DATA MINING FOR REGIONAL AND GRAPH-STRUCTURED DATA OBJECTS

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

BY

DERYA DİNLER

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
INDUSTRIAL ENGINEERING

MAY 2019

Approval of the thesis:

DATA MINING FOR REGIONAL AND GRAPH-STRUCTURED DATA OBJECTS

submitted by **DERYA DİNLER** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Industrial Engineering Department, Middle East Technical University** by,

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

Prof. Dr. Yasemin Serin
Head of Department, **Industrial Engineering** ________________

Assist. Prof. Dr. Mustafa Kemal Tural
Supervisor, **Industrial Engineering, METU** ________________

Prof. Dr. Nur Evin Özdemirel ________________
Co-supervisor, **Industrial Engineering, METU**

Examining Committee Members:

Prof. Dr. Sinan Gürel
Industrial Engineering, METU ________________

Assist. Prof. Dr. Mustafa Kemal Tural
Industrial Engineering, METU ________________

Assoc. Prof. Dr. Cem İyigün
Industrial Engineering, METU ________________

Assoc. Prof. Dr. Ayşegül Altın Kayhan
Industrial Engineering, TOBB ETÜ ________________

Assist. Prof. Dr. Mustafa Gökçe Baydoğan
Industrial Engineering, Boğaziçi University ________________

Date:

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

Name, Surname: Derya Dinler

Signature :

ABSTRACT

DATA MINING FOR REGIONAL AND GRAPH-STRUCTURED DATA OBJECTS

Dinler, Derya
Ph.D., Department of Industrial Engineering
Supervisor: Assist. Prof. Dr. Mustafa Kemal Tural
Co-Supervisor : Prof. Dr. Nur Evin Özdemirel

May 2019, 204 pages

Three research problems are addressed in this study. The first one is a semi-supervised clustering problem with instance-level constraints where each data object is either a closed convex bounded polytope or a closed disk. We first model the problem of computing the centroid of a given cluster as a second order cone programming problem. Also, a subgradient algorithm is adopted for its faster solution. We then propose a mixed-integer second order cone programming formulation and six heuristic approaches for the considered clustering problem. Finally, we compare solution approaches in terms of computational time and quality on randomly generated and real life datasets.

The second problem deals with mining a single graph to find central group of nodes of the graph. For the identification of central nodes, we utilize the group betweenness centrality (GBC) measure. We propose a method that first computes upper and lower bounds on the GBC of several groups. The method then eliminates groups with upper bounds that are lower than the maximum lower bound obtained to find candidates for the optimal group. Finally, an approximating or the optimal group can be returned to the user. We conduct computational experiments with randomly generated and real life networks to test the performance of the proposed method.

The last problem is the clustering of m-ary trees where nodes are unweighted, edges are unweighted or weighted, and node correspondence is known. To measure the distance between two trees, we utilize vertex/edge overlap (VEO) and graph edit distance (GED) measures from the literature. To find a representative (centroid) tree for a given set of trees, we propose exact and heuristic solution approaches. We test our algorithms on randomly generated and real life datasets.

Keywords: data mining, semi-supervised clustering, centrality, clustering, regional data, group betweennes centrality, tree-structured data, vertex/edge overlap, graph edit distance, optimization, heuristics

ÖZ

BÖLGESEL VE ÇİZGE-YAPILI VERİ NESNELERİ İÇİN VERİ MADENCİLİĞİ

Dinler, Derya
Doktora, Endüstri Mühendisliği Bölümü
Tez Yöneticisi: Dr. Öğr. Üyesi. Mustafa Kemal Tural
Ortak Tez Yöneticisi : Prof. Dr. Nur Evin Özdemirel

Mayıs 2019 , 204 sayfa

Bu çalışmada üç problem ele alınmıştır. İlk problem veri nesnelerinin kapalı konveks sınırlı politoplar ya da kapalı diskler olduğu durumlar için yarı gözetimli kümeleme problemidir. Kümeleme öncesi belirtilmiş bir takım nesne düzeyi kısıtlar olduğu düşünülmektedir. Verilen bir küme için küme merkezi bulma problemi ikinci dereceden konik programlama yardımıyla formüle edilmiştir. Ayrıca hızlı şekilde çözümler elde etmek için alt gradyan metodu yardımıyla da çözülmüştür. Sonrasında ise ele alınan kümele problemi için karmaşık tamsayılı ikinci dereceden konik proramlama formülasyonu ve altı tane sezgisel yöntem önerilmiştir. Son olarak, çözüm yöntemleri çözüm kalitesi ve çözüm zamanı açısından hem rassal olarak üretilen hem de gerçek hayat veri setleri kullanılarak karşılaştırılmıştır.

İkinci problem verilen bir çizgede en merkezi düğüm grubunu bulma problemidir. Merkezi düğümleri bulmak için grup aradalık merkeziliği (GAM) kullanılmıştır. Problemi çözmek için önerilen yöntemde istenen tüm düğüm gruplarının GAM değerine alt ve üst sınırlar hesaplanır, üst sınırı bulunan en büyük alt sınırdan küçük olan grup-lar elenir ve optimal düğüm grubu olabilecek adaylar kalır. Geliştirilen metodun performansını ölçmek için hem rassal olarak üretilmiş hem de gerçek hayattan ağlarla sayısal çalışmalar yapılmıştır.

Son problem m-li ağaç veri nesnelerinin kümelenmesidir. Ağaçlardaki düğümler ağırlıksızken, kenarlar ağırlıksız ya da ağırlıklı olabilir. Ayrıca herhangi bir ağaç üzerinde verilen bir düğümün başka bir ağaç üzerinde hangi düğüme denk olduğunun bilindiği varsayılmaktadır. Problemin çözümü için k-means tabanlı yöntemler önerilmiştir. İki ağaç arasındaki mesafeyi ölçmek için literatürde kullanılan düğüm/kenar örtüşmesi (DKÖ) ve çizge düzeltme uzaklığı (ÇDU) ölçüleri kullanılmıştır. Verilen bir ağaç kümesini temsil edecek ağacı bulma problemi için matematiksel formülasyonlar ve bunlar için etkili çözüm yöntemleri önerilmiştir. Geliştirilen yöntemlerin performanslarını ölçmek için hem rassal olarak üretilen hem de gerçek hayattan veri setleri kullanılmıştır.

Anahtar Kelimeler: veri madenciliği, yarı-gözetimli kümeleme, merkezilik, kümeleme, bölgesel veri, grup aradalık merkeziliği, ağaç-yapılı veri, düğüm/kenar örtüşmesi, çizge düzeltme uzaklığı, eniyileme, sezgiseller

To the future, no matter what it holds...

ACKNOWLEDGMENTS

I gratefully acknowledge those who have helped me, academically and psychologically, in completing this thesis, and apologize in advance to the ones whose names are not mentioned in this limited page.

First of all, I would like to thank my advisors Assist. Prof. Dr. Mustafa Kemal Tural and Prof. Dr. Nur Evin Özdemirel for their support, guidance, effort and patience throughout the thesis period. It was a pleasure to work with them.

I would also thank to the committee members Prof. Dr. Sinan Gürel and Assoc. Prof. Dr. Ayşegül Altın Kayhan for their feedbacks in every 6-month which increased the quality of the work. Along with them, the committee members Assoc. Prof. Dr. Cem İyigün and Assist. Prof. Dr. Mustafa Gökçe Baydoğan made invaluable comments.

Being a teaching and research assistant in METU-IE department during this period made many things possible. I had the chance to collaborate with valuable professors and colleagues, to teach amazing students and to make some scientific contributions. I am proud of being a member of this department both as a student and an assistant.

It is also pleasure to acknowledge all my friends for not leaving me alone and easing my work with their great ideas and supports. Specially, I would like to thank Derya İpek Eroğlu for being a great fellow in every way. She kindly supported me by making our long working hours enjoyable and taking care of my concerns. I would also thank to Gamze Erel for being there whenever I need her. Along with them, I am thankful for Haluk Damgacıoğlu, Yasemin Limon and Elif Akça since they made me feel like we were always side by side despite long distance.

Last but not the least, I am grateful for my family because of their endless support, love and understanding.

