summary of computational results are given in Table 7.4.

The first column of Table 7.4 presents the names of the problem settings. The names of the problem settings are the original names of the data sets as explained in Chapter 5. The names reflect the number of original features and the number of clusters in the data set. For example, 5d-10c represents the data problem setting having 5 original features and 10 clusters. For each problem setting, there are three additional settings depending on the number of redundant features added. The third column represents the number of redundant features added to the data set. For each pair of problem setting and the number of redundant features there are 10 instances, and the values in each row represents the averages across these 10 instances. As in the case of MOCNC, nondominated solution in the final population that gives the best RI and ARI values is taken as the solution for each problem instance. For the MOCNC-F and Δ-MOCK-F algorithms, the averages of the RI and ARI values across 10 instances, the average number of clusters found, the average number of original and redundant features selected are reported. It should be noted that, since this is a multi-objective problem, the nondominated solutions may have the same clustering solution with different feature subsets hence different objective function values. We report one of the solutions when there are alternative solutions with the same RI and ARI values.

According to Table 7.4, in 34 out of 36 problem settings, MOCNC-F shows better performance than Δ-MOCK-F. In two problem settings, Δ-MOCK-F performs slightly better in terms of RI and ARI values. When we analyze the performance of the algorithms on individual data sets, MOCNC-F finds the target clustering solution in 284 data sets. The worst-case ARI performance of MOCNC-F is 0.452 with only one data set, which is 40d-5c-3 which has 40 redundant features. However, when we run the MOCNC-F algorithm until 20 generations for this data set, the ARI performance is increased to 0.999. The second worst-case ARI performance of MOCNC-F is 0.907 for 20d-10c-5 with 20 redundant features. On the other hand, Δ-MOCK-F could not find the target clustering solution in any of the 360 data sets. Moreover, the worst-case ARI performance of Δ-MOCK-F is 0.196 for 5d-5c-9 with only 1 original feature selected. In fact, in Δ-MOCK-F solutions, ARI is below 0.3 for 31 problem instances and below 0.5 for 76 problem instances.

We again perform the 2-sample t-test at 5% significance level to see if the differences in RI and ARI values between the two algorithms are statistically significant. When we compare the RI and ARI values of all 360 data sets, MOCNC-F shows significantly better performance compared to Δ-MOCK-F. Moreover, we analyze the individual problem settings for the significance between mean values. MOCNC-F has significantly better performance in 26 problem settings. The performance gap between the two algorithms decreases as the number of original features increases.

MOCNC and Δ-MOCK perform equally well for the clustering problem. When it comes to the feature selection problem, however, MOCNC-F clearly outperforms Δ-MOCK-F. We believe that the main reason for this is the choice of objectives.

The objective functions of MOCNC-F are the averaged versions of the original MOCNC objective functions with the selected number of features. When the compactness objective function is as in MOCNC, the compactness value naturally increases as the number of features increases. However, after dividing this by the number of features selected, the compactness function does not follow an increasing or decreasing trend. On the other hand, the relationship between the separation objective function and the number of features selected is not straightforward. The original separation objective function of MOCNC has a tendency to improve as the number of features selected increases, but not strictly as in the original compactness function. As the number of features increases, the clusters mostly become more separated, but this behavior is not always observed. It happens only when the additional selected features increase the separation of clusters. As the selected feature subset is changed, the closest points between the pairs of clusters may change, which may cause a decrease or increase in the separation function value. Because of this, the separation function can also be normalized with the number of features.

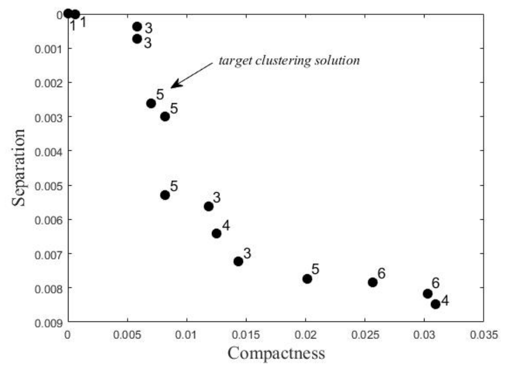
It should be noted that, as the number of features selected does not result in an increasing or decreasing trend in compactness and separation objective functions of MOCNC-F, the multi-objective optimization becomes truly effective since it reveals the tradeoffs between these two objectives.

The connectivity objective of Δ-MOCK replaces our separation objective. It sums the penalties assigned to points when they are not in the same cluster as their nearest neighbors for a fixed neighborhood size. This only takes into account the rank of the nearest neighbor within the neighborhood, not the actual distances. Since the neighborhood is constructed from scratch for each feature subset, the connectivity objective is not directly affected by the number of features selected. However, as the number of features selected decreases, although the clusters may become overlapped, their compactness objective may be improved in the reduced space. Hence, compactness favors fewer features, but connectivity does not have a significant counteracting effect to increase the number of features and to force clusters to be separated from each other. It was stated in Handl and Knowles (2007) that "the concept of spatial separation is intrinsic (opposite) to that of connectedness" meaning that maximizing separation corresponds to minimizing their connectedness measure. However, we observe that in feature selection problem these objectives are not substitutes for each other.

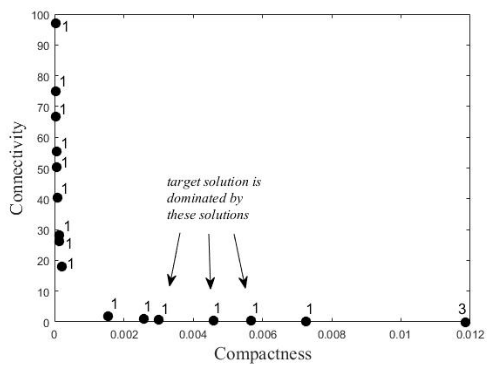
The compactness and separation measures tend to counteract or balance each other in selecting the right features. However, the connectivity measure in Δ-MOCK-F has no significant counteracting effect against the compactness measure. In fact, as the number of features decreases, when the clusters become overlapped in fewer dimensions, multi-objective optimization leads to merging clusters so that the compactness is decreased and hence the connectivity is improved. Therefore, in most problem settings, Δ-MOCK-F selects fewer original and redundant features compared to MOCNC-F. We can conclude that the objective functions in MOCNC-F is more suitable for the feature selection problem. This choice facilitates finding different and better nondominated clustering solutions for different feature subsets.

To illustrate this situation with an example, we present the Pareto fronts of MOCNC-F and Δ-MOCK-F final populations for 5d-5c-1 with 1 redundant feature in Figure 7.2. In the figure the points represent the nondominated solutions. The number next to each point stands for the number of features selected in that solution. In Figure 7.2(a), the Pareto front of MOCNC-F is presented. The target clustering solution is one of the nondominated solutions. On the other hand, the target clustering solution could not be found by Δ-MOCK-F since it is dominated with compactness and connectivity objectives. The compactness and connectivity values of the target clustering solution are 0.007 and 0.621, respectively. However, in Δ-MOCK-F this solution is dominated by three solutions whose compactness and connectivity values are (0.003, 0.590), (0.005, 0.379), (0.006, 0.322). Moreover, the Pareto front of Δ-MOCK-F shows that the number of features selected in nondominated solutions are mostly 1, whereas there are different numbers of features selected in MOCNC-F solutions.

In our experiments, we expected both MOCNC-F and Δ-MOCK-F algorithms to se-



(a) Pareto front of MOCNC-F



(b) Pareto front of Δ-MOCK-F

Figure 7.2: Pareto fronts of MOCNC-F and Δ-MOCK-F for 5d-5c-1 with 1 redundant feature

lect the original features and eliminate the redundant features in the best solutions. Since Δ-MOCK-F could not find the target clustering solution in any of the data sets, reaching conclusions on the redundancy of features based on Δ-MOCK-F results is difficult. However, this can be done for MOCNC-F results. There are many instances for which MOCNC-F finds the target clustering solution with fewer than the original features. For example, for 5d-5c-5 MOCNC finds the target clustering with 5 features, but MOCNC-F finds the same clustering with only 4 original features, after adding 3 redundant features. This indicates that one of the original features is not informative. As another example, MOCNC finds the target clustering solutions for all 10 data sets in 10d-5c problem setting with 10 original features, whereas MOCNC-F again finds the target clusterings with an average of 7.6, 7.8, and 8.8 original features when there are 2, 5, and 10 redundant features, respectively. We may conclude that especially when the dimensionality increases, some of the original features may also be redundant.

In some data sets, a few redundant features are also selected in addition to the original features and the target clustering is found. When we run the MOCNC-F algorithm for more generations for these data sets, some of the redundant features are eliminated while the clustering solution remains the same. This means that as the number of generations increase, the chance of eliminating the redundant features increases. For instance, 20d-10c-8 with 20 redundant features has an ARI value of 1 with 18 original and 6 redundant features selected at generation 15. When we run this data set for 5

Table 7.3: Solution quality of MOCNC-F and Δ-MOCK-F for high dimensional data sets from literature

Data Set	# of Points	MOCNC-F				Δ-MOCK-F			
		RI	*ARI*	*k*	*OF*	*RI*	*ARI*	*k*	*OF*
100d-10c-1*	2892	1.000	1.000	10	37	0.999	0.994	10	40
100d-10c-2**	1951	0.999	0.996	10	36	0.998	0.991	10	49
100d-20c-1**	5485	0.999	0.999	20	38	0.999	0.989	21	51
200d-10c-1**	2666	1.000	0.999	10	95	0.999	0.993	10	110

* From Handl and Knowles (2007)
** From Garza-Fabre et al. (2017)
k : Number of clusters found in the solution
OF : Number of redundant features selected in the solution

more generations, the number of original features selected becomes 19 whereas the number of redundant features selected decreases to 4.

We also experimented with MOCNC-F on three data sets from Garza-Fabre et al. (2017), and one data set from Handl and Knowles (2007). These data sets are extremely high-dimensional data sets and we aim to see whether there are redundant features in these data sets. The experimental results of MOCNC-F and Δ-MOCK-F are given in Table 7.3. Although MOCNC-F selects only a portion of the features, it finds good clustering solutions. Therefore, we can conclude that in such high dimensions there can be redundant features with the multivariate normal data set generator used.

The computing times of the MOCNC-F and Δ-MOCK-F algorithms are summarized in Table 7.5. After analyzing the computational time results of MOCNC and Δ-MOCK for the clustering problem in Section 5.4, we expected the execution time of MOCNC-F to be longer than Δ-MOCK. Indeed, the computational times of Δ-MOCK-F are significantly shorter than MOCNC-F. Although the computational times of MOCNC-F are long, feature selection is not a problem to be solved frequently and long times can be afforded. Hence, considering its solution quality, MOCNC-F is a promising algorithm for simultaneous feature selection and clustering.