This study was supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under grant 2211.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vii

ACKNOWLEDGMENTS x

TABLE OF CONTENTS xi

LIST OF TABLES xv

LIST OF FIGURES xxi

CHAPTERS

1 INTRODUCTION 1

- 1.1 Semi-supervised Clustering for Regional Data Objects 3
- 1.2 Data Mining for Central Nodes in a Graph-structured Data Object . . 5
- 1.3 Data Mining for Finding Centroid Trees in the Clustering of Tree-structured Data Objects 7

2 SEMI-SUPERVISED CLUSTERING FOR REGIONAL DATA OBJECTS . 11

- 2.1 Introduction 11
- 2.2 Literature Review 13
  - 2.2.1 Unsupervised Clustering 13
  - 2.2.2 Semi-supervised Clustering 16
    - 2.2.2.1 Semi-supervised Clustering with Labeled Data 19

2.2.2.2 Semi-supervised Clustering with Instance-Level Constraints 19

2.2.2.3 Semi-supervised Clustering with Cluster-Level Constraints 20

2.2.2.4 Feasibility Issues for the Clustering with Constraints . . 21

2.2.3 Clustering of Uncertain Data 22

2.3 Notation 24

2.4 Computation of the Centroid of a Given Cluster 25

2.4.1 An SOCP Formulation for the CCP 26

2.4.2 Subgradient Algorithm for the CCP 28

2.5 Semi-supervised Clustering with Reginal Data 32

2.5.1 An MISOCP Formulation for the SSC 32

2.5.2 K-means based Algorithms for the SSC 35

2.5.2.1 UCOP-k-means 35

2.5.2.2 UPC-k-means 36

2.5.2.3 UCVQE 39

2.5.2.4 USeeded-k-means and UConstrained-k-means 41

2.5.3 Agglomerative Hiearchical Clustering based Algorithm for the SSC 44

2.5.4 More on Literature of Semi-supervised Clustering 46

2.6 Computational Studies 48

2.6.1 Performance Measures 52

2.6.2 Computational Results 54

2.6.2.1 Artificial Datasets 54

2.6.2.2 Real Life Datasets 67

2.7 Conclusion and Future Work 80

3 DATA MINING FOR CENTRAL NODES IN A GRAPH-STRUCTURED DATA OBJECT 83

3.1 Introduction 83

3.2 Notation 85

3.3 Betweenness and Group Betweenness Centrality 86

3.3.1 Bounds on the Group Betweenness Centrality 89

3.4 Computation of the New Bounds 90

3.4.1 Preprocessing Step 92

3.4.2 Upper Bounds 92

3.4.3 Lower Bounds 95

3.4.4 Illustrative Example 97

3.4.5 An Algorithm for the Group with the Highest GBC Value . . . 99

3.5 Computational Experiments 101

3.5.1 Selection of Bounds 101

3.5.2 Randomly Generated Graphs 109

3.5.3 Real-life Networks 113

3.6 Conclusion and Future Work 118

4 DATA MINING FOR FINDING CENTROID TREES IN THE CLUSTERING OF TREE-STRUCTURED DATA OBJECTS 121

4.1 Introduction 121

4.2 Literature Review 122

4.3 Notation and Problem Description 129

4.4 Finding Representative Tree of a Given Population of Trees 132

4.4.1 Unweighted Vertex/Edge Overlap (UWVEO) 132

4.4.2 Unweighted Graph Edit Distance (UWGED) 135

4.4.3 Weighted Vertex/edge Overlap (WVEO) 137

4.4.4 Weighted Graph Edit Distance (WGED) 138

4.5 Clustering of Tree-Structured Data 140

4.6 Computational Studies 142

4.6.1 Parameter Selection 142

4.6.1.1 Performance Measures 146

4.6.1.2 Results for tree-kmeans-UWVEO 147

4.6.1.3 Results for tree-kmeans-UWGED 150

4.6.1.4 Results for tree-kmeans-WVEO 154

4.6.1.5 Results for tree-kmeans-WGED 158

4.6.2 Comparison of tree-kmeans-UWVEO, tree-kmeans-UWGED and kmodes 163

4.6.3 Comparison of tree-kmeans-WVEO, tree-kmeans-WGED and kmeans 170

4.6.4 Performance on real life datasets 174

4.7 Conclusion and Future Work 180

5 CONCLUSION 183

REFERENCES 189

CURRICULUM VITAE 205

LIST OF TABLES

TABLES

Table 2.1 Complexity of the clustering feasibility problem 22

Table 2.2 Comparison of SM and SOCP in terms of computational time (in seconds) 32

Table 2.3 Parameters for rectangular region genaration 55

Table 2.4 Solution time (in seconds) for small size instances with actively generated constraints 57

Table 2.5 Performance measures (percent deviation of global and local objective function value from the best global and local value found (*ObjG* and *ObjL*), rand index (*Rand*), pairwise F-measure (*FM*), and mutual information (*MI*)) for $P_{20,2,10}$ where constraints are generated actively 58

Table 2.6 Performance measures (percent deviation of global and local objective function value from the best global and local value found in the corresponding row (*ObjG* and *ObjL*)) for the problem instance with 100 rectangular data objects and 5 centroids 64

Table 2.7 Performance measures (percent deviation of global and local objective function value from the best global and local value found for the problem instance (*ObjG'* and *ObjL'*)) for the problem instance with 100 rectangular data objects and 5 centroids 65

Table 2.8 Performance measures (rand index (*Rand*), F-measure (*FM*), and mutual information (*MI*)) for the problem instance with 100 rectangular data objects and 5 centroids 66

Table 2.9 Fat and Oil dataset 67

Table 2.10 Performance measures (rand index (*Rand*), F-measure (*FM*), and mutual information (*MI*)) for the Fat and Oil dataset 68

Table 2.11 Interval valued Iris dataset 71

Table 2.12 Performance measures (rand index (*Rand*), F-measure (*FM*), and mutual information (*MI*)) for interval valued Iris dataset 72

Table 2.13 Performance measures (rand index (*Rand*), F-measure (*FM*), and mutual information (*MI*)) for interval valued Wine dataset 76

Table 2.14 P-values for the effect of algorithm on different performance measures for each constraint generation and initialization combination 77

Table 2.15 Best algorithm(s) for different performance measures for each constraint generation and initialization combinations 77

Table 2.16 P-values for the effects of constraint generation, initialization and the number of constraints on different performance measures of algorithms 78

Table 2.17 Performance measures (rand index (*Rand*), F-measure (*FM*), and mutual information (*MI*)) of UConstrained k-means for the interval valued Gamma dataset 79

Table 3.1 The performances of different bounds on the Zachary, and randomly generated preferential attachment (PA) and Erdös-Renyi (ER) graphs in terms of the percentage of finding the best bounds for different group sizes 104

Table 3.2 The total computational time of different bounds on the Zachary, and randomly generated preferential attachment (PA) and Erdös-Renyi (ER) graphs for all of the groups with different group sizes 105

Table 3.3 The number of possible groups; the number of remaining groups after elimination by YAT-HW and KCBl-KCBu bounds; the total time spent for YAT-HW and KCBl-KCBu bound calculations; and the total time spent for KCBl-KCBu bound calculations with the original method in [92] on the Zachary, and randomly generated preferential attachment (PA) and Erdös-Renyi (ER) graphs for different group sizes 106

Table 3.4 The mean and median gaps for YAT-HW and KCBl-KCBu bound pairs; and the mean gaps between the selected bounds (YAT and HW) and the exact values on the Zachary, and randomly generated preferential attachment (PA) and Erdös-Renyi (ER) graphs for all of the groups with different group sizes 107

Table 3.5 The number of remaining groups after elimination by YAT-HW bound and the total time spent for YAT-HW bound calculations on randomly generated different Erdös-Renyi and preferential attachment graphs for all groups with different k values 110

Table 3.6 The mean and median gaps for YAT-HW bound pair on randomly generated different Erdös-Renyi and preferential attachment graphs for all of the groups with different group sizes k 111

Table 3.7 The estimated and exact GBC values, the gap and the optimality gap for the best group returned by Algorithm 10 for randomly generated different Erdös-Renyi and preferential attachment graphs for different group sizes k 112

Table 3.8 The mean and median gaps for YAT-HW bound pair on randomly generated different Erdös-Renyi and preferential attachment graphs for randomly selected 10000 groups with different large group sizes k 112

Table 3.9 The number of all possible groups formed with the 20 nodes with the highest betweenness values; the number of remaining groups after elimination by YAT-HW bound pair; the total time spent for YAT-HW bound calculations; mean and median gaps for YAT-HW bound pair; the estimated and exact GBC values together with the optimality gaps for the best group returned by Algorithm 10; and the exact GBC value of the group with the highest GBC value on the Facebook network for different group sizes k 113

Table 3.10 The number of remaining groups among randomly selected 1000 groups after elimination by YAT-HW bound pair; mean and median gaps for YAT-HW bound pair; the estimated and exact GBC values for the best group returned by Algorithm 10; and the total time spent for YAT-HW bound calculations and exact GBC calculations on HepTh network for different group sizes k 116

Table 3.11 The number of remaining groups among randomly selected 1000 groups after elimination by YAT-HW bound pair; mean and median gaps for YAT-HW bound pair; the estimated and exact GBC values for the best group returned by Algorithm 10; and the total time spent for YAT-HW bound calculations and exact GBC calculations on HepPh network for different group sizes k 117

Table 4.1 The properties of randomly generated problem instances in SmallDataUW 144

Table 4.2 The properties of randomly generated problem instances in LargeDataUW 144

Table 4.3 The properties of randomly generated problem instances in SmallDataW 145

Table 4.4 The properties of randomly generated problem instances in LargeDataW 145

Table 4.5 The performance of tree-kmeans-UWVEO on SmallDataUW 149

Table 4.6 The performance of tree-kmeans-UWVEO on LargeDataUW 149

Table 4.7 The performance of tree-kmeans-UWGED on SmallDataUW 151

Table 4.8 The performance of tree-kmeans-UWGED on LargeDataUW 152

Table 4.9 Wilcoxon signed rank test in terms of AvPD on SmallDataUW for tree-kmeans-UWGED 153

Table 4.10 Nondominated parametric settings for tree-kmeans-UWGED (1 means nondominated; 0 means dominated) 153

Table 4.11 The performance of tree-kmeans-WVEO on SmallDataW 156

Table 4.12 The performance of tree-kmeans-WVEO on LargeDataW 157

Table 4.13 Nondominated parametric settings for tree-kmeans-WVEO (1 means nondominated; 0 means dominated) 158

Table 4.14 The performance of tree-kmeans-WGED on SmallDataW 161

Table 4.15 The performance of tree-kmeans-WGED on LargeDataW 162

Table 4.16 Nondominated parametric settings for tree-kmeans-WGED (1 means nondominated; 0 means dominated) 163

Table 4.17 Comparison of tree-kmeans-UWVEO, tree-kmeans-UWGED and kmodes with respect to BPD of UWVEO objective 166

Table 4.18 Comparison of tree-kmeans-UWGED, tree-kmeans-UWVEO and kmodes with respect to BPD of UWGED objective 167

Table 4.19 Comparison of tree-kmeans-UWVEO, tree-kmeans-UWGED and kmodes with respect to external performance measures 169

Table 4.20 Comparison of tree-kmeans-WVEO, tree-kmeans-WGED and k-means with respect to BPD of WVEO objective 170

Table 4.21 Comparison of tree-kmeans-WGED, tree-kmeans-WVEO and kmeans with respect to BPD of WGED objective 171

Table 4.22 Comparison of tree-kmeans-WVEO, tree-kmeans-WGED and kmeans with respect to external performance measures 172

Table 4.23 Outlier case 173

Table 4.24 Results for clustering whole dataset with different data representations 177

Table 4.25 p-values obtained by Kruskal-Wallis test for the equality of age medians 178

Table 4.26 p-values obtained by Wilcoxon rank sum test for the equality of pairwise age medians 179

LIST OF FIGURES

FIGURES

Figure 2.1 An example problem instance with $n = 50$ and $k = 3$, and its solution 56

Figure 2.2 Performance measures (global objective function value, rand index, pairwise F-measure and mutual information) for the problem instance with 200 rectangular data objects and 5 centroids where constraints are generated actively 59

Figure 2.3 The best partition found by the algorithms (also the desired partition) for the Iris dataset 73

Figure 2.4 Partitions for the Wine dataset 78

Figure 2.5 Partitions for Gamma dataset 80

Figure 3.1 A graph with $n = 5$ 96

Figure 3.2 Complete graph with vertex set $S = \{2, 3, 4, 5\}$ and edge weights $P(i, j)$ 98

Figure 3.3 Zachary's karate club network 102

Figure 3.4 Randomly generated preferential attachment network with $n =$ 30 and $e = 2$, PA 102

Figure 3.5 Randomly generated Erdös-Renyi graph with $n = 30$ and $p =$ 0.2, ER 103

Figure 3.6 Distribution of gaps for YAT-HW and KCBl-KCBu bound pairs for different group sizes on the Zachary's network 108

Figure 3.7 Average time to calculate lower bound, lower bound and upper bound, and exact value of GBC of a single group on HepTh network for different group sizes 116

Figure 3.8 Average time to calculate lower bound, lower bound and upper bound, and exact value of GBC of a single group on HepPh network for different group sizes 118

Figure 4.1 An example of a population $P = \{T_1, T_2, T_3\}$ of 3-ary trees, its generator G, and support tree ST 130

Figure 4.2 Small dataset with 4 3-ary trees 141

Figure 4.3 Conditional frequencies of edges appearing in the dataset in Figure 4.2 141

Figure 4.4 Information of patients in brain artery data 174

Figure 4.5 Binary trees for patient 2 with child and radius correspondence, respectviely 175

CHAPTER 1

INTRODUCTION

Data mining, the science of extracting new and useful information from data, has gained a lot of attention recently among scientists and in society as a whole due to the wide availability of huge datasets and the increased power of computers. According to [110], the amount of data produced worldwide during the year 2002 and stored on paper, film, magnetic media, and optical devises was estimated to be between 3,400,000 and 5,600,000 terabytes. These numbers are about twice the size of the data produced during the year 1999 [109]. The amount of data that has exploded year over year introduced new challenges and led to continued innovation in data storage and data mining techniques.

A fundamental categorization in data mining techniques is:

1. Supervised learning,
2. Unsupervised learning.

In supervised learning, the dataset is split into training data and test data. Observations in the training data are provided with desired or known output values (labels) which are used to construct a function that predicts the output values of the observations in the test data. Classification and regression are good examples of supervised learning problems. Typically, supervised learning algorithms require a large training dataset, which is not always available or may be very costly to acquire. In unsupervised learning, however, there is no learning from known cases (no training data), but instead one tries to find intrinsic "natural" structures in "unlabeled" data. As unsupervised learning methods are completely unguided, the structure extracted from the data may not always be relevant to the analyst or user. If there exists some prior knowledge, the analyst can apply semi-supervised learning methods which incorporate some prior knowledge into the learning process to extract the desired structure from the data. Clustering is one of the most widely used unsupervised data mining techniques.

In this study, three research problems are addressed. The first problem is a semi-supervised clustering problem where the locations of the data objects are subject to uncertainty, i.e., instead of crisp points in the feature space, they are defined as regions. The second one is the problem of finding a group of central nodes on a given graph considering the group betweenness centrality (GBC) measure. The third one is the problem of clustering a given population of graph structured data objects (m-ary trees) into groups.

Before the explanation of the problems in more detail, it is beneficial to give some general information about clustering as it constitutes two thirds of the thesis. Clustering is used to partition unlabeled data objects into groups (clusters) so that the objects in a group will be similar to each other with respect to some similarity criteria (i.e., high intra-cluster similarity) and different from the objects in other groups (i.e., high inter-cluster dissimilarity). In some clustering problems, the number of clusters to be formed, k, is given as a parameter. In some others, however, k may be unknown. Clustering has been employed in many disciplines such as statistics, biology, marketing and medicine. It has mainly been used for three purposes:

- to discover the underlying structure of the data,
- to find natural classification of data objects by identifying similarities among them, and
- to organize and summarize the data [83].

In some cases, clustering can be used as a preprocessing tool for other tasks such as regression, principal component analysis, and association analysis. See for instance [48] in which the authors used clustering techniques to reduce complexity in solving automated planning problems.

In the following subsections, most related literature, and our motivations and contributions are summarized for each research problem of the thesis.