Table 7.4: Solution quality of MOCNC-F and Δ-MOCK-F for generated data sets

Problem	**# of Points**	**RdF**	**MOCNC-F**					**Δ-MOCK-F**				
			RI	*ARI*	*k*	*OF*	*RF*	*RI*	*ARI*	*k*	*OF*	*RF*
5d-5c	1561.20	1	0.996	0.991	5.1	4.7	0.0	0.784	0.477	3.8	1.7	0.0
		3	0.997	0.991	5.0	4.6	0.1	0.792	0.488	4.0	1.7	0.0
		5	0.996	0.991	5.0	4.7	0.1	0.808	0.509	4.7	1.5	0.0
5d-10c	2822.80	1	0.998	0.989	9.9	5.0	0.0	0.830	0.386	8.2	1.4	0.0
		3	0.997	0.988	9.8	5.0	0.0	0.872	0.546	8.7	2.7	0.0
		5	0.998	0.990	9.8	5.0	0.0	0.866	0.509	8.4	2.5	0.0
5d-20c	6073.20	1	0.997	0.977	18.7	4.6	0.0	0.972	0.777	21.0	3.7	0.0
		3	0.998	0.981	19.1	4.7	0.0	0.995	0.961	19.3	4.7	0.0
		5	0.998	0.987	19.2	4.7	0.0	0.981	0.849	19.6	4.0	0.0
10d-5c	1460.40	2	1.000	1.000	5.0	7.6	0.0	0.717	0.352	3.9	1.0	0.0
		5	1.000	1.000	5.0	7.8	0.0	0.778	0.489	3.7	2.7	0.0
		10	1.000	1.000	5.0	8.8	0.0	0.943	0.868	4.9	6.9	0.0
10d-10c	3114.50	2	1.000	1.000	10.0	9.4	0.0	0.878	0.575	7.6	4.3	0.0
		5	1.000	1.000	10.0	9.2	0.0	0.936	0.769	8.6	6.5	0.0
		10	1.000	1.000	10.0	9.6	0.0	0.945	0.807	8.6	7.4	0.0
10d-20c	5964.00	2	1.000	1.000	20.1	8.4	0.0	0.999	0.994	20.0	6.8	0.0
		5	1.000	1.000	20.0	9.5	0.0	0.999	0.994	20.0	6.9	0.0
		10	1.000	1.000	20.2	9.1	0.0	0.999	0.994	20.0	6.9	0.0
20d-5c	1405.30	4	1.000	1.000	5.0	12.9	0.0	0.993	0.982	4.9	11.9	0.0
		10	1.000	1.000	5.0	15.8	1.0	0.997	0.991	5.0	12.4	0.1
		20	1.000	1.000	5.0	17.5	4.6	0.997	0.991	5.0	16.7	0.9
20d-10c	2971.50	4	1.000	1.000	10.0	13.8	0.0	0.978	0.919	9.6	10.9	0.0
		10	1.000	1.000	10.0	17.0	1.0	0.999	0.994	10.0	12.4	0.0
		20	0.996	0.983	9.5	18.1	4.8	0.998	0.992	10.9	16.9	0.7
20d-20c	5752.10	4	1.000	1.000	20.0	11.6	0.0	0.999	0.994	20.0	7.8	0.0
		10	1.000	1.000	20.0	15.9	0.1	0.999	0.994	20.0	9.5	0.0
		20	1.000	1.000	20.0	18.5	0.6	0.999	0.994	20.0	19.3	1.0
40d-5c	1557.00	8	1.000	1.000	5.0	30.0	0.9	0.997	0.993	5.0	18.0	0.0
		20	1.000	1.000	5.0	30.3	5.4	0.997	0.992	5.0	24.4	0.8
		40	0.972	0.945	4.8	30.0	12.6	0.996	0.990	5.2	31.2	5.7
40d-10c	3098.10	8	1.000	1.000	10.0	28.4	0.4	0.999	0.994	10.0	20.3	0.0
		20	1.000	1.000	10.0	33.0	3.7	0.999	0.994	10.0	27.9	1.2
		40	1.000	0.999	10.2	32.9	17.1	0.993	0.968	9.3	30.9	5.4
40d-20c	5973.10	8	1.000	1.000	20.0	28.4	0.1	0.999	0.994	20.0	14.6	0.0
		20	1.000	1.000	20.0	33.5	2.1	0.999	0.994	20.0	30.5	1.4
		40	1.000	1.000	20.0	31.4	13.5	0.999	0.994	20.0	30.5	4.9

RdF : Number of redundant features added to the original data set
k : Average number of clusters found in the solutions
OF : Average number of original features selected in the solutions
RF : Average number of redundant features selected in the solutions

Table 7.5: Computational times (in seconds) of MOCNC-F and Δ-MOCK-F for generated data sets

Problem	**# of Points**	**RdF**	**MOCNC-F**			**Δ-MOCK-F**
			NC	*GA*	*Total*	
5d-5c	1561.20	1	2321.26	1132.40	3453.65	252.78
		3	2424.51	583.19	3007.70	245.92
		5	2531.49	306.08	2837.57	243.89
5d-10c	2822.80	1	11348.63	3041.38	14390.01	866.16
		3	11515.98	690.50	12206.48	821.93
		5	12581.55	311.69	12893.24	795.82
5d-20c	6073.20	1	82487.38	19605.81	102093.20	6212.19
		3	83191.97	1419.97	84611.94	5671.92
		5	89122.54	1153.68	90276.22	5459.11
10d-5c	1460.40	2	2245.25	169.19	2414.44	228.63
		5	2402.92	171.58	2574.51	227.47
		10	2500.18	216.17	2716.35	232.21
10d-10c	3114.50	2	16118.97	563.09	16682.05	1035.69
		5	16743.13	514.93	17258.05	1020.92
		10	17474.28	502.57	17976.85	965.98
10d-20c	5964.00	2	76938.29	2231.81	79170.11	5616.11
		5	83481.88	2027.15	85509.02	5308.02
		10	84112.95	2035.38	86148.32	4995.39
20d-5c	1405.30	4	2316.52	727.43	3043.95	231.55
		10	2571.91	850.66	3422.57	250.65
		20	2937.44	1769.79	4707.23	254.32
20d-10c	2971.50	4	13226.35	3623.23	16849.58	839.75
		10	15012.64	2392.63	17405.27	893.72
		20	19907.94	3431.57	23339.51	816.01
20d-20c	5752.10	4	71377.06	13981.53	85358.59	4474.69
		10	74755.90	7464.67	82220.57	4157.74
		20	80169.18	1952.98	82122.16	3673.09
40d-5c	1557.00	8	3800.12	15583.15	19383.27	312.17
		20	4368.99	24757.39	29126.39	332.46
		40	5088.76	37899.13	42987.89	377.33
40d-10c	3098.10	8	18060.42	47172.35	65232.77	1038.11
		20	19984.65	56927.08	76911.73	997.46
		40	23324.60	142247.27	165571.86	978.36
40d-20c	5973.10	8	90719.91	131276.52	221996.43	4550.86
		20	90923.81	12865.42	103789.23	4264.77
		40	106561.52	71078.24	177639.76	3580.02

RdF : Number of redundant features added to the original data set

CHAPTER 8

CONCLUSIONS

In this study, we address the clustering problem for data sets that have all numerical features. There may be density differences between and/or within arbitrary shaped clusters, and there are no outliers or noise. Data sets may be high-dimensional with a number of redundant features. Moreover, the number of clusters is unknown. Each of these characteristics needs special treatment and makes the clustering problem difficult.

In addition to the data set characteristics, the choice of the objective function(s) for the clustering algorithms is not straightforward. This choice may result in completely different clustering solutions. For the defined clustering problem, we propose a multi-objective evolutionary clustering algorithm, namely MOCNC. The MOCNC algorithm takes into account the three objectives of clustering: compactness, separation, and connectivity. MOCNC uses the multi-objective framework and nondominated sorting property of the well-known evolutionary algorithm NSGA-II (Deb et al. 2002) and simultaneously optimizes two conflicting objectives of clustering. The objective functions are minimizing the intra-cluster distances (compactness) and maximizing the inter-cluster distances (separation).

To handle the third objective, which is the connectivity of data points, MOCNC uses a special Neighborhood Construction (NC) algorithm (İnkaya et al. 2015a) as a preprocessor. NC uses the underlying connectivity and density information in the data set to construct unique neighborhoods for data points. It then forms closures considering overlaps of these neighborhoods. The data points within a closure are connected and assumed to be in the same cluster. The closures are then given as input to the evolutionary algorithm. Throughout the generations of MOCNC, the final clusters are searched by combining these closures.

There are two advantages of using the NC closures: (i) connectivity of data points in closures is ensured and (ii) the problem becomes more scalable as the points within closures are fixed. The output of MOCNC is a set of nondominated clustering solutions. The nondominated solutions represent different tradeoffs between two conflicting objectives (compactness and separation) and may have different numbers of clusters.

We used two groups of data sets to test the solution quality of MOCNC. The first group consists of randomly generated data sets having different number of clusters and features. The second group includes 2-dimensional data sets having arbitrary shaped clusters with density differences. We also conducted a comparison analysis with two well-known single-objective clustering objectives (k-means and single-link) and a recent multi-objective clustering algorithm (Δ-MOCK). The empirical results show that both MOCNC and Δ-MOCK are capable of finding target clustering solutions for most of the data sets, whereas k-means and single-link are not nearly as successful. The MOCNC algorithm has limited success on data sets having larger intra-cluster distances than inter-cluster distances. When this is the case, the NC closures contain data points from different clusters, which results in mixed clusters and poor solutions.

The computation time of MOCNC consists of mainly two parts, NC and GA. The shares of NC and GA computation times change as the dimensionality increases. The NC times increase proportional to the size of the data set and the dimensionality. The GA times are mainly affected by the number of closures and the number of features since the chromosome length is determined by the number of closures and the number of features affect fitness calculation times.

The computation times of MOCNC are longer than those of the algorithms used for comparison. The k-means and single-link algorithms are simple and fast clustering algorithms. However, their clustering performances are poorer than the multi-objective clustering algorithms. The Δ-MOCK algorithm uses two significant user-defined parameters to increase the efficiency and reduce the computation time, which are encoding-length parameter that determines the chromosome length, and neighborhood size parameter that affects the connectivity objective function computations. As opposed to the Δ-MOCK algorithm, MOCNC does not need any user-defined problem parameters. It is evident that the choice of the parameters defined above decreases the computation time of Δ-MOCK. However, choice of these parameters requires a preliminary study. On the other hand, we propose a parameter-free algorithm, in return for the higher computation time.

After obtaining empirical results for MOCNC, we extend it for the feature selection problem and propose a new algorithm, namely MOCNC-F. In MOCNC-F, the clustering problem defined above remains the same, but the data sets may contain redundant features. The number of redundant features is unknown. MOCNC-F finds the set of nondominated clustering solutions each with different compactness and separation values, and possibly with different selected feature subsets.

To the best of our knowledge, there is no algorithm that takes into account the feature selection and clustering problems simultaneously in a multi-objective perspective, where the number of clusters and the number of redundant features are unknown. Hence, to compare the solution quality of MOCNC-F, we adapt the Δ-MOCK algorithm to the feature selection problem, namely Δ-MOCK-F. We apply both of the algorithms to randomly generated data sets used in MOCNC experiments by adding different number of redundant features. MOCNC-F provides better results than Δ-MOCK-F in almost all data sets. The empirical results prove that the choice of objective functions in MOCNC-F increases the simultaneous feature selection and clustering solution quality.

The computation time of MOCNC-F is directly affected by that of MOCNC. The main limitation of the MOCNC-F algorithm is the long computation times. There is a need for the improvements to make the algorithm run more efficiently and reduce the execution times. However, considering that feature selection problem is not solved frequently, MOCNC-F has a very good performance. Hence, MOCNC-F is a new and promising algorithm for clustering with feature selection.

In order to prevent MOCNC from mixing clusters due to fixed NC closures, some evaluation measures can be defined for the NC closures and based on these measures division techniques can be applied to the NC closures as well as combining them as in the currently used mutation operator.

In this study, all attributes of the data set are numerical and the Euclidean distance is used to evaluate the similarity between data points. Another future research issue may be analyzing the algorithms using different compactness and separation measures based on the Euclidean distance. Also, different multiple objectives can be defined to perform clustering for data sets having categorical or mixed features.

Focus of this study is on the unsupervised clustering problem. Both MOCNC and MOCNC-F can be adapted for the semi-supervised version of the problem. Given that some of the data points are in the same cluster (which correspond to must-link constraints) and others are in different clusters (which correspond to cannot-link constraints), it may be possible to preprocess the NC closures accordingly. The evolutionary algorithm operators may also be updated to handle such additional information.

REFERENCES

Bellman, R. E. *Adaptive control processes: a guided tour.* Princeton, N.J., Princeton University Press, 1961., 1961.

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

Celebi, M. E. and Aydin, K. *Unsupervised Learning Algorithms.* Springer, 2016.

Chandrashekar, G. and Sahin, F. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

Chen, E. and Wang, F. Dynamic clustering using multi-objective evolutionary algorithm. In *International Conference on Computational and Information Science*, pages 73–80. Springer, 2005.

Corne, D. W., Jerram, N. R., Knowles, J. D., and Oates, M. J. Pesa-ii: Region-based selection in evolutionary multiobjective optimization. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 283–290. Morgan Kaufmann Publishers Inc., 2001.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2): 182–197, 2002.

Demirtaş, K. *An interactive preference based multiobjective evolutionary algorithm for the clustering problem. [Electronic resource].* Ankara : METU, 2011.

Du, J., Korkmaz, E. E., Alhajj, R., and Barker, K. Alternative clustering by utilizing multi-objective genetic algorithm with linked-list based chromosome encoding. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 346–355. Springer, 2005.

Duda, R. O., Hart, P. E., and Stork, D. G. *Pattern classification.* John Wiley & Sons, 2012.

Dutta, D., Dutta, P., and Sil, J. Simultaneous continuous feature selection and k clustering by multi objective genetic algorithm. In *Advance Computing Conference (IACC), 2013 IEEE 3rd International*, pages 937–942. IEEE, 2013.

Ferligoj, A. and Batagelj, V. Direct multicriteria clustering algorithms. *Journal of Classification*, 9(1):43–61, 1992.

Freitas, A. A. A critical review of multi-objective optimization in data mining: a position paper. *ACM SIGKDD Explorations Newsletter*, 6(2):77–86, 2004.

Garza-Fabre, M., Handl, J., and Knowles, J. An improved and more scalable evolutionary approach to multiobjective clustering. *IEEE Transactions on Evolutionary Computation*, 2017.

Guha, S., Rastogi, R., and Shim, K. Cure: an efficient clustering algorithm for large databases. In *ACM Sigmod Record*, volume 27, pages 73–84. ACM, 1998.

Guyon, I. and Elisseeff, A. An introduction to feature extraction. In *Feature extraction*, pages 1–25. Springer, 2006.

Handl, J. and Knowles, J. Evolutionary multiobjective clustering. In *International Conference on Parallel Problem Solving from Nature*, pages 1081–1091. Springer, 2004.

Handl, J. and Knowles, J. Exploiting the trade-off—the benefits of multiple objectives in data clustering. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 547–560. Springer, 2005a.

Handl, J. and Knowles, J. Improvements to the scalability of multiobjective clustering. In *Congress on Evolutionary Computation*, pages 2372–2379. Citeseer, 2005b.

Handl, J. and Knowles, J. On semi-supervised clustering via multiobjective optimization. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1465–1472. ACM, 2006a.

Handl, J. and Knowles, J. Feature subset selection in unsupervised learning via multiobjective optimization. *International Journal of Computational Intelligence Research*, 2(3):217–238, 2006b.

Handl, J. and Knowles, J. Semi-supervised feature selection via multiobjective optimization. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pages 3319–3326. IEEE, 2006c.

Handl, J. and Knowles, J. An evolutionary approach to multiobjective clustering. *IEEE transactions on Evolutionary Computation*, 11(1):56–76, 2007.

Handl, J. and Knowles, J. Clustering criteria in multiobjective data clustering. In *International Conference on Parallel Problem Solving from Nature*, pages 32–41. Springer, 2012.

Hruschka, E. R., Campello, R. J., Freitas, A. A., et al. A survey of evolutionary algorithms for clustering. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 39(2):133–155, 2009.

İnkaya, T. *A methodology of swarm intelligence application in clustering based on neighborhood construction. [Electronic resource].* Ankara : METU, 2011.

İnkaya, T., Kayalıgil, S., and Özdemirel, N. E. An adaptive neighbourhood construction algorithm based on density and connectivity. *Pattern Recognition Letters*, 52: 17–24, 2015a.

İnkaya, T., Kayalıgil, S., and Özdemirel, N. E. Ant colony optimization based clustering methodology. *Applied Soft Computing*, 28:301–311, 2015b.

Jain, A. K. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31 (8):651–666, 2010.

Jain, A. K., Murty, M. N., and Flynn, P. J. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

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

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

Kira, K. and Rendell, L. A. The feature selection problem: Traditional methods and a new algorithm. In *Aaai*, volume 2, pages 129–134, 1992.

MacQueen, J. et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

Martínez-Peñaloza, M.-G., Mezura-Montes, E., Cruz-Ramírez, N., Acosta-Mesa, H.-G., and Ríos-Figueroa, H.-V. Improved multi-objective clustering with automatic determination of the number of clusters. *Neural Computing and Applications*, 28 (8):2255–2275, 2017.

Matake, N., Hiroyasu, T., Miki, M., and Senda, T. Multiobjective clustering with automatic k-determination for large-scale data. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 861–868. ACM, 2007.

Mierswa, I. and Wurst, M. Information preserving multi-objective feature selection for unsupervised learning. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1545–1552. ACM, 2006.

Morita, M., Sabourin, R., Bortolozzi, F., and Suen, C. Y. Unsupervised feature selection using multi-objective genetic algorithms for handwritten word recognition. In *null*, page 666. IEEE, 2003.

Özyer, T., Liu, Y., Alhajj, R., and Barker, K. Multi-objective genetic algorithm based clustering approach and its application to gene expression data. In *International Conference on Advances in Information Systems*, pages 451–461. Springer, 2004.

Pappa, G. L., Freitas, A. A., and Kaestner, C. A. Multi-objective algorithms for attribute selection in data mining. In *Applications of Multi-Objective Evolutionary Algorithms*, pages 603–626. World Scientific, 2004.

Park, Y. and Song, M. A genetic algorithm for clustering problems. In Koza, J. R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M. H., Goldberg, D. E., Iba, H., and Riolo, R., editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 568–575, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

Prakash, J. and Singh, P. K. An effective multiobjective approach for hard partitional clustering. *Memetic Computing*, 7(2):93–104, 2015.

Rayward-Smith, V. J. Metaheuristics for clustering in kdd. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 3, pages 2380–2387. IEEE, 2005.

Ripon, K. S. N., Tsang, C.-H., and Kwong, S. Multi-objective data clustering using variable-length real jumping genes genetic algorithm and local search method. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pages 3609–3616. IEEE, 2006.

Saeys, Y., Inza, I., and Larrañaga, P. A review of feature selection techniques in bioinformatics. *bioinformatics*, 23(19):2507–2517, 2007.

Saha, S. and Bandyopadhyay, S. A symmetry based multiobjective clustering technique for automatic evolution of clusters. *Pattern recognition*, 43(3):738–751, 2010.

Saha, S., Ekbal, A., Alok, A. K., and Spandana, R. Feature selection and semi-supervised clustering using multiobjective optimization. *SpringerPlus*, 3(1):465, 2014.

Saha, S., Alok, A. K., and Ekbal, A. Use of semisupervised clustering and feature-selection techniques for identification of co-expressed genes. *IEEE journal of biomedical and health informatics*, 20(4):1171–1177, 2016a.

Saha, S., Kaushik, K., Alok, A. K., and Acharya, S. Multi-objective semi-supervised clustering of tissue samples for cancer diagnosis. *Soft Computing*, 20(9):3381–3392, 2016b.

Santos, D. S., De Oliveira, D., and Bazzan, A. L. A multiagent, multiobjective clustering algorithm. In *Data mining and multi-agent integration*, pages 239–249. Springer, 2009.

Shannon, C. E. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 5(1):3–55, 2001.

Srinivas, N. and Deb, K. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3):221–248, 1994.

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

Tsai, C.-W., Chen, W.-L., and Chiang, M.-C. A modified multiobjective ea-based clustering algorithm with automatic determination of the number of clusters. In *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, pages 2833–2838. IEEE, 2012.

Voorhees, E. M. The effectiveness and efficiency of agglomerative hierarchic clustering in document retrieval. Technical report, Cornell University, 1985.

Witten, I. H., Frank, E., Hall, M. A., and Pal, C. J. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

Won, J.-M., Ullah, S., and Karray, F. Data clustering using multi-objective hybrid evolutionary algorithm. In *Control, Automation and Systems, 2008. ICCAS 2008. International Conference on*, pages 2298–2303. IEEE, 2008.

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

Zhang, Q. and Li, H. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6):712–731, 2007.

Zhang, T., Ramakrishnan, R., and Livny, M. Birch: an efficient data clustering method for very large databases. In *ACM Sigmod Record*, volume 25, pages 103–114. ACM, 1996.

Zitzler, E., Laumanns, M., and Thiele, L. Spea2: Improving the strength pareto evolutionary algorithm. *TIK-report*, 103, 2001.

APPENDIX A

PROOF OF EQUIVALENCE OF COMPACTNESS MEASURES

In this appendix, we give the proof of equivalence of the equation 4.1 and 4.2. The notation used is as follows.

C_k	set of points in cluster k
B_k	set of closures in cluster k
N	number of data points in set D
$numK$	number of clusters found in a solution
i,j	indices for data points, $i, j = 1, ..., N$
k	indices for clusters, $k, l = 1, ..., numK$
m	indices for closures
f	indice for features
d_{ij}	Euclidean distance between points i and j
c_k	centroid of cluster k
c_m	centroid of closure m
n_m	number of points in closure m
$comp_m$	compactness of closure m

$$
\begin{aligned}
CCrA &= \frac{1}{N}\sum_{k=1}^{numK}\sum_{i\in C_k} d_{ic_k}^2 \\
&= \frac{1}{N}\sum_{k=1}^{numK}\sum_{m\in B_k}\sum_{i=1}^{n_m}\sum_{f}(x_{im}^f - c_k^f)^2 \\
&= \frac{1}{N}\sum_{k=1}^{numK}\sum_{m\in B_k}\sum_{i=1}^{n_m}\sum_{f}\big((x_{im}^f - c_m^f) + (c_m^f - c_k^f)\big)^2 \\
&= \frac{1}{N}\sum_{k=1}^{numK}\sum_{m\in B_k}\sum_{i=1}^{n_m}\sum_{f}(x_{im}^f - c_m^f)^2 + 2(x_{im}^f - c_m^f)(c_m^f - c_k^f) + (c_m^f - c_k^f)^2 \\
&= \frac{1}{N}\sum_{k=1}^{numK}\Big(\sum_{m\in B_k} n_m * \big(comp_m + \sum_f (c_m^f - c_k^f)^2\big) \\
&+ 2\sum_f\big(\sum_{m\in B_k}\sum_{i=1}^{n_m} x_{im}^f c_m^f - \sum_{m\in B_k}\sum_{i=1}^{n_m} x_{im}^f c_k^f - \sum_{m\in B_k}\sum_{i=1}^{n_m}(c_m^f)^2 + \sum_{m\in B_k}\sum_{i=1}^{n_m} c_k^f c_m^f)\Big) \\
&= \frac{1}{N}\sum_{k=1}^{numK}\Big(\sum_{m\in B_k} n_m * \big(comp_m + \sum_f (c_m^f - c_k^f)^2\big) \\
&+ 2\sum_f\big(\sum_{m\in B_k} n_m(c_m^f)^2 - N(c_k^f)^2 - \sum_{m\in B_k} n_m(c_m^f)^2 + N(c_k^f)^2)\Big) \\
&= \frac{1}{N}\sum_{k=1}^{numK}\sum_{m\in B_k} n_m * \big(comp_m + \sum_f (c_m^f - c_k^f)^2\big) \\
&= \frac{1}{N}\sum_{k=1}^{numK}\sum_{m\in B_k} n_m * \big(comp_m + d_{c_k c_m}^2\big)
\end{aligned}
$$

where

$$comp_m = \frac{1}{n_m}\sum_{i=1}^{n_m} d_{ic_m}^2$$

APPENDIX B

EXPERIMENTAL RESULTS OF THE MOCNC ALGORITHM

In this appendix, we give the detailed experimental results of the MOCNC algorithm for individual problem instances.

Some of the column headings used in all tables are explained below.