1.1 Semi-supervised Clustering for Regional Data Objects

Most of the clustering studies in the literature are devoted to the analysis of fixed (point) data in the feature space. However, in some recent cases data objects may be regions or graphs instead of points. Such data objects arise because of improving technology/measurement capabilities and the need for deeper analyses which lead to collect more complex datasets. For example, in [18], authors discuss the need for interval valued version of one of the most famous classification datasets, namely the Iris dataset. In the original version of the dataset, four features of 150 irises are provided. In other words, there are 150 points in $\mathbf{R}^4$. However, authors aggregate individual observations that are collected under similar conditions and obtain 30 interval valued observations, i.e., 30 hyper-rectangles in $\mathbf{R}^4$.

In addition to interval valued observations, there may be uncertainty in the location of the data objects due to several reasons such as imprecision in measurement, reporting error, randomness, and so on. In such cases, using a region to represent a data object is more appropriate than using a point. For instance, consider forests in a certain region that are to be clustered to build a given number of fire stations. The initial site of a possible fire in a forest is not known in advance. In this case, each forest can be thought of as a region and the location of a possible fire can be represented by a probability distribution over the forest. Other examples are the facility location or coverage problems where the demand may come from geographical regions instead of individual customers.

The first problem considered in this thesis is the semi-supervised clustering of data objects, whose locations are subject to uncertainty and represented as regions. Each uncertainty set is assumed to be either a polytope or a closed disk. The distance between the centroid of a cluster and an uncertain/regional data object is measured as the distance between the centroid and the farthest point of the data object. Hence one of the aims in our clustering problem is to minimize the weighted sum of squared maximum Euclidean distances between the data objects and the centroids of their respective clusters

Semi-supervised learning is at the intersection of supervised and unsupervised learning and able to eliminate disadvantages of both learning types. Since it can be used when there is only small amount of prior information about the data, the cost and time to obtain large amounts of training data for supervised learning is avoided. Also, as we have some amount of information to use in the learning process, the process is not completely unguided as in the unsupervised learning which may lead to extracting irrelevant information. Moreover, studies show that even with a small amount of prior information the clustering performance can be improved and computing time of the learning process can be decreased significantly [144]. Thus, in recent decades, semi-supervised clustering has attracted the attention of researchers and practitioners in several disciplines.

Prior information used in semi-supervised clustering may have different sources and forms. This information may arise from expert opinion, user feedback, or needs of the problem owner expressed as a few labeled data objects or constraints. The user may specify some instance-level constraints indicating that the data objects should be in the same cluster (must-link constraints) or in different clusters (cannot-link constraints). Alternatively, she may define some cluster-level constraints limiting the size or radii of the clusters. If prior information is allowed to be violated in the final clustering it is called soft information, otherwise it is hard information.

In our problem, to cluster the regional data objects into given number of clusters, we assume that there are some soft instance level must-link and cannot-link constraints. Hence, the overall aim is to minimize the sum of two terms: (1) total violation cost of the unsatisfied instance level constraints and (2) weighted sum of squared Euclidean distances between the farthest points of the data objects and the centroids of the clusters they are assigned to.

Studies that are most relevant to our problem are [10, 11, 43, 116, 145] which deal with semi-supervised clustering of point data objects. It should be noted that, to the best of our knowledge, all of semi-supervised clustering literature deals with point data objects. In [145], authors propose an extension of k-means for semi-supervised clustering of point data objects when there are hard instance-level constraints. In the existence of soft instance-level constraints, [11] and [43] provide k-means based algorithms. When there is prior information in the form of labeled data, in [10], authors propose two extensions of k-means, one for hard information case and one for soft information case. In [116], an agglomerative hierarchical clustering algorithm is provided in the presence of hard must-link constraints and soft cannot-link constraints.

The main contribution of our study to the literature of semi-supervised clustering is to address the regional data objects. Specifically, first the centroid computation problem is investigated for a cluster of regional data objects. For this purpose, a second order cone programming formulation which is an extension of a previous formulation in [54] is developed, and a novel subgradient method is proposed for its solution. While the formulation in [54] considers polygons in $\mathbf{R}^2$., our formulation here handles polytopes and disks in $\mathbf{R}^d$. Next, for the considered semi-supervised clustering problem, a novel mixed-integer second order cone programming formulation is proposed and six different semi-supervised clustering algorithms from the literature (five of them are k-means based and one of them is agglomerative hierarchical clustering based) are modified to make them applicable for the case of regional data objects. Finally, the solution approaches are compared on artificial and real-life datasets.

1.2 Data Mining for Central Nodes in a Graph-structured Data Object

The second problem of the thesis considers data mining for a single graph-structured data object. It aims to find a given number of central nodes in terms of group betweenness centrality (GBC) measure on an unweighted, undirected and connected graph.

Central node identification problems are essential mainly in network analysis (social networks, collaboration networks, computer networks, and so on). The answer of the question that "Which group of nodes is the most central in a graph?" is purpose and context dependent. The answer changes with the centrality measure in use. Degree centrality, closeness centrality, and betweenness centrality are some frequently used measures in the literature. Each one has different application areas since each has different assumptions [114]. Degree centrality of a node is equal to the degree of the node. This measure assumes that the larger the number of connections of the node, the higher its importance. For example, consider that we have a social network where nodes represent individuals and edges represent their friendship. If one wants to find the most popular individual in the network and defines the popularity as the number of friends, degree centrality is the most appropriate measure in this case. On the other hand, if we have a network where nodes represent residental areas and edges represent roads between them, and want to locate a hospital on the network, closeness centrality will be the most appropriate measure. Because closeness centrality of a node is inversely proportional to the total distance of that node to the other nodes in the network. In other words, this measure assumes that the closer the node to all other nodes, the higher its importance.

Betweenness centrality which is the one used in this thesis assumes that information flows through shortest paths. This has several applications. For instance, for an investor who wishes to locate a roadhouse on a road network where edges represent roads and nodes represent intersection points of roads, the betweenness centrality is the most appropriate measure. Because the drivers, i.e., flows on the network, have a tendecy to use shortest paths between their origins and destinations. Since the investor needs to increase the number of people who are driving by the roadhouse as each one is a potential customer, his problem is actually the problem of finding the node with the highest betweenness centrality on the network. Another application can be found in computer networks where nodes represent individual users and edges represent information flows between users. If one needs to locate a server on the network, she should use betweenness centrality to improve the performance since information flows through shortest paths.

GBC is an extension of betweenness centrality quantifying the centrality of a group of nodes instead of a single node. The GBC of a group of nodes measures the influence the group has on communications between every pair of nodes in the network under the shortest path assumption. Finding the most central group of nodes (in terms of GBC) with given size, say k, is a combinatorial problem. To calculate the GBC of a group, in [125], authors propose an exact algorithm requiring $O(k^3)$ time after a preprocessing step taking $O(n^3)$ time where n is the number of nodes in the network. Also, a bounding procedure is provided in [92]. To find upper and lower bounds for the GBC of a group, this procedure requires $O(n^3 + k^3)$ time.

For the problem of finding a group of nodes of a given size that has the highest GBC value, we propose a new method that first computes upper and lower bounds for the GBC of different groups and then eliminates groups with upper bounds that are lower than the maximum lower bound obtained to find candidates for the optimal group. Our method combines good properties of the studies by [125] and [92]. It uses the preprocessing step of [125] which takes $O(n^3)$ time. After the preprocessing step, our algorithm uses the bounding scheme in [92] with stronger and more time-efficient upper and lower bounds found in the probability theory literature to bound GBC values of several groups successively. For each group, time to find bounds is $O(k^2)$. The proposed method is tested on both randomly generated and real-life networks. Experiments show that just by using the bounds, only a small fraction of groups remains as candidates for the optimal group. Thus, with our method, the problem of finding the most central group of nodes can be solved in a more time-efficient manner. Assume that we have s groups among which we need to find the most central group. Time of the algorithm in [125] is $O(n^3 + sk^3)$ while time of our method is $O(n^3 + sk^2 + lk^3)$ where l is the number of remaining groups after the elimination based on the bounds and $l \ll s$.