RI	: Rand Index for the solution
ARI	: Adjusted Rand Index for the solution
k	: Number of clusters found in the solution

Table B.1: Solution quality of MOCNC, Δ-MOCK, *k*-means and Single-Link for 2 dimensional generated data sets

Problem	**Number of Points**	**Number of NC Closures**	**MOCNC**						**Δ-MOCK**			***k*-means**			**Single-Link**		
			Before Improvement			**After Improvement**											
			RI	*ARI*	*k*	*RI*	*ARI*	*k*	*RI*	*ARI*	*k*	*RI*	*ARI*	*k*	*RI*	*ARI*	*k*
2d-5c-1	1555	83	1.000	1.000	6	1.000	1.000	5	1.000	1.000	5	0.977	0.934	9	0.999	0.998	10
2d-5c-2	1370	57	1.000	1.000	6	0.997	0.991	5	1.000	1.000	5	0.928	0.805	6	0.797	0.576	8
2d-5c-3	1408	48	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.981	0.943	7	0.941	0.842	5
2d-5c-4	1485	69	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.955	0.883	5	1.000	0.999	6
2d-5c-5	1921	52	1.000	1.000	5	1.000	0.999	5	1.000	1.000	5	0.971	0.909	6	0.898	0.739	10
2d-5c-6	942	37	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.936	0.801	7	0.903	0.760	4
2d-5c-7	1377	56	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.931	0.820	4	0.998	0.993	7
2d-5c-8	1416	78	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.978	0.941	5	1.000	1.000	5
2d-5c-9	1839	64	1.000	0.999	6	1.000	1.000	5	1.000	1.000	5	0.989	0.965	5	0.775	0.509	7
2d-5c-10	1358	63	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.972	0.929	4	0.998	0.995	9
2d-10c-1	3330	147	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.989	0.942	11	0.959	0.824	19
2d-10c-2	2794	121	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.990	0.956	10	0.995	0.976	11
2d-10c-3	3627	161	1.000	1.000	10	1.000	1.000	10	0.998	0.991	11	0.989	0.949	10	1.000	0.998	20
2d-10c-4	3033	147	1.000	0.999	11	1.000	0.999	11	1.000	1.000	10	0.999	0.993	10	0.990	0.955	9
2d-10c-5	2974	125	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.987	0.938	10	0.999	0.996	19
2d-10c-6	3213	125	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.997	0.984	10	0.949	0.787	13
2d-10c-7	3199	137	1.000	1.000	11	1.000	1.000	10	1.000	1.000	10	0.962	0.834	7	0.925	0.715	13
2d-10c-8	2305	91	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.969	0.872	9	0.944	0.801	14
2d-10c-9	3340	134	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.996	0.982	9	0.997	0.987	9
2d-10c-10	3137	134	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.989	0.945	11	0.999	0.993	20
2d-20c-1	5641	262	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.994	0.945	19	0.992	0.937	30
2d-20c-2	6350	230	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.988	0.887	20	0.992	0.929	22
2d-20c-3	6810	271	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.995	0.957	19	1.000	0.999	29
2d-20c-4	4235	202	1.000	1.000	20	1.000	1.000	20	0.997	0.977	19	0.990	0.917	24	0.959	0.734	40
2d-20c-5	5967	243	0.998	0.987	19	0.998	0.987	19	1.000	0.999	20	0.990	0.919	19	1.000	0.998	29
2d-20c-6	4815	253	0.999	0.989	20	0.999	0.989	20	1.000	1.000	20	0.992	0.934	20	0.996	0.970	31
2d-20c-7	5508	212	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.990	0.905	24	1.000	0.998	33
2d-20c-8	4894	259	0.997	0.977	19	0.997	0.977	19	0.997	0.977	19	0.992	0.933	24	0.984	0.884	26
2d-20c-9	6309	272	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.995	0.955	20	0.989	0.911	20
2d-20c-10	6046	264	1.000	1.000	20	1.000	1.000	20	1.000	0.999	21	0.990	0.922	21	0.986	0.895	22

Table B.2: Solution quality of MOCNC, Δ-MOCK, *k*-means and Single-Link for 5 dimensional generated data sets

Problem	**Number of Points**	**Number of NC Closures**	**MOCNC**						**Δ-MOCK**			***k*-means**			**Single-Link**		
			Before Improvement			**After Improvement**											
			RI	*ARI*	*k*	*RI*	*ARI*	*k*	*RI*	*ARI*	*k*	*RI*	*ARI*	*k*	*RI*	*ARI*	*k*
5d-5c-1	1408	38	0.998	0.993	7	0.995	0.986	5	1.000	1.000	5	0.941	0.851	4	0.823	0.615	7
5d-5c-2	1999	34	1.000	0.999	7	1.000	1.000	5	1.000	1.000	5	0.997	0.992	5	0.813	0.577	6
5d-5c-3	1097	40	0.998	0.996	8	0.999	0.998	5	1.000	1.000	5	0.995	0.987	5	0.951	0.880	9
5d-5c-4	1502	24	0.999	0.997	7	0.999	0.998	5	1.000	1.000	5	0.993	0.982	5	0.937	0.844	8
5d-5c-5	1538	38	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.977	0.939	4	0.790	0.555	10
5d-5c-6	1682	28	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.974	0.923	6	0.998	0.994	10
5d-5c-7	1351	25	1.000	0.999	6	0.997	0.992	5	1.000	1.000	5	0.994	0.984	5	0.798	0.567	7
5d-5c-8	1426	27	0.999	0.997	9	0.999	0.996	5	1.000	1.000	5	0.989	0.971	5	0.633	0.339	9
5d-5c-9	1664	34	1.000	0.999	7	1.000	1.000	5	1.000	1.000	5	0.981	0.943	6	0.631	0.327	10
5d-5c-10	1945	25	1.000	0.999	6	1.000	1.000	5	1.000	1.000	5	0.953	0.851	6	0.739	0.459	10
5d-10c-1	2780	46	0.995	0.981	9	1.000	0.999	10	1.000	1.000	10	0.969	0.868	9	0.873	0.611	15
5d-10c-2	3350	68	0.994	0.968	9	0.994	0.968	9	1.000	1.000	10	0.999	0.996	10	0.999	0.997	18
5d-10c-3	2456	63	1.000	1.000	10	0.991	0.961	9	1.000	1.000	10	0.982	0.925	9	0.972	0.887	20
5d-10c-4	2403	52	1.000	1.000	12	1.000	1.000	10	1.000	1.000	10	0.992	0.965	9	0.702	0.322	15
5d-10c-5	3687	79	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.981	0.908	10	0.783	0.416	13
5d-10c-6	2667	46	0.975	0.902	8	0.975	0.902	8	1.000	1.000	10	1.000	0.999	10	1.000	1.000	13
5d-10c-7	2602	56	0.980	0.913	9	0.969	0.869	8	1.000	1.000	10	1.000	0.999	10	1.000	0.998	18
5d-10c-8	3099	66	0.990	0.954	9	0.990	0.954	9	1.000	1.000	10	0.993	0.969	9	0.934	0.742	20
5d-10c-9	2627	56	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.995	0.975	11	0.983	0.920	15
5d-10c-10	2557	62	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.996	0.981	9	0.938	0.772	15
5d-20c-1	4615	103	0.997	0.974	18	0.997	0.974	18	1.000	1.000	20	0.995	0.960	20	1.000	1.000	22
5d-20c-2	6947	141	0.996	0.967	18	0.996	0.967	18	1.000	1.000	20	0.996	0.961	18	0.993	0.938	23
5d-20c-3	5895	113	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.996	0.968	21	1.000	0.999	24
5d-20c-4	5998	138	0.999	0.988	19	0.999	0.988	19	1.000	1.000	20	0.995	0.954	19	1.000	1.000	21
5d-20c-5	6664	138	0.999	0.989	19	0.999	0.989	19	1.000	1.000	20	0.998	0.978	20	1.000	0.999	28
5d-20c-6	5153	124	0.997	0.979	17	0.997	0.979	17	1.000	1.000	20	0.999	0.994	19	1.000	0.999	24
5d-20c-7	6613	136	0.999	0.991	19	0.999	0.991	19	1.000	1.000	20	0.994	0.949	20	1.000	1.000	26
5d-20c-8	5646	109	0.997	0.973	19	0.997	0.973	19	1.000	1.000	20	0.993	0.942	17	1.000	0.998	34
5d-20c-9	6370	182	0.997	0.973	19	0.997	0.973	19	1.000	1.000	20	0.995	0.955	22	1.000	0.998	38
5d-20c-10	6831	153	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.998	0.979	19	0.990	0.913	29

Table B.3: Solution quality of MOCNC, Δ-MOCK, *k*-means and Single-Link for 10 dimensional generated data sets

Problem	**Number of Points**	**Number of NC Closures**	**MOCNC**						**Δ-MOCK**			***k*-means**			**Single-Link**		
			Before Improvement			**After Improvement**											
			RI	*ARI*	*k*	*RI*	*ARI*	*k*	*RI*	*ARI*	*k*	*RI*	*ARI*	*k*	*RI*	*ARI*	*k*
10d-5c-1	1564	17	0.999	0.997	8	1.000	1.000	5	1.000	1.000	5	0.957	0.863	6	0.631	0.324	9
10d-5c-2	1277	24	0.999	0.997	8	0.999	0.998	5	1.000	1.000	5	0.998	0.994	5	0.854	0.661	10
10d-5c-3	2070	38	0.998	0.995	11	1.000	1.000	5	1.000	1.000	5	1.000	0.999	5	0.679	0.380	9
10d-5c-4	1357	35	0.993	0.981	19	1.000	1.000	5	1.000	1.000	5	0.993	0.981	6	0.873	0.716	9
10d-5c-5	1579	23	0.999	0.997	11	1.000	1.000	5	1.000	1.000	5	0.972	0.926	4	0.248	0.003	9
10d-5c-6	1898	28	0.999	0.997	10	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.397	0.112	10
10d-5c-7	1336	24	0.997	0.993	12	1.000	1.000	5	1.000	1.000	5	0.995	0.989	5	0.598	0.303	10
10d-5c-8	1370	23	0.997	0.992	14	1.000	1.000	5	1.000	1.000	5	0.961	0.896	6	0.660	0.382	5
10d-5c-9	1428	26	0.996	0.989	16	1.000	0.999	5	1.000	1.000	5	0.985	0.961	4	0.261	0.000	10
10d-5c-10	725	11	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.808	0.595	9
10d-10c-1	2960	52	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.985	0.929	11	1.000	0.999	11
10d-10c-2	2890	47	1.000	0.999	11	1.000	1.000	10	1.000	1.000	10	0.981	0.911	10	0.954	0.816	11
10d-10c-3	3654	70	1.000	0.998	15	1.000	1.000	10	1.000	1.000	10	0.989	0.948	9	0.969	0.863	14
10d-10c-4	3877	55	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.998	0.992	9	1.000	1.000	11
10d-10c-5	2937	48	1.000	1.000	11	1.000	1.000	10	1.000	1.000	10	0.998	0.993	9	0.851	0.557	16
10d-10c-6	2604	33	1.000	1.000	11	1.000	1.000	10	1.000	1.000	10	0.978	0.900	11	1.000	0.999	14
10d-10c-7	2285	34	0.999	0.994	24	1.000	1.000	10	1.000	1.000	10	0.982	0.926	8	0.990	0.959	19
10d-10c-8	3006	55	1.000	0.998	14	1.000	1.000	10	1.000	1.000	10	0.995	0.974	9	0.862	0.552	15
10d-10c-9	4121	94	1.000	0.999	14	1.000	1.000	10	1.000	1.000	10	0.993	0.965	11	0.722	0.326	6
10d-10c-10	2811	53	1.000	0.999	14	1.000	1.000	10	1.000	1.000	10	0.987	0.936	13	0.990	0.958	11
10d-20c-1	5870	101	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.998	0.980	18	1.000	1.000	22
10d-20c-2	6722	119	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.996	0.964	21	1.000	1.000	20
10d-20c-3	6674	101	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.996	0.964	19	1.000	1.000	20
10d-20c-4	5935	90	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.994	0.948	20	1.000	1.000	20
10d-20c-5	4634	77	1.000	0.999	22	1.000	1.000	20	1.000	1.000	20	0.994	0.956	17	1.000	1.000	21
10d-20c-6	7039	140	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.995	0.948	21	1.000	1.000	20
10d-20c-7	4846	79	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.998	0.983	20	1.000	1.000	20
10d-20c-8	6283	110	1.000	0.999	23	1.000	1.000	20	1.000	1.000	20	0.998	0.982	23	0.997	0.975	19
10d-20c-9	6569	124	1.000	0.998	28	1.000	1.000	20	1.000	1.000	20	0.997	0.975	21	1.000	1.000	20
10d-20c-10	5068	99	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.998	0.983	23	1.000	1.000	20

Table B.4: Solution quality of MOCNC, Δ-MOCK, *k*-means and Single-Link for 20 dimensional generated data sets

Problem	**Number of Points**	**Number of NC Closures**	**MOCNC**						**Δ-MOCK**			***k*-means**			**Single-Link**		
			Before Improvement			**After Improvement**											
			RI	*ARI*	*k*	*RI*	*ARI*	*k*	*RI*	*ARI*	*k*	*RI*	*ARI*	*k*	*RI*	*ARI*	*k*
20d-5c-1	1297	48	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.990	0.974	4	0.877	0.724	4
20d-5c-2	1381	49	0.999	0.996	9	1.000	1.000	5	1.000	1.000	5	0.970	0.918	6	0.794	0.574	3
20d-5c-3	1623	52	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.945	0.843	7	0.999	0.998	8
20d-5c-4	1315	55	0.999	0.997	8	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.794	0.573	4
20d-5c-5	1425	56	0.999	0.998	7	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.942	0.850	6
20d-5c-6	1104	48	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.971	0.937	3	0.998	0.996	7
20d-5c-7	2042	74	0.999	0.998	8	1.000	1.000	5	1.000	1.000	5	0.980	0.939	7	0.502	0.196	10
20d-5c-8	804	15	0.999	0.996	7	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.918	0.792	6
20d-5c-9	1645	77	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.969	0.919	4	0.968	0.916	9
20d-5c-10	1417	30	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.977	0.928	6	0.999	0.998	7
20d-10c-1	2531	76	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.982	0.913	11	1.000	1.000	11
20d-10c-2	2988	115	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.995	0.978	9	1.000	1.000	10
20d-10c-3	3546	157	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.979	0.898	8	1.000	0.999	14
20d-10c-4	2727	101	0.999	0.997	16	1.000	1.000	10	1.000	1.000	10	0.982	0.925	9	0.993	0.970	11
20d-10c-5	2720	109	1.000	0.999	11	1.000	1.000	10	1.000	1.000	10	0.992	0.968	10	0.999	0.998	19
20d-10c-6	2553	94	1.000	0.999	12	1.000	1.000	10	1.000	1.000	10	0.989	0.945	11	1.000	0.999	14
20d-10c-7	3609	139	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.990	0.952	9	0.966	0.845	10
20d-10c-8	2993	115	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	1.000	1.000	11
20d-10c-9	3558	127	1.000	0.999	12	1.000	1.000	10	1.000	1.000	10	0.998	0.987	11	1.000	0.998	17
20d-10c-10	2490	72	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.987	0.936	13	1.000	0.999	11
20d-20c-1	5891	210	1.000	0.999	24	1.000	1.000	20	1.000	1.000	20	0.999	0.991	21	1.000	1.000	20
20d-20c-2	6582	278	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.994	0.949	19	1.000	1.000	20
20d-20c-3	4904	165	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.987	0.899	19	1.000	1.000	20
20d-20c-4	6058	233	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	1.000	0.997	19	1.000	1.000	20
20d-20c-5	6332	238	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.998	0.983	22	1.000	1.000	20
20d-20c-6	5800	192	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.997	0.970	21	1.000	1.000	20
20d-20c-7	6229	246	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.995	0.958	20	1.000	1.000	20
20d-20c-8	5233	205	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.995	0.956	22	1.000	1.000	20
20d-20c-9	5735	235	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.995	0.955	21	1.000	1.000	20
20d-20c-10	4757	205	1.000	0.999	21	1.000	1.000	20	1.000	1.000	20	0.994	0.953	22	1.000	1.000	20

Table B.5: Solution quality of MOCNC, Δ-MOCK, *k*-means and Single-Link for 40 dimensional generated data sets

Problem	**Number of Points**	**Number of NC Closures**	**MOCNC**						**Δ-MOCK**			***k*-means**			**Single-Link**		
			Before Improvement			**After Improvement**											
			RI	*ARI*	*k*	*RI*	*ARI*	*k*	*RI*	*ARI*	*k*	*RI*	*ARI*	*k*	*RI*	*ARI*	*k*
40d-5c-1	1649	150	1.000	0.999	6	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.957	0.892	4
40d-5c-2	1580	154	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.931	0.836	3	0.999	0.997	9
40d-5c-3	1926	267	1.000	1.000	6	1.000	1.000	5	1.000	1.000	5	0.970	0.921	4	0.689	0.409	8
40d-5c-4	1416	151	0.999	0.999	6	1.000	1.000	5	1.000	1.000	5	0.936	0.851	3	0.773	0.546	5
40d-5c-5	1935	177	0.997	0.992	14	1.000	1.000	5	1.000	1.000	5	0.975	0.919	6	0.893	0.727	7
40d-5c-6	947	71	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.956	0.904	4	1.000	1.000	5
40d-5c-7	2039	212	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.964	0.890	6	0.969	0.915	4
40d-5c-8	1681	138	0.998	0.995	12	1.000	1.000	5	1.000	1.000	5	0.969	0.903	6	0.429	0.138	6
40d-5c-9	1273	108	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	1.000	1.000	6
40d-5c-10	1124	72	1.000	1.000	5	1.000	1.000	5	1.000	1.000	5	0.972	0.932	4	0.763	0.531	7
40d-10c-1	3365	322	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.991	0.960	9	1.000	0.999	12
40d-10c-2	3381	371	1.000	0.998	14	1.000	1.000	10	1.000	1.000	10	0.982	0.914	11	0.959	0.837	12
40d-10c-3	2929	280	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.986	0.940	8	1.000	1.000	10
40d-10c-4	3093	308	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.999	0.997	12	1.000	1.000	10
40d-10c-5	3308	288	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.991	0.953	12	1.000	1.000	10
40d-10c-6	2253	154	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.982	0.913	13	1.000	1.000	11
40d-10c-7	3678	321	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.979	0.884	15	1.000	1.000	10
40d-10c-8	2656	187	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.986	0.938	11	1.000	1.000	10
40d-10c-9	3878	433	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.991	0.953	12	1.000	1.000	10
40d-10c-10	2440	176	1.000	1.000	10	1.000	1.000	10	1.000	1.000	10	0.992	0.966	8	1.000	1.000	10
40d-20c-1	6746	693	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.994	0.951	19	1.000	1.000	20
40d-20c-2	6022	589	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.994	0.947	21	1.000	1.000	20
40d-20c-3	5561	442	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.995	0.953	23	1.000	1.000	20
40d-20c-4	6334	508	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.992	0.931	16	1.000	1.000	20
40d-20c-5	6318	610	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.993	0.935	20	1.000	1.000	20
40d-20c-6	6457	665	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.996	0.963	19	1.000	1.000	20
40d-20c-7	5501	566	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.993	0.941	19	1.000	1.000	20
40d-20c-8	5882	496	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.996	0.967	21	1.000	1.000	20
40d-20c-9	5283	438	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.992	0.944	16	1.000	1.000	20
40d-20c-10	5627	492	1.000	1.000	20	1.000	1.000	20	1.000	1.000	20	0.992	0.929	18	1.000	1.000	20

APPENDIX C

EXPERIMENTAL RESULTS OF THE MOCNC-F ALGORITHM

In this appendix, we give the detailed experimental results of the MOCNC-F algorithm for individual problem instances.

Some of the column headings used in all tables are explained below.

RdF	: Number of redundant features added to the original data set
RI	: Rand Index for the solution
ARI	: Adjusted Rand Index for the solution
k	: Number of clusters found in the solution
OF	: Number of original features selected in the solution
RF	: Number of redundant features selected in the solution

Table C.1: Solution quality of MOCNC-F and Δ-MOCK-F for 5d-5c problem setting

Problem	**# of Points**	**RdF**	**MOCNC-F**					**Δ-MOCK-F**				
			RI	*ARI*	*k*	*OF*	*RF*	*RI*	*ARI*	*k*	*OF*	*RF*
5d-5c-1	1408	1	1.000	1.000	5	5	0	0.819	0.608	3	3	0
5d-5c-2	1999	1	0.999	0.998	5	5	0	0.815	0.538	4	1	0
5d-5c-3	1097	1	1.000	0.999	5	5	0	0.740	0.488	2	3	0
5d-5c-4	1502	1	0.969	0.919	5	3	0	0.829	0.628	3	2	0
5d-5c-5	1538	1	0.999	0.998	5	5	0	0.701	0.208	6	1	0
5d-5c-6	1682	1	1.000	1.000	5	5	0	0.727	0.400	3	1	0
5d-5c-7	1351	1	0.999	0.996	6	5	0	0.806	0.488	4	1	0
5d-5c-8	1426	1	0.999	0.998	5	4	0	0.976	0.935	4	3	0
5d-5c-9	1664	1	1.000	1.000	5	5	0	0.685	0.196	4	1	0
5d-5c-10	1945	1	0.999	0.998	5	5	0	0.746	0.278	5	1	0
5d-5c-1	1408	3	1.000	1.000	5	5	0	0.819	0.608	3	3	0
5d-5c-2	1999	3	0.999	0.998	5	5	0	0.802	0.521	3	1	0
5d-5c-3	1097	3	1.000	0.999	5	5	0	0.740	0.488	2	3	0
5d-5c-4	1502	3	0.969	0.921	5	3	0	0.829	0.628	3	2	0
5d-5c-5	1538	3	1.000	1.000	5	4	0	0.701	0.208	6	1	0
5d-5c-6	1682	3	1.000	1.000	5	5	0	0.749	0.419	4	1	0
5d-5c-7	1351	3	1.000	1.000	5	5	1	0.806	0.488	4	1	0
5d-5c-8	1426	3	0.999	0.998	5	4	0	0.976	0.935	4	3	0
5d-5c-9	1664	3	1.000	1.000	5	5	0	0.685	0.196	4	1	0
5d-5c-10	1945	3	0.999	0.998	5	5	0	0.814	0.388	7	1	0
5d-5c-1	1408	5	1.000	1.000	5	5	0	0.819	0.608	3	3	0
5d-5c-2	1999	5	0.999	0.998	5	5	0	0.815	0.538	4	1	0
5d-5c-3	1097	5	1.000	0.999	5	5	0	0.737	0.457	3	1	0
5d-5c-4	1502	5	0.969	0.919	5	3	0	0.829	0.628	3	2	0
5d-5c-5	1538	5	0.999	0.998	5	5	0	0.703	0.205	7	1	0
5d-5c-6	1682	5	1.000	1.000	5	5	0	0.749	0.419	4	1	0
5d-5c-7	1351	5	0.999	0.999	5	5	1	0.806	0.488	4	1	0
5d-5c-8	1426	5	0.999	0.998	5	4	0	0.976	0.935	4	3	0
5d-5c-9	1664	5	1.000	1.000	5	5	0	0.832	0.429	8	1	0
5d-5c-10	1945	5	0.999	0.998	5	5	0	0.814	0.388	7	1	0