1.3 Data Mining for Finding Centroid Trees in the Clustering of Tree-structured Data Objects

The last problem of the thesis assumes that data objects to be clustered are trees. As it is stated before, in recent years, more complex datasets consisting of data objects which are not points are available. Tree-structured data objects arise in many domains, however, clustering of them is a challenging problem and is not studied much in the literature. In [5], the authors represent each patient's brain artery structure with a tree. Using MRA image of a patient's brain, they obtain a binary tree where nodes represent split-up points of vessels, edges represent vessels between split-up points, and node correspondence is known. Then, they show that age and sex of patients are related with brain artery structures. Clustering of brain artery structures can reveal further effects of age and sex on these structures.

In this study we investigate clustering of m-ary rooted trees where node correspondence is known, nodes are unweighted, and edges can be either weighted or unweighted. To measure the similarity between a tree-structured data object and a cluster centroid which is also a tree, we utilize the vertex/edge overlap measure. To find the distance between them, we use the graph edit distance measure. We develop four mathematical programming formulations to find the optimal centroid trees of clusters and propose solution procedures, when these two measures are used for the case of weighted or unweighted edges. The aim is to maximize the sum of vertex/edge overlaps or to minimize the sum of graph edit distances between the data objects and the centroids they are assigned to.

In [65], the author considers three clustering problems one of which is the same as our problem. To cluster m-ary trees with known node correspondence, she proposes an extension of the k-means algorithm which uses Hamming distance as the distance measure. The graph edit distance measure we use is equivalent to Hamming distance when edges are unweighted, but we also consider the weighted version of this problem. In addition, we consider the additional similarity measure of vertex/edge overlap for both the unweighted and weighted versions. Thus, we compare our three novel approaches with an existing solution approach.

In [108, 107], clustering of retinal vessel structures is considered. Each patient's retinal vessel structure is represented by an m-ary tree where nodes show the split-up points of retinal vessels, edges show the vessels between nodes, and node correspondence is known. To cluster those trees, authors use k-means algorithms with different distance measures after transforming each tree to a point in a low dimensional space. The performances of the algorithms to separate normal and retinopathy patients are not good even in the presence of visually separable clusters. Unlike this studies which transform trees into points in Euclidean space, we work on the original tree space. To the best of our knowledge, there are not any other studies on direct clustering of tree-structured data objects with known node correspondence.

As it is stated before, most of the clustering literature is devoted to point data objects. The main contribution of our study is to cluster tree-structured data objects in their original space and to propose three novel solution approaches for this purpose.

To find the centroid tree of a given cluster of trees, we provide mathematical programming formulations for each of four different measures, namely unweighted and weighted vertex/edge overlap, and unweighted and weighted graph edit distance. We solve these formulations (except the one for the weighted vertex/edge overlap) to optimality. Then, for the clustering of tree-structured data objects, we propose k-means based algorithms for each of the four different measures. Finally, we compare our approaches with the traditional k-modes and k-means algorithms on both randomly generated and real-life datasets.

The outline of this thesis is as follows. In Chapter 2, we provide details of our work on semi-supervised clustering of regional data. In Chapter 3, the focus is on the identification of the central nodes on a single graph. In Chapter 4, we describe our work on clustering of tree-structured data objects.

CHAPTER 2

SEMI-SUPERVISED CLUSTERING FOR REGIONAL DATA OBJECTS

2.1 Introduction

Traditional clustering algorithms use only (fixed) data points as input data objects. However, there can be uncertainty in the location of the data objects. In such cases, as taking the location of the data objects as fixed points will be a naive assumption, the data objects can be considered as multi-dimensional regions in the space. Clustering of data objects whose locations are uncertain appears in several real life settings, see e.g. [84].

As clustering algorithms are completely unguided, the structures obtained are not always relevant or useful to the user. In some cases, the data analyst may have a priori (domain) knowledge about the underlying structure of the data. With completely unlabeled data or a few number of labeled data, the supervised learning techniques like classification may not be suitable. One way of learning from such data would be to completely ignore the prior knowledge and apply unsupervised learning methods. As it is noted, this may however result in extracting irrelevant structure from the data. Another way of learning from such data is by means of semi-supervised learning which incorporates the prior knowledge to the learning process to improve the quality of the result. In the last two decades, semi-supervised clustering which is also known as constrained clustering or clustering with side information has attracted several researchers as it has been observed that even with a small amount of prior knowledge, the clustering performance can be improved and running time of the process can be decreased significantly [144].

In this study, we consider a semi-supervised clustering problem in the presence of instance-level constraints where the locations of the data objects are subject to uncertainty. Uncertainty in the location of the data objects may arise due to several reasons. Thus, the clustering of uncertain data is an important issue which should be addressed.

The distance between an uncertain data object and the representative centroid is subject to change for each realization of the location of the data object. Our aim is to minimize the sum-of squares objective function in the worst-case. In other words, for any realization of the locations of the data objects, the evaluated objective function should not be greater than the optimal objective function value. For this reason, the distance between the centroid of a cluster and a regional data object is measured in terms of the maximum distance between them. This objective function is also meaningful in the sense that any realization of a data object is not expected to be very far away from the representative cluster centroid as the worst-case scenario is considered.

Each uncertainty set is assumed to be a polytope or disk. Given a (bounded) regional data object, we can take its convex hull as the maximum distanced point from a given centroid does not change. We can then approximate the convexified object in any accuracy with a polytope. So, the solution techniques provided in this chapter are more generally applicable to any bounded, not necessarily convex, regional data objects.

A related problem is the problem of minimizing the maximum of the maximum distances between the regional data objects and the cluster centers in the presence of instance-level constraints. This problem is a generalization of the well studied clustering/facility location problem that minimizes the maximum radius among the clusters which is known as the k-center problem (k represents the number of clusters to be formed), see e.g., [31]. The problem will be considered in this study, on the other hand, reduces to the minimum-sum-of-squares (semi-supervised) clustering problem when all the regional data objects are points.

Given a cluster consisting of some regional data, the problem of how to compute its centroid is a significant problem since it appears as a subproblem in many of the clustering algorithms proposed. Therefore its solution is of great importance and we consider this problem first. We provide mathematical formulation and application of subgradient method for the problem.

After the single cluster case, we provide mathematical programming formulation and some heuristic approaches to find a partition of regional data objects which is expected to be in accordance with a given number of instance-level constraints. The objective function of the problem is to minimize the total of the sum of the violation costs of the unsatisfied instance level constraints and a weighted sum of squared maximum Euclidean distances between the locations of the data objects and the centroids of the clusters they are assigned to.

Finally, we compare all of the proposed solution methods in terms of solution quality and computational time on both randomly generated and real life datasets.

The rest of the chapter is organized as follows: In Section 2.2, we provide a brief literature review on unsupervised clustering, semi-supervised clustering, and clustering in the presence of uncertainty. In Section 2.3, we introduce the notation used throughout the chapter. In Section 2.4, we discuss solution approaches that compute the centroid of a given cluster consisting of regional data objects. In Section 2.5, we provide mathematical programming formulation and heuristic approaches, which are modifications of the algorithms from the literature, for the considered clustering problem. The computational studies are presented in Section 3.5 and we conclude in Section 2.7 with some future research directions.

2.2 Literature Review

2.2.1 Unsupervised Clustering

A common approach in clustering is to consider an optimization problem and solve the resulting problem using exact algorithms, heuristics, or approximation algorithms. Different objective functions have been employed in the literature. The most commonly used one is to minimize the sum-of-squared distances between each data object and the centroid of the cluster the object belongs to. Here, the centroid of a cluster can be considered as the representative of the cluster.

Most clustering algorithms can be categorized as hierarchical or partitional. Hierarchical clustering algorithms build a hierarchy of clusters by merging (agglomerative methods) or splitting (divisive methods) clusters successively [59]. The hierarchy of clusters is usually represented by a tree known as dendrogram. By cutting the dendrogram at the proper level, clustering with the desired number of clusters can be obtained. For more on hierarchical clustering algorithms, [112, 118, 155] can be reviewed.