Table C.2: Solution quality of MOCNC-F and Δ-MOCK-F for 5d-10c problem setting

Problem	**# of Points**	**RdF**	**MOCNC-F**					**Δ-MOCK-F**				
			RI	*ARI*	*k*	*OF*	*RF*	*RI*	*ARI*	*k*	*OF*	*RF*
5d-10c-1	2780	1	1.000	0.999	10	5	0	0.757	0.252	5	1	0
5d-10c-2	3350	1	1.000	1.000	10	5	0	0.889	0.259	27	1	0
5d-10c-3	2456	1	0.997	0.988	9	5	0	0.788	0.369	4	1	0
5d-10c-4	2403	1	1.000	0.999	10	5	0	0.797	0.386	5	1	0
5d-10c-5	3687	1	1.000	1.000	10	5	0	0.818	0.331	6	1	0
5d-10c-6	2667	1	1.000	1.000	10	5	0	0.998	0.993	10	5	0
5d-10c-7	2602	1	0.997	0.987	10	5	0	0.854	0.445	7	1	0
5d-10c-8	3099	1	0.983	0.923	9	5	0	0.819	0.297	6	1	0
5d-10c-9	2627	1	1.000	0.999	11	5	0	0.768	0.263	5	1	0
5d-10c-10	2557	1	1.000	0.999	10	5	0	0.813	0.267	7	1	0
5d-10c-1	2780	3	0.999	0.998	10	5	0	0.813	0.495	4	4	0
5d-10c-2	3350	3	1.000	1.000	10	5	0	0.889	0.259	27	1	0
5d-10c-3	2456	3	0.991	0.961	9	5	0	0.983	0.931	9	5	0
5d-10c-4	2403	3	1.000	0.999	10	5	0	0.797	0.386	5	1	0
5d-10c-5	3687	3	1.000	1.000	10	5	0	0.843	0.343	9	1	0
5d-10c-6	2667	3	1.000	1.000	10	5	0	0.998	0.993	10	5	0
5d-10c-7	2602	3	0.999	0.997	10	5	0	0.827	0.428	5	1	0
5d-10c-8	3099	3	0.983	0.924	9	5	0	0.774	0.274	5	1	0
5d-10c-9	2627	3	1.000	1.000	10	5	0	0.833	0.495	5	3	0
5d-10c-10	2557	3	1.000	1.000	10	5	0	0.963	0.855	8	5	0
5d-10c-1	2780	5	0.995	0.980	9	5	0	0.813	0.495	4	4	0
5d-10c-2	3350	5	1.000	1.000	10	5	0	0.889	0.259	27	1	0
5d-10c-3	2456	5	0.997	0.986	9	5	0	0.998	0.993	10	5	0
5d-10c-4	2403	5	1.000	0.999	11	5	0	0.797	0.386	5	1	0
5d-10c-5	3687	5	1.000	1.000	10	5	0	0.818	0.331	6	1	0
5d-10c-6	2667	5	0.997	0.987	10	5	0	0.816	0.375	5	1	0
5d-10c-7	2602	5	0.999	0.996	10	5	0	0.947	0.789	8	5	0
5d-10c-8	3099	5	0.990	0.954	9	5	0	0.819	0.297	6	1	0
5d-10c-9	2627	5	1.000	1.000	10	5	0	0.797	0.310	5	1	0
5d-10c-10	2557	5	1.000	0.999	10	5	0	0.963	0.855	8	5	0

Table C.3: Solution quality of MOCNC-F and Δ-MOCK-F for 5d-20c problem setting

Problem	# of Points	RdF	MOCNC-F					Δ-MOCK-F				
			RI	*ARI*	*k*	*OF*	*RF*	*RI*	*ARI*	*k*	*OF*	*RF*
5d-20c-1	4615	1	0.997	0.978	18	4	0	0.999	0.993	20	5	0
5d-20c-2	6947	1	0.996	0.966	18	5	0	0.990	0.913	19	5	0
5d-20c-3	5895	1	1.000	1.000	20	5	0	0.999	0.994	20	5	0
5d-20c-4	5998	1	0.999	0.988	19	5	0	0.999	0.994	20	5	0
5d-20c-5	6664	1	0.993	0.942	18	4	0	0.999	0.994	20	5	0
5d-20c-6	5153	1	0.997	0.979	17	5	0	0.999	0.994	20	5	0
5d-20c-7	6613	1	0.999	0.990	19	5	0	0.891	0.315	10	1	0
5d-20c-8	5646	1	0.997	0.974	18	4	0	0.937	0.241	47	1	0
5d-20c-9	6370	1	0.995	0.957	19	4	0	0.915	0.382	15	1	0
5d-20c-10	6831	1	1.000	1.000	21	5	0	0.994	0.946	19	4	0
5d-20c-1	4615	3	0.999	0.993	19	5	0	0.999	0.993	20	5	0
5d-20c-2	6947	3	0.997	0.972	18	5	0	0.990	0.913	19	5	0
5d-20c-3	5895	3	1.000	1.000	20	5	0	0.999	0.994	20	5	0
5d-20c-4	5998	3	1.000	1.000	20	5	0	0.992	0.930	18	4	0
5d-20c-5	6664	3	0.993	0.942	18	4	0	0.999	0.994	20	5	0
5d-20c-6	5153	3	0.997	0.979	17	5	0	0.999	0.994	20	5	0
5d-20c-7	6613	3	0.999	0.991	19	5	0	0.983	0.866	18	4	0
5d-20c-8	5646	3	0.997	0.972	19	4	0	0.998	0.982	19	5	0
5d-20c-9	6370	3	0.995	0.957	19	4	0	0.999	0.995	20	5	0
5d-20c-10	6831	3	1.000	0.999	22	5	0	0.994	0.946	19	4	0
5d-20c-1	4615	5	0.997	0.978	18	4	0	0.999	0.993	20	5	0
5d-20c-2	6947	5	0.996	0.967	18	5	0	0.990	0.913	19	5	0
5d-20c-3	5895	5	1.000	1.000	20	5	0	0.999	0.994	20	5	0
5d-20c-4	5998	5	1.000	0.999	20	4	0	0.996	0.962	19	4	0
5d-20c-5	6664	5	1.000	1.000	20	5	0	0.999	0.994	20	5	0
5d-20c-6	5153	5	0.999	0.991	18	5	0	0.999	0.994	20	5	0
5d-20c-7	6613	5	1.000	0.998	20	5	0	0.929	0.312	28	1	0
5d-20c-8	5646	5	0.997	0.977	18	5	0	0.906	0.389	11	1	0
5d-20c-9	6370	5	0.995	0.957	19	4	0	0.999	0.995	20	5	0
5d-20c-10	6831	5	1.000	1.000	21	5	0	0.994	0.946	19	4	0

Table C.4: Solution quality of MOCNC-F and Δ-MOCK-F for 10d-5c problem setting

Problem	**# of Points**	**RdF**	**MOCNC-F**					**Δ-MOCK-F**				
			RI	*ARI*	*k*	*OF*	*RF*	*RI*	*ARI*	*k*	*OF*	*RF*
10d-5c-1	1564	2	1.000	1.000	5	8	0	0.697	0.275	3	1	0
10d-5c-2	1277	2	1.000	1.000	5	8	0	0.719	0.255	5	1	0
10d-5c-3	2070	2	1.000	1.000	5	6	0	0.717	0.383	3	1	0
10d-5c-4	1357	2	0.999	0.998	5	10	0	0.635	0.219	3	1	0
10d-5c-5	1579	2	1.000	1.000	5	7	0	0.569	0.265	2	1	0
10d-5c-6	1898	2	1.000	1.000	5	8	0	0.787	0.399	6	1	0
10d-5c-7	1336	2	1.000	1.000	5	8	0	0.868	0.675	5	1	0
10d-5c-8	1370	2	1.000	1.000	5	8	0	0.724	0.226	6	1	0
10d-5c-9	1428	2	1.000	1.000	5	7	0	0.709	0.429	2	1	0
10d-5c-10	725	2	1.000	1.000	5	6	0	0.746	0.393	4	1	0
10d-5c-1	1564	5	1.000	1.000	5	8	0	0.997	0.992	5	8	0
10d-5c-2	1277	5	1.000	1.000	5	9	0	0.736	0.324	4	1	0
10d-5c-3	2070	5	1.000	1.000	5	7	0	0.717	0.383	3	1	0
10d-5c-4	1357	5	0.999	0.998	5	10	0	0.635	0.219	3	1	0
10d-5c-5	1579	5	1.000	1.000	5	7	0	0.997	0.992	5	10	0
10d-5c-6	1898	5	1.000	1.000	5	8	0	0.738	0.275	4	1	0
10d-5c-7	1336	5	1.000	1.000	5	6	0	0.837	0.634	3	1	0
10d-5c-8	1370	5	1.000	1.000	5	8	0	0.724	0.226	6	1	0
10d-5c-9	1428	5	1.000	1.000	5	9	0	0.709	0.429	2	1	0
10d-5c-10	725	5	1.000	1.000	5	6	0	0.691	0.419	2	2	0
10d-5c-1	1564	10	1.000	1.000	5	8	0	0.998	0.993	5	9	0
10d-5c-2	1277	10	1.000	1.000	5	9	0	0.997	0.990	5	8	0
10d-5c-3	2070	10	1.000	1.000	5	9	0	0.998	0.994	5	8	0
10d-5c-4	1357	10	1.000	1.000	5	9	0	0.997	0.992	5	10	0
10d-5c-5	1579	10	1.000	1.000	5	10	0	0.997	0.992	5	10	0
10d-5c-6	1898	10	1.000	1.000	5	8	0	0.787	0.399	6	1	0
10d-5c-7	1336	10	1.000	1.000	5	7	0	0.997	0.992	5	8	0
10d-5c-8	1370	10	1.000	1.000	5	8	0	0.997	0.994	5	8	0
10d-5c-9	1428	10	1.000	1.000	5	10	0	0.997	0.993	5	6	0
10d-5c-10	725	10	1.000	1.000	5	10	0	0.668	0.342	3	1	0

Table C.5: Solution quality of MOCNC-F and Δ-MOCK-F for 10d-10c problem setting

Problem	**# of Points**	**RdF**	**MOCNC-F**					**Δ-MOCK-F**				
			RI	*ARI*	*k*	*OF*	*RF*	*RI*	*ARI*	*k*	*OF*	*RF*
10d-10c-1	2960	2	1.000	1.000	10	10	0	0.999	0.994	10	10	0
10d-10c-2	2890	2	1.000	1.000	10	9	0	0.783	0.271	5	1	0
10d-10c-3	3654	2	1.000	1.000	10	9	0	0.999	0.995	10	9	0
10d-10c-4	3877	2	1.000	1.000	10	10	0	0.999	0.995	10	9	0
10d-10c-5	2937	2	1.000	1.000	10	7	0	0.839	0.250	11	1	0
10d-10c-6	2604	2	1.000	1.000	10	10	0	0.782	0.346	4	1	0
10d-10c-7	2285	2	1.000	1.000	10	10	0	0.742	0.309	4	1	0
10d-10c-8	3006	2	1.000	1.000	10	10	0	0.845	0.376	7	1	0
10d-10c-9	4121	2	1.000	1.000	10	9	0	0.815	0.296	7	1	0
10d-10c-10	2811	2	1.000	1.000	10	10	0	0.980	0.916	8	9	0
10d-10c-1	2960	5	1.000	1.000	10	9	0	0.999	0.994	10	8	0
10d-10c-2	2890	5	1.000	1.000	10	8	0	0.999	0.994	10	10	0
10d-10c-3	3654	5	1.000	1.000	10	9	0	0.968	0.859	9	9	0
10d-10c-4	3877	5	1.000	1.000	10	10	0	0.999	0.995	10	9	0
10d-10c-5	2937	5	1.000	1.000	10	9	0	0.999	0.994	10	8	0
10d-10c-6	2604	5	1.000	1.000	10	9	0	0.782	0.346	4	1	0
10d-10c-7	2285	5	1.000	0.999	10	8	0	0.998	0.992	10	9	0
10d-10c-8	3006	5	1.000	1.000	10	10	0	0.808	0.229	6	1	0
10d-10c-9	4121	5	1.000	1.000	10	10	0	0.815	0.296	7	1	0
10d-10c-10	2811	5	1.000	1.000	10	10	0	0.999	0.993	10	9	0
10d-10c-1	2960	10	1.000	1.000	10	10	0	0.999	0.994	10	9	0
10d-10c-2	2890	10	1.000	1.000	10	8	0	0.783	0.271	5	1	0
10d-10c-3	3654	10	1.000	1.000	10	10	0	0.999	0.995	10	9	0
10d-10c-4	3877	10	1.000	1.000	10	10	0	0.999	0.995	10	10	0
10d-10c-5	2937	10	1.000	1.000	10	10	0	0.956	0.830	9	8	0
10d-10c-6	2604	10	1.000	1.000	10	10	0	0.999	0.994	10	10	0
10d-10c-7	2285	10	1.000	1.000	10	8	0	0.998	0.992	10	9	0
10d-10c-8	3006	10	1.000	1.000	10	10	0	0.923	0.705	6	7	0
10d-10c-9	4121	10	1.000	1.000	10	10	0	0.800	0.304	6	1	0
10d-10c-10	2811	10	1.000	1.000	10	10	0	0.999	0.993	10	10	0