Agglomerative hierarchical clustering algorithms begin with each data object in a separate cluster and in a progressive manner merge two clusters that are the closest to reduce the number of clusters by one until all the data objects are in a single cluster. The steps of a basic agglomerative hierarchical clustering algorithm are as follows:

Algorithm 1 Agglomerative Hierarchical Clustering

```
Let $C_i = \{x_i\}$, for every $i = 1, 2, \ldots, n$ and let $C = \{C_1, C_2, \ldots, C_n\}$
for $k = n$ to $1$ do
    $Dendrogram(k) = C$
    Let $(u, v) = \text{argmin}_{i \neq j:\ C_i, C_j \in C}\ \rho\left(C_i, C_j\right)$
    $C_u = C_u \bigcup C_v$ and $C = C \setminus C_v$
end for
```

where $X = \{x_1, ..., x_n\}$ indicates the set of point data objects to be clustered, n is the number of data objects and $\rho\left(C_i, C_j\right)$ represents the distance between the clusters C_i and C_j and (u, v) the arguments of the closest two clusters. Distance between two clusters can be measured in several different ways. In single linkage clustering, the distance between two clusters is measured by the minimum distance between the data objects in the two clusters. In complete linkage clustering, distance between clusters is measured by the two most distant objects rather than the closest ones. Average linkage clustering uses on the other hand the average distance between the objects in the two clusters. There are several other ways to measure distances between groups, see for instance [155].

Partitional clustering algorithms, on the other hand, attempts to construct a one-level clustering of the data objects without a hierarchy. The problem of clustering a given number of point data objects so that the sum-of-squares objective function is minimized is NP-hard in general [115] and thereof heuristic solution approaches have been widely used. The k-means algorithm [111], which is among the most popular data mining algorithms, and its variants are the most commonly used partitional clustering heuristics proposed for the problem. Note that because of its easy to implement nature, simplicity, efficiency, and empirical success, its framework has also been commonly used in algorithms developed for semi-supervised clustering problems. The steps of k-means algorithm are based on two simple observations. First, if the centers of the clusters are known, then each data object is assigned to the cluster whose center is closest to the object. Second, if all the data objects belonging to a cluster are known, then the center of the cluster is computed by averaging all data objects in the cluster. Algorithm 2 shows the details of the k-means algorithm where k represents the number of clusters to be formed. Algorithm starts with a set of cluster centroids which can, for instance, be selected at random. Then, the assignment and update steps are repeated until convergence is achieved. In the assignment step, each data object is assigned to the closest cluster (in terms of the distances between the object and the centroids). In the update step, the coordinates of each centroid are updated as the average of the coordinates of the data objects belonging to the cluster.

Algorithm 2 k-means

```
Initialization: Start with initial cluster centroids, $\mu_j$, $j \in \{1, \dots, k\}$.
repeat
    Let $C_j = \emptyset$, for all $j \in \{1, \dots, k\}$.
    Assignment: Assign each data object $x_i \in X$, $i \in \{1, \dots, n\}$ to the closest cluster $j^*$, and let $x_i \in C_{j^*}$, where $j^* = \text{argmin}_j \|x_i - \mu_j\|$.
    Update: Update the cluster centroids by averaging the data objects assigned to them, i.e.,
        $\mu_j = \frac{\sum_{x_i \in C_j} x_i}{|C_j|}, i \in \{1, \dots, n\}, j \in \{1, \dots, k\}.$
until Convergence is achieved.
return Partition $\{C_1, \dots, C_k\}$ of $X$.
```

The k-means algorithm converges to a local solution, but the convergence can be slow. The final clustering is highly dependent on the set of initial cluster centroids and therefore the algorithm may be run several times with different set of initial centroids with the hope of getting better solutions. Several initialization heuristics have been proposed in the literature to be able to start with a good set of initial centroids, see, e.g., [34]. Due to the known problems with the classical k-means algorithm, several variants of the algorithm have been developed [74, 77, 104].

Note that the review in this section is by no means complete since the topic is not directly relevant to the problem we consider. We just reviewed the important definitions and algorithms which are used in the rest of the study.

2.2.2 Semi-supervised Clustering

In some cases, there may be some priori knowledge about the underlying structure of the data. This knowledge can arise from expert opinion, user feedback, or the needs of the problem owner. Incorporating such knowledge into the clustering process and hence allowing the user to guide the process toward the desired partitioning of the data or help the clustering algorithm avoid local minima is known as semi-supervised clustering and has received extensive attention recently.

Semi-supervised clustering techniques have been successfully applied in several fields. In gene clustering based on gene expression data obtained with DNA microarrays, databases of co-occurrence data have been used to generate constraints forcing that certain genes must be in the same cluster [60, 129, 153].

In some agricultural areas, each farmer cultivates a large number of small and dispersed land parcels. This has several disadvantages. A solution to this problem would be land consolidation which refers to the process that the farmers surrender their dispersed parcels in order to receive a more continuous equivalent land area. Land consolidation is clearly a clustering problem with several constraints. Firstly, neighboring parcels should be assigned to a farmer. Secondly, the total land area of a farmer should not change much after consolidation. Thirdly, the quality of soil of each farmer's land should also not change by too much. There are other constraints about the geometry of the land a farmer receives, such as it should not be a continuous long but narrow land [21].

In text clustering, the goal is to automatically categorize a large number of text documents into smaller and manageable groups (clusters) based on their content. The user may specify that some documents should be clustered into the same group as they have similar contents or have the same authors and/or some documents should be separated from each other due to the differences in their subjects [79].

For more applications of semi-supervised clustering, the survey written by Davidson and Basu [42] can be investigated.

In semi-supervised clustering, the prior knowledge can appear in the form of labeled data or constraints. In the former one, labels of some of the data objects are known, but the amount of available information (number of known labels) may not sufficient to perform classification. In such cases, one can perform clustering instead of classification incorporating the known labels in some way to the clustering procedure.

Instead of having labeled data, there may be some constraints on the data objects or clusters. For example, there may be constraints on pairs of data objects in the form of pairwise relations. This type of knowledge is generally more practical. Getting the true labels of the data objects may require too much effort or may be costly, while whether pairs of data objects belong to the same cluster or different clusters can be easily specified by an expert.

Constraints encountered in clustering problems can be categorized as instance-level and cluster-level constraints. Instance-level constraints are pairwise must-link and cannot-link constraints on some pairs of data objects. A must-link constraint between two data objects is used to indicate that the objects are to be placed in the same cluster. A cannot-link constraint between two data objects, on the other hand, is used to indicate that the objects are to be placed in different clusters. It should be noted that prior information in the form of pairwise constraints is weaker than the prior information in the form of labeled data [91]. While labeled data can easily be transformed into pairwise constraints, the labels of data objects cannot be inferred from pairwise constraints.

In addition to instance-level constraints, there can be several other constraints in a semi-supervised clustering problem. Balancing constraints would force the sizes of the clusters to be comparable. More generally, given data objects with associated positive weights, one may restrict the total weight of the data objects in each cluster (this is known as the capacitated clustering problem). In particular, if all the weights are one, then the size of each cluster is restricted. There could be constraints that put lower or upper bounds on the radii of the clusters. Given an initial clustering of the data, one may want to obtain a clustering that is "different" from the initial one. All such constraints, i.e., the constraints different from instance-level constraints, can be named as cluster-level constraints.

In semi-supervised clustering algorithms, the given constraints can be considered as hard or soft. Hard constraints must be satisfied in the final clustering given by the algorithm. Soft constraints, however, are allowed to be violated by incurring some violation costs which are usually incorporated in the objective function. The advantage of algorithms considering constraints as soft over algorithms considering constraints as hard is that the former usually better handles noisy constraints. Algorithms in which the constraints are taken as hard may not find a feasible partition if there is noise with the constraints. While algorithms may end up with no partition at all in hard constraint case, the algorithms find a partition in every case with some amount of constraint violations in the soft constraint case.

Semi-supervised clustering algorithms fall into three categories [12], namely search based methods, distance based methods and hybrid methods. In search based methods, clustering algorithms are modified to incorporate the prior knowledge into the clustering task. In other words, the solution space to be searched is adjusted according to the constraints. Common techniques in search based methods are modifying the objective function by adding penalty terms for unsatisfied constraints, enforcing constraints to be satisfied and using prior knowledge to initialize clusters. In distance based methods, an existing clustering method is generally used but the distance measure of the method is modified in accordance with the prior knowledge. The distance measure is adjusted in such a way that data objects that should be placed in the same cluster will be closer to each other while data objects that should be placed in different clusters will be farther away from each other. Hybrid methods incorporate search and distance based methods and usually outperform the individual methods [12].