Table C.6: Solution quality of MOCNC-F and Δ-MOCK-F for 10d-20c problem setting

Problem	**# of Points**	**RdF**	**MOCNC-F**					**Δ-MOCK-F**				
			RI	*ARI*	*k*	*OF*	*RF*	*RI*	*ARI*	*k*	*OF*	*RF*
10d-20c-1	5870	2	1.000	1.000	20	6	0	0.999	0.994	20	7	0
10d-20c-2	6722	2	1.000	1.000	20	7	0	0.999	0.995	20	7	0
10d-20c-3	6674	2	1.000	1.000	20	7	0	0.999	0.995	20	7	0
10d-20c-4	5935	2	1.000	1.000	20	10	0	0.999	0.994	20	8	0
10d-20c-5	4634	2	1.000	1.000	20	10	0	0.999	0.993	20	7	0
10d-20c-6	7039	2	1.000	1.000	20	7	0	0.999	0.995	20	7	0
10d-20c-7	4846	2	1.000	1.000	20	10	0	0.999	0.993	20	6	0
10d-20c-8	6283	2	1.000	1.000	20	10	0	0.999	0.994	20	6	0
10d-20c-9	6569	2	1.000	1.000	20	7	0	0.999	0.995	20	6	0
10d-20c-10	5068	2	1.000	0.999	21	10	0	0.999	0.994	20	7	0
10d-20c-1	5870	5	1.000	1.000	20	10	0	0.999	0.994	20	7	0
10d-20c-2	6722	5	1.000	1.000	20	10	0	0.999	0.995	20	7	0
10d-20c-3	6674	5	1.000	1.000	20	10	0	0.999	0.995	20	7	0
10d-20c-4	5935	5	1.000	1.000	20	10	0	0.999	0.994	20	7	0
10d-20c-5	4634	5	1.000	1.000	20	10	0	0.999	0.993	20	7	0
10d-20c-6	7039	5	1.000	1.000	20	8	0	0.999	0.995	20	7	0
10d-20c-7	4846	5	1.000	1.000	20	10	0	0.999	0.993	20	6	0
10d-20c-8	6283	5	1.000	1.000	20	7	0	0.999	0.994	20	7	0
10d-20c-9	6569	5	1.000	1.000	20	10	0	0.999	0.995	20	6	0
10d-20c-10	5068	5	1.000	1.000	20	10	0	0.999	0.994	20	8	0
10d-20c-1	5870	10	1.000	0.998	21	10	0	0.999	0.994	20	7	0
10d-20c-2	6722	10	1.000	1.000	20	10	0	0.999	0.995	20	8	0
10d-20c-3	6674	10	1.000	1.000	20	8	0	0.999	0.995	20	6	0
10d-20c-4	5935	10	1.000	1.000	20	10	0	0.999	0.994	20	7	0
10d-20c-5	4634	10	1.000	1.000	20	8	0	0.999	0.993	20	6	0
10d-20c-6	7039	10	1.000	1.000	20	5	0	0.999	0.995	20	7	0
10d-20c-7	4846	10	1.000	1.000	20	10	0	0.999	0.993	20	6	0
10d-20c-8	6283	10	1.000	0.999	21	10	0	0.999	0.994	20	8	0
10d-20c-9	6569	10	1.000	1.000	20	10	0	0.999	0.995	20	6	0
10d-20c-10	5068	10	1.000	1.000	20	10	0	0.999	0.994	20	8	0

Table C.7: Solution quality of MOCNC-F and Δ-MOCK-F for 20d-5c problem setting

Problem	**# of Points**	**RdF**	**MOCNC-F**					**Δ-MOCK-F**				
			RI	*ARI*	*k*	*OF*	*RF*	*RI*	*ARI*	*k*	*OF*	*RF*
20d-5c-1	1297	4	1.000	1.000	5	12	0	0.997	0.991	5	10	0
20d-5c-2	1381	4	1.000	1.000	5	12	0	0.997	0.993	5	13	0
20d-5c-3	1623	4	1.000	1.000	5	15	0	0.998	0.993	5	11	0
20d-5c-4	1315	4	1.000	1.000	5	14	0	0.956	0.891	4	14	0
20d-5c-5	1425	4	1.000	1.000	5	15	0	0.997	0.992	5	11	0
20d-5c-6	1104	4	1.000	1.000	5	12	0	0.996	0.992	5	10	0
20d-5c-7	2042	4	1.000	1.000	5	15	0	0.998	0.994	5	16	0
20d-5c-8	804	4	1.000	1.000	5	10	0	0.995	0.986	5	11	0
20d-5c-9	1645	4	1.000	1.000	5	12	0	0.998	0.993	5	10	0
20d-5c-10	1417	4	1.000	1.000	5	12	0	0.997	0.992	5	13	0
20d-5c-1	1297	10	1.000	1.000	5	18	1	0.997	0.993	5	13	0
20d-5c-2	1381	10	1.000	1.000	5	15	0	0.994	0.984	5	6	0
20d-5c-3	1623	10	1.000	1.000	5	15	1	0.998	0.993	5	12	0
20d-5c-4	1315	10	1.000	1.000	5	17	3	0.997	0.993	5	16	0
20d-5c-5	1425	10	1.000	1.000	5	17	2	0.997	0.991	5	11	0
20d-5c-6	1104	10	1.000	1.000	5	14	0	0.996	0.992	5	11	0
20d-5c-7	2042	10	1.000	1.000	5	17	1	0.998	0.994	5	14	0
20d-5c-8	804	10	0.999	0.998	5	11	0	0.995	0.986	5	12	0
20d-5c-9	1645	10	1.000	1.000	5	16	0	0.998	0.994	5	13	0
20d-5c-10	1417	10	1.000	1.000	5	18	2	0.997	0.992	5	16	1
20d-5c-1	1297	20	1.000	1.000	5	17	3	0.997	0.993	5	17	1
20d-5c-2	1381	20	1.000	1.000	5	19	2	0.996	0.991	5	15	1
20d-5c-3	1623	20	1.000	1.000	5	17	6	0.998	0.993	5	17	1
20d-5c-4	1315	20	0.999	0.997	5	17	3	0.997	0.993	5	18	1
20d-5c-5	1425	20	1.000	1.000	5	18	7	0.997	0.991	5	19	1
20d-5c-6	1104	20	1.000	1.000	5	19	7	0.996	0.991	5	15	1
20d-5c-7	2042	20	1.000	1.000	5	19	2	0.998	0.994	5	17	1
20d-5c-8	804	20	1.000	1.000	5	16	8	0.995	0.985	5	16	1
20d-5c-9	1645	20	1.000	1.000	5	18	6	0.997	0.991	5	14	0
20d-5c-10	1417	20	1.000	1.000	5	15	2	0.997	0.992	5	19	1

Table C.8: Solution quality of MOCNC-F and Δ-MOCK-F for 20d-10c problem setting

Problem	**# of Points**	**RdF**	**MOCNC-F**					**Δ-MOCK-F**				
			RI	*ARI*	*k*	*OF*	*RF*	*RI*	*ARI*	*k*	*OF*	*RF*
20d-10c-1	2531	4	1.000	1.000	10	13	0	0.998	0.993	10	11	0
20d-10c-2	2988	4	1.000	1.000	10	14	0	0.999	0.994	10	12	0
20d-10c-3	3546	4	1.000	1.000	10	12	0	0.999	0.994	10	15	0
20d-10c-4	2727	4	1.000	1.000	10	12	0	0.997	0.989	10	7	0
20d-10c-5	2720	4	1.000	1.000	10	14	0	0.999	0.994	10	15	0
20d-10c-6	2553	4	1.000	1.000	10	14	0	0.790	0.256	6	1	0
20d-10c-7	3609	4	1.000	1.000	10	16	0	0.999	0.994	10	13	0
20d-10c-8	2993	4	1.000	1.000	10	15	0	0.999	0.994	10	11	0
20d-10c-9	3558	4	1.000	1.000	10	16	0	0.999	0.994	10	13	0
20d-10c-10	2490	4	1.000	1.000	10	12	0	0.998	0.992	10	11	0
20d-10c-1	2531	10	1.000	1.000	10	14	0	0.998	0.993	10	13	0
20d-10c-2	2988	10	1.000	1.000	10	19	2	0.999	0.994	10	11	0
20d-10c-3	3546	10	1.000	1.000	10	15	0	0.999	0.994	10	15	0
20d-10c-4	2727	10	1.000	1.000	10	12	0	0.999	0.994	10	10	0
20d-10c-5	2720	10	1.000	1.000	10	19	4	0.998	0.994	10	15	0
20d-10c-6	2553	10	1.000	1.000	10	19	1	0.998	0.993	10	9	0
20d-10c-7	3609	10	1.000	1.000	10	17	1	0.999	0.994	10	13	0
20d-10c-8	2993	10	1.000	1.000	10	16	0	0.999	0.994	10	12	0
20d-10c-9	3558	10	1.000	1.000	10	19	0	0.999	0.994	10	13	0
20d-10c-10	2490	10	1.000	1.000	10	20	2	0.998	0.992	10	13	0
20d-10c-1	2531	20	1.000	1.000	10	19	6	0.996	0.983	19	10	0
20d-10c-2	2988	20	1.000	1.000	10	19	5	0.999	0.993	10	19	1
20d-10c-3	3546	20	1.000	1.000	10	19	3	0.999	0.993	10	13	0
20d-10c-4	2727	20	0.979	0.919	8	16	4	0.998	0.994	10	19	1
20d-10c-5	2720	20	0.976	0.907	7	18	9	0.998	0.994	10	15	0
20d-10c-6	2553	20	1.000	1.000	10	18	4	0.998	0.993	10	19	1
20d-10c-7	3609	20	1.000	1.000	10	16	2	0.999	0.994	10	19	1
20d-10c-8	2993	20	1.000	1.000	10	18	6	0.999	0.994	10	18	1
20d-10c-9	3558	20	1.000	1.000	10	18	7	0.999	0.994	10	18	1
20d-10c-10	2490	20	1.000	1.000	10	20	2	0.998	0.992	10	19	1