2.2.2.1 Semi-supervised Clustering with Labeled Data

Semi-supervised clustering with limited number of labeled data can be considered as a multi objective optimization problem with objectives of maximizing intra-cluster similarity, maximizing inter-cluster dissimilarity and minimizing cluster impurity which is a measure of the consistency between the partition and the prior knowledge (labels). To solve such a multi objective optimization problem Basu et al.[10] proposes an algorithm which is a modifications of k-means and Demiriz et al. [47] uses a genetic algorithm. These methods can be categorized as search based method.

The same problem can also be handled in two stages instead of considering it as a multi objective optimization problem. In the first stage, cluster impurity is considered. In this stage, data objects are transformed into a new space using the prior knowledge on hand. This transformation is done in such a way that objects having the same label will be closer to each other and objects with different labels will be farther away from each other in the new space. This first stage is called as distance metric learning. After the metric is learnt, cluster dispersion is considered in the second stage where traditional clustering algorithms are generally employed. Such methods are distance based methods. In [161], authors propose a parametric distance learning method for the clustering problem with labeled data.

2.2.2.2 Semi-supervised Clustering with Instance-Level Constraints

In search based methods for clustering with constraints, the problem can be considered as a multi objective optimization problem as in the case with clustering in the presence of some labeled data. In addition to cluster dispersion, a measure of constraint violation is used as another objective instead of using cluster impurity. Such methods generally modify the well known clustering algorithms like k-means and COBWEB to solve the problem (see, e.g. [144], [145], [11], [146], [44], [9]). Alternatively, there are graph-theoretic solution method [158] and an algorithm that make use of instance-level constraints in an agglomerative hierarchical clustering [116].

Distance based methods handle the same problem in two stages. In the first stage, a new distance metric which brings must linked data objects closer and pushes cannot linked data objects apart is defined. A measure of dispersion as a function of this newly defined distance metric is then used in the second stage for clustering. [91], [87] and [35] propose new distance metrics that uses the instance level constraints. Xing et al. [154] finds the optimal distance metric for a given set of instance level constraints by a convex optimization model. Bar-Hillel et al. [8] uses relative component analysis (RCA) clustering with instance level constraints.

Hybrid methods benefit from the advantages of both search based and distance based methods and generally perform better than the individual methods. Lately, noticing this potential, some authors proposed hybrid methods (see, e.g., [12], [17], [13]).

Law et al. [98, 99] also address the problem of clustering with instance level constraints. Most of the methods mentioned so far consider the constraints as correct and consistent. Authors consider the constraints as random variables and propose an expectation-maximization (EM) algorithm in their study.

2.2.2.3 Semi-supervised Clustering with Cluster-Level Constraints

Semi-supervised clustering with cluster-level constraints did not receive too much attention as clustering with instance level constraints in data mining community. Constraints like forcing the sizes of the groups (clusters) to be comparable and putting lower or upper bounds on the radii or diameter of the groups are generally used in facility location problems. A facility location problem is the problem of finding locations of a predetermined number of facilities to serve the customers so as to optimize a certain objective like a function of distances between facilities and customers assigned to them. If we consider the facility locations as cluster centers, customers as data objects and the objective to be optimized in the facility location problem as a cluster dispersion measure, facility location problem with such constraints and the clustering with cluster-level constraints are very alike. For example, in [57], Drezner proposes two heuristics and an optimal algorithm for the p-center problem encountered in the facility location literature which can be considered as clustering with upper bounds on the radii of the clusters.

The methods proposed for clustering with cluster-level constraints are all search based methods. A distance metric learning in such problems is not applicable as there is no aim of bringing some data objects closer to each other while pulling some others apart.

In the paper by Davidson and Ravi [43], instance-level constraints and two types of cluster-level constraints are used in a search based agglomerative hierarchical clustering algorithm. The cluster level constraints considered are δ–constraint that enforces a distance of at least δ between any two data objects that are not in the same cluster and ϵ–constraint that requires that for any data object if there is any other object in the same cluster, then there should be at least one object which is at most ϵ away from it.

[7] and [69] propose algorithms to find balanced clusters (with equal size). Balancing constraints can be seen as a special case of size constraints which put constraints on the size of each cluster (or on the total weight of the data objects in each cluster in the case with weights on the data objects). This is known as the capacitated clustering problem. Zhu et al. [162] provide 0-1 integer linear programming problem. Bradley et al. [25] also uses integer programming but thanks to the unimodularity of their problem they solve LP. [67] modifies k-means algorithm. [140] uses a graph based algorithm. Other solutions approaches and applications of the capacitated clustering problem can be found in [38, 68, 117].

Gonzales [70] considers the clustering problem that minimizes the maximum intercluster distance. He proposes an approximation algorithm which has $O(kn)$ time complexity where k is the number of clusters and n is the number of data objects.

2.2.2.4 Feasibility Issues for the Clustering with Constraints

In this section, we review the complexity of the clustering feasibility problem under constraints that is the problem of finding a feasible partition of the data satisfying all of the given constraints. Four types of constraints have been considered in the literature [43, 44, 91]; namely must-link, cannot-link, δ, and ϵ constraints. Two types of feasibility problems are considered:

- Given a value of k, does there exist a feasible partition of the data into k clusters satisfying all the given constraints?

- Given the constraints, does there exist a feasible partition of the data (into any number of clusters) satisfying all the constraints?

The complexity results are due to [43, 44, 91]. The following table summarizes the complexity of the clustering feasibility problem under different combinations of the constraints and is taken from [43].

Table 2.1: Complexity of the clustering feasibility problem

Constraint Combination	k given	k not given
Cannot-link	NP-complete [44, 91]	P [43]
Must-link and δ	P [44]	P [43]
Must-link and ϵ	NP-complete [44]	P [43]
δ and ϵ	P [44]	P [43]
Must-link, Cannot-link, δ and ϵ	NP-complete [44]	NP-complete [43]

In general, the feasibility problem with only must-link constraints can be solved in polynomial time whether k is given or not. The feasibility problem with only cannot-link constraints is polynomial time solvable when k is not given, but is NP-complete in general when k ($\geq$ 3) is specified. The feasibility problem with only δ constraint or only ϵ constraint can be solved in polynomial time in both cases. Note that, in general, when a combination of ϵ constraint and must-link constraints are given, the feasibility problem becomes NP-complete for a fixed value of k even though the problem is easy if only one of the constraint types is given.

2.2.3 Clustering of Uncertain Data

The uncertainty in the location of the data objects can be due to various reasons, e.g., imprecision in measurements, sampling error, reporting errors, randomness inherent in various phenomena. In such cases, taking the locations of the data objects as regions is more appropriate than taking them as fixed points. Clustering of data objects whose locations are uncertain (i.e. data objects can be represented as regions) is an important issue and appears in several real life settings. Consider, for example, clustering of digital cameras whose different aspects (e.g., image quality, battery performance) are rated by users [84]. The ratings of each camera can be represented by a multivariate probability distribution on a multidimensional region. If the worst-case scenarios are important, then the density of the distribution is of no relevance, but rather the multidimensional region that the distribution is defined over (where the density is nonzero) is important. If certain cameras are (not) to be placed in the same cluster, some instance level constraints can be introduced to guide the clustering algorithm in the direction of the desired clustering.

Another example would be clustering of cities based on atmospheric conditions like temperature and humidity on a particular month. As the atmospheric conditions may vary during a specific month, each city may be represented by a multivariate probability distribution on a multidimensional region which is not necessarily a hyper-rectangle as temperature and humidity are not independent.

When data objects are represented by regions or multivariate probability distributions, the distance between an object and a given point can be measured in different ways, such as

- expected distance,
- maximum distance,
- minimum distance.

The expected distance is generally used when the distance to every point in the region is important. If the realizations of the distribution occur frequently, then the use of the expected distance may be justified. For uncertain (or regional) data objects, the expected distance has been used in a number of studies in the clustering and facility location literature.

In [36], the authors use expected distances and propose an extension of the k-means algorithm for clustering of uncertain data objects each of which is represented by an uncertainty region and a probability density function. An improved versions of this algorithm are proposed in [119] and [100]. They aim to reduce the number of expected distance calculations and the time needed for those calculations. As the computation of the expected distance between an uncertain object and a point is expensive, the authors in [72] propose an extension of the k-medoids algorithm [88] which computes the expected distances once at the beginning.