Table C.9: Solution quality of MOCNC-F and Δ-MOCK-F for 20d-20c problem setting

Problem	**# of Points**	**RdF**	**MOCNC-F**					**Δ-MOCK-F**				
			RI	*ARI*	*k*	*OF*	*RF*	*RI*	*ARI*	*k*	*OF*	*RF*
20d-20c-1	5891	4	1.000	1.000	20	10	0	0.999	0.994	20	8	0
20d-20c-2	6582	4	1.000	1.000	20	10	0	0.999	0.994	20	8	0
20d-20c-3	4904	4	1.000	1.000	20	12	0	0.999	0.994	20	8	0
20d-20c-4	6058	4	1.000	1.000	20	10	0	0.999	0.994	20	7	0
20d-20c-5	6332	4	1.000	1.000	20	10	0	0.999	0.994	20	8	0
20d-20c-6	5800	4	1.000	1.000	20	13	0	0.999	0.994	20	8	0
20d-20c-7	6229	4	1.000	1.000	20	12	0	0.999	0.994	20	8	0
20d-20c-8	5233	4	1.000	1.000	20	13	0	0.999	0.994	20	7	0
20d-20c-9	5735	4	1.000	1.000	20	13	0	0.999	0.994	20	9	0
20d-20c-10	4757	4	1.000	1.000	20	13	0	0.999	0.994	20	7	0
20d-20c-1	5891	10	1.000	1.000	20	16	0	0.999	0.994	20	13	0
20d-20c-2	6582	10	1.000	1.000	20	13	0	0.999	0.994	20	10	0
20d-20c-3	4904	10	1.000	1.000	20	16	0	0.999	0.994	20	10	0
20d-20c-4	6058	10	1.000	1.000	20	16	0	0.999	0.994	20	9	0
20d-20c-5	6332	10	1.000	1.000	20	16	0	0.999	0.994	20	8	0
20d-20c-6	5800	10	1.000	1.000	20	18	0	0.999	0.994	20	7	0
20d-20c-7	6229	10	1.000	1.000	20	16	0	0.999	0.994	20	12	0
20d-20c-8	5233	10	1.000	1.000	20	15	0	0.999	0.994	20	9	0
20d-20c-9	5735	10	1.000	1.000	20	16	1	0.999	0.994	20	8	0
20d-20c-10	4757	10	1.000	1.000	20	17	0	0.999	0.994	20	9	0
20d-20c-1	5891	20	1.000	1.000	20	18	0	0.999	0.994	20	20	1
20d-20c-2	6582	20	1.000	1.000	20	16	0	0.999	0.994	20	20	1
20d-20c-3	4904	20	1.000	1.000	20	19	1	0.999	0.994	20	19	1
20d-20c-4	6058	20	1.000	1.000	20	19	0	0.999	0.994	20	19	1
20d-20c-5	6332	20	1.000	1.000	20	19	2	0.999	0.994	20	19	1
20d-20c-6	5800	20	1.000	1.000	20	20	1	0.999	0.994	20	20	1
20d-20c-7	6229	20	1.000	1.000	20	18	1	0.999	0.995	20	20	1
20d-20c-8	5233	20	1.000	1.000	20	20	1	0.999	0.994	20	19	1
20d-20c-9	5735	20	1.000	1.000	20	18	0	0.999	0.994	20	18	1
20d-20c-10	4757	20	1.000	1.000	20	18	0	0.999	0.994	20	19	1

Table C.10: Solution quality of MOCNC-F and Δ-MOCK-F for 40d-5c problem setting

Problem	# of Points	RdF	MOCNC-F					Δ-MOCK-F				
			RI	*ARI*	*k*	*OF*	*RF*	*RI*	*ARI*	*k*	*OF*	*RF*
40d-5c-1	1649	8	1.000	1.000	5	26	0	0.998	0.994	5	19	0
40d-5c-2	1580	8	1.000	1.000	5	34	2	0.997	0.994	5	19	0
40d-5c-3	1926	8	1.000	1.000	5	29	3	0.998	0.994	5	17	0
40d-5c-4	1416	8	1.000	1.000	5	29	1	0.997	0.993	5	16	0
40d-5c-5	1935	8	1.000	1.000	5	26	1	0.998	0.994	5	18	0
40d-5c-6	947	8	1.000	1.000	5	29	0	0.995	0.988	5	13	0
40d-5c-7	2039	8	1.000	1.000	5	31	0	0.998	0.994	5	19	0
40d-5c-8	1681	8	1.000	1.000	5	29	0	0.998	0.993	5	18	0
40d-5c-9	1273	8	1.000	1.000	5	35	1	0.997	0.992	5	17	0
40d-5c-10	1124	8	1.000	1.000	5	32	1	0.997	0.991	5	24	0
40d-5c-1	1649	20	1.000	1.000	5	34	7	0.998	0.994	5	24	1
40d-5c-2	1580	20	1.000	1.000	5	27	6	0.997	0.993	5	23	1
40d-5c-3	1926	20	1.000	1.000	5	30	5	0.998	0.994	5	27	1
40d-5c-4	1416	20	1.000	1.000	5	29	4	0.997	0.992	5	26	0
40d-5c-5	1935	20	1.000	1.000	5	30	6	0.998	0.994	5	25	0
40d-5c-6	947	20	1.000	1.000	5	31	8	0.995	0.988	5	19	1
40d-5c-7	2039	20	1.000	1.000	5	29	3	0.998	0.994	5	27	1
40d-5c-8	1681	20	1.000	1.000	5	31	5	0.998	0.993	5	22	0
40d-5c-9	1273	20	1.000	1.000	5	30	3	0.997	0.992	5	27	2
40d-5c-10	1124	20	1.000	1.000	5	32	7	0.997	0.991	5	24	1
40d-5c-1	1649	40	1.000	1.000	5	34	12	0.997	0.993	5	33	8
40d-5c-2	1580	40	1.000	1.000	5	29	14	0.997	0.992	5	30	5
40d-5c-3	1926	40	0.720	0.452	3	29	13	0.990	0.973	7	29	5
40d-5c-4	1416	40	1.000	1.000	5	28	10	0.996	0.990	5	33	6
40d-5c-5	1935	40	1.000	1.000	5	33	14	0.997	0.992	5	30	8
40d-5c-6	947	40	1.000	1.000	5	26	10	0.996	0.991	5	34	7
40d-5c-7	2039	40	1.000	1.000	5	31	14	0.998	0.993	5	30	4
40d-5c-8	1681	40	1.000	1.000	5	30	12	0.998	0.993	5	32	4
40d-5c-9	1273	40	1.000	1.000	5	31	17	0.996	0.990	5	29	5
40d-5c-10	1124	40	1.000	1.000	5	29	10	0.995	0.988	5	32	5

Table C.11: Solution quality of MOCNC-F and Δ-MOCK-F for 40d-10c problem setting

Problem	**# of Points**	**RdF**	**MOCNC-F**					**Δ-MOCK-F**				
			RI	*ARI*	*k*	*OF*	*RF*	*RI*	*ARI*	*k*	*OF*	*RF*
40d-10c-1	3365	8	1.000	1.000	10	30	0	0.999	0.995	10	21	0
40d-10c-2	3381	8	1.000	1.000	10	31	2	0.999	0.994	10	25	0
40d-10c-3	2929	8	1.000	1.000	10	29	0	0.999	0.994	10	17	0
40d-10c-4	3093	8	1.000	1.000	10	25	0	0.999	0.995	10	21	0
40d-10c-5	3308	8	1.000	1.000	10	27	0	0.999	0.994	10	23	0
40d-10c-6	2253	8	1.000	1.000	10	29	0	0.998	0.992	10	18	0
40d-10c-7	3678	8	1.000	1.000	10	27	1	0.999	0.994	10	19	0
40d-10c-8	2656	8	1.000	1.000	10	31	0	0.999	0.994	10	20	0
40d-10c-9	3878	8	1.000	1.000	10	24	0	0.999	0.995	10	20	0
40d-10c-10	2440	8	1.000	1.000	10	31	1	0.998	0.993	10	19	0
40d-10c-1	3365	20	1.000	1.000	10	35	3	0.999	0.994	10	29	1
40d-10c-2	3381	20	1.000	1.000	10	31	3	0.999	0.995	10	32	2
40d-10c-3	2929	20	1.000	1.000	10	32	3	0.999	0.994	10	23	1
40d-10c-4	3093	20	1.000	1.000	10	37	3	0.999	0.994	10	22	0
40d-10c-5	3308	20	1.000	1.000	10	36	5	0.999	0.994	10	30	2
40d-10c-6	2253	20	1.000	1.000	10	33	3	0.998	0.992	10	28	1
40d-10c-7	3678	20	1.000	1.000	10	33	4	0.999	0.994	10	26	1
40d-10c-8	2656	20	1.000	1.000	10	29	4	0.998	0.993	10	27	1
40d-10c-9	3878	20	1.000	1.000	10	31	2	0.999	0.995	10	33	2
40d-10c-10	2440	20	1.000	1.000	10	33	7	0.998	0.993	10	29	1
40d-10c-1	3365	40	1.000	1.000	10	31	26	0.990	0.954	9	31	6
40d-10c-2	3381	40	1.000	1.000	10	35	14	0.994	0.975	9	33	6
40d-10c-3	2929	40	0.998	0.992	11	35	24	0.990	0.958	9	28	4
40d-10c-4	3093	40	1.000	1.000	10	33	19	0.999	0.995	10	32	4
40d-10c-5	3308	40	0.999	0.994	11	32	12	0.999	0.994	11	31	6
40d-10c-6	2253	40	1.000	1.000	10	31	16	0.998	0.992	10	31	6
40d-10c-7	3678	40	1.000	1.000	10	35	11	0.999	0.994	10	31	5
40d-10c-8	2656	40	1.000	1.000	10	33	14	0.988	0.949	8	28	5
40d-10c-9	3878	40	1.000	1.000	10	31	19	0.993	0.967	9	31	6
40d-10c-10	2440	40	1.000	1.000	10	33	16	0.975	0.900	8	33	6

Table C.12: Solution quality of MOCNC-F and Δ-MOCK-F for 40d-20c problem setting

Problem	**# of Points**	**RdF**	**MOCNC-F**					**Δ-MOCK-F**				
			RI	*ARI*	*k*	*OF*	*RF*	*RI*	*ARI*	*k*	*OF*	*RF*
40d-20c-1	6746	8	1.000	1.000	20	29	0	0.999	0.995	20	13	0
40d-20c-2	6022	8	1.000	1.000	20	27	0	0.999	0.995	20	17	0
40d-20c-3	5561	8	1.000	1.000	20	27	0	0.999	0.994	20	18	0
40d-20c-4	6334	8	1.000	1.000	20	25	0	0.999	0.994	20	11	0
40d-20c-5	6318	8	1.000	1.000	20	37	1	0.999	0.995	20	15	0
40d-20c-6	6457	8	1.000	1.000	20	27	0	0.999	0.994	20	11	0
40d-20c-7	5501	8	1.000	1.000	20	25	0	0.999	0.994	20	15	0
40d-20c-8	5882	8	1.000	1.000	20	28	0	0.999	0.994	20	12	0
40d-20c-9	5283	8	1.000	1.000	20	32	0	0.999	0.994	20	17	0
40d-20c-10	5627	8	1.000	1.000	20	27	0	0.999	0.994	20	17	0
40d-20c-1	6746	20	1.000	1.000	20	31	0	0.999	0.995	20	31	2
40d-20c-2	6022	20	1.000	1.000	20	34	2	0.999	0.995	20	33	2
40d-20c-3	5561	20	1.000	1.000	20	36	1	0.999	0.994	20	31	2
40d-20c-4	6334	20	1.000	1.000	20	33	1	0.999	0.994	20	29	1
40d-20c-5	6318	20	1.000	1.000	20	35	3	0.999	0.995	20	31	1
40d-20c-6	6457	20	1.000	1.000	20	34	3	0.999	0.994	20	28	1
40d-20c-7	5501	20	1.000	1.000	20	32	2	0.999	0.994	20	29	1
40d-20c-8	5882	20	1.000	1.000	20	34	4	0.999	0.994	20	31	1
40d-20c-9	5283	20	1.000	1.000	20	32	3	0.999	0.994	20	29	1
40d-20c-10	5627	20	1.000	1.000	20	34	2	0.999	0.994	20	33	2
40d-20c-1	6746	40	1.000	1.000	20	29	14	0.999	0.995	20	30	5
40d-20c-2	6022	40	1.000	1.000	20	32	18	0.999	0.995	20	30	3
40d-20c-3	5561	40	1.000	1.000	20	34	14	0.999	0.994	20	28	3
40d-20c-4	6334	40	1.000	1.000	20	32	11	0.999	0.994	20	31	6
40d-20c-5	6318	40	1.000	1.000	20	33	11	0.999	0.994	20	29	5
40d-20c-6	6457	40	1.000	1.000	20	28	16	0.999	0.994	20	32	4
40d-20c-7	5501	40	1.000	1.000	20	33	15	0.999	0.994	20	33	6
40d-20c-8	5882	40	1.000	1.000	20	34	14	0.999	0.994	20	31	5
40d-20c-9	5283	40	1.000	1.000	20	28	13	0.999	0.994	20	30	7
40d-20c-10	5627	40	1.000	1.000	20	31	9	0.999	0.994	20	31	5