Expected distance has also been used in the facility location literature for the cases where the customers are represented by regions or with probability distributions, see, e.g., [106, 15, 40, 4, 33].

One disadvantage of using expected distances is that the centroid found may be very far away from certain realizations of the locations of the data objects. In cases where the worst case scenarios are important, e.g., when locating emergency facilities or when looking for robust solutions, the maximum distance is rather used, see e.g., [54, 85, 86, 58].

The minimum distance is usually used in the facility location community for the problems where drop-off and take-away points of a regional demanding entity, i.e., customer, are on the boundary of the region and the internal distribution within the region is of no concern, see e.g., [28, 29]. A survey on facility location problems where the customers are represented by regions or with probability distributions is given in [50].

2.3 Notation

In this section, the notation is introduced. The data objects that are to be clustered are assumed to be either polytopes or disks in d-dimensional Euclidean space $\mathbf{R}^d$. Let R_i denote the i^{th} regional data object, $i \in \{1, 2, \ldots, n+m\}$. It is assumed that the first n objects, i.e., $R_1, R_2, \ldots, R_n$, represent polytopes and the last m objects, i.e., $R_{n+1}, R_{n+2}, \ldots, R_{n+m}$, represent closed disks. The set of coordinates of corners of polytope R_i is denoted by ℓ^i ($\subset \mathbf{R}^d$) for every $i \in \{1, 2, \ldots, n\}$ (so R_i has $|\ell^i|$ many corners). The center and radius of disk R_i is denoted by o_i and r_i, respectively, for any $i \in \{n+1, n+2, \ldots, n+m\}$. Associated with the i^{th} regional data object R_i, there is a positive weight α_i, $i \in \{1, 2, \ldots, n+m\}$.

A partition, $C_1, C_2, \ldots, C_k$, of the $n+m$ regional data objects, $R_1, R_2, \ldots, R_{n+m}$, into

k clusters is looked for given some must-link and cannot-link constraints. Here C_j represents the j^{th} cluster and its centroid is denoted by μ_j for any $j \in \{1, 2, \dots, k\}$. The set of all must-link and cannot-link constraints are indicated by ML and CL, respectively. Any constraint $\{i_1, i_2\} \in ML$ implies that the regional data objects R_{i_1} and R_{i_2} are to be placed in the same cluster. A cannot-link constraint $\{i_1, i_2\} \in CL$, on the other hand, is used to express the condition that the regional data objects R_{i_1} and R_{i_2} should not be placed in the same cluster. When the constraints are taken as hard constraints, no constraint violation is allowed in the final clustering. In the soft constraint case, however, constraints are allowed to be violated in the final clustering. In this case, violation costs (penalties) are incurred by violated constraints. The cost associated with the violation of a must-link constraint and a cannot-link constraint are assumed to be equal to $c_1 > 0$ and $c_2 > 0$, respectively.

2.4 Computation of the Centroid of a Given Cluster

Given a cluster consisting of some polytopes and disks, the problem of how to compute its centroid, namely the centroid computation problem (CCP), is considered in this section. This problem appears as a subproblem in many of the clustering algorithms discussed in this chapter and therefore its solution is of great importance.

Assume that cluster C consists of n polytopes and m closed disks in $\mathbf{R}^d$. Its centroid $\mu^* \in \mathbf{R}^d$ is the point in the Euclidean space that minimizes the sum of squares of the maximum Euclidean distances between the regional data objects and μ^*. Mathematically, we have

$$\mu^* = \operatorname*{argmin}_{\mu \in \mathbf{R}^d} \phi(\mu), \tag{2.1}$$

where

$$\phi(\mu) = \sum_{i=1}^{n} \alpha_i (\max_{p_i \in \ell_i}(\|p_i - \mu\|))^2 + \sum_{i=n+1}^{n+m} \alpha_i (\|o_i - \mu\| + r_i)^2. \tag{2.2}$$

Here, $\max_{p_i \in \ell_i}(\|p_i - \mu\|)$ is the maximum Euclidean distance between μ and the polytope R_i, $i \in \{1, 2, \dots, n\}$ which is obtained by considering the distances between μ and the corners of R_i and taking the maximum. The term $\|o_i - \mu\| + r_i$ in the second summation is the maximum distance between μ and the disk R_i, $i \in \{n+1, n+2, \dots, n+m\}$, which is equal to the distance between μ and the center of the disk,

o_i, plus the radius r_i of the disk. Note that as the sum and the maximum of convex functions are convex, ϕ is a convex function.

The rest of this section is devoted to solution approaches for the CCP. In Section 2.4.1, a second order cone programming (SOCP) formulation for the problem is introduced. This formulation shows that the CCP can be solved in polynomial time. A similar SOCP formulation for the CCP with only polygonal data objects is formulated in [54]. In Section 2.4.2, a subgradient algorithm is proposed to be able to solve the CCP faster. This is important as the CCP is solved several times as a subproblem in some clustering algorithms considered in this chapter.

2.4.1 An SOCP Formulation for the CCP

A second order cone programming (SOCP) problem is the problem of optimizing a linear objective function over second order cone constraints. It is a convex optimization problem of the form

$$\begin{array}{ll} \text{minimize} & v^T x \\ \text{subject to} & \|A_t x + b_t\| \leq \gamma_t^T x + \theta_t, t = 1, 2, \ldots, w. \end{array} \tag{SOCP}$$

Here, $x \in \mathbf{R}^q$ denotes the vector of decision variables, and $v \in \mathbf{R}^q, A_t \in \mathbf{R}^{q_t \times q}, b_t \in \mathbf{R}^{q_t}, \gamma_t \in \mathbf{R}^q, \theta_t \in \mathbf{R}$ denote the problem parameters. The constraints of (SOCP) are called second order cone constraints. Note that a linear inequality $a^T x \leq \beta$ can easily be written as a second order cone constraint as $\|Ax + b\| \leq \gamma^T x + \theta$, where A is the matrix of all zeros, b is the vector of all zeros, $\gamma = -a$ and $\theta = \beta$. Therefore SOCP problems generalize linear programming (LP) problems. Like LP problems, SOCP problems are polynomial time solvable. Several numerically stable and computationally efficient implementations of solution algorithms have been developed for SOCP problems. For an overview of SOCP, examples, application areas and solution approaches, the reader is referred to [105], [3].

A mixed integer second order cone programming (MISOCP) problem is a related problem in which some or all decision variables in (SOCP) are constrained to take on integer values. MISOCP problems are NP-hard in general and usually solved by branch-and-bound algorithms that call an SOCP solver at each node of the branch-

and-bound tree.

An SOCP formulation for the CCP is proposed in [54] for the case when all the data objects are polygons in the plane. So, the formulation provided is not novel, but it is extended here to also handle the other regional data objects, namely disks, considered in this chapter. Let e_i denote the squared maximum distance between the polytope R_i and the cluster centroid μ, $i \in \{1, 2, \dots, n\}$, and f_i denote the squared maximum distance between the disk R_i and the cluster centroid μ, $i \in \{n+1, n+2, \dots, n+m\}$. An SOCP formulation for the CCP problem is given below.

$$
\begin{array}{ll}
\text{minimize} & \left(\sum_{i=1}^{n} \alpha_i e_i + \sum_{i=n+1}^{n+m} \alpha_i f_i\right) \\
\text{subject to} & e_i \geq \|p_i - \mu\|^2, i = 1, 2, \dots, n, p_i \in \ell^i, \\
 & f_i \geq \delta_i^2, i = n+1, n+2, \dots, n+m, \\
 & \delta_i - r_i \geq \|o_i - \mu\|, i = n+1, n+2, \dots, n+m.
\end{array}
\qquad \text{(SOCP-CCP)}
$$

The objective function minimizes the weighted sum of squared maximum Euclidean distances between the data objects and the cluster centroid. The first set of constraints imply that e_i is greater than or equal to the squared distance between the cluster centroid and every corner of R_i for any $i \in \{1, 2, \dots, n\}$. The sense of the objective function and the positive coefficient α_i make sure that in an optimal solution e_i is equal to the squared maximum distance among all. Similarly, the second and the third sets of constraints enforce that f_i is greater than or equal to the squared maximum Euclidean distance between the cluster centroid and the disk R_i for any $i \in \{n+1, n+2, \dots, n+m\}$. The objective function ensures the equality in an optimal solution.

The first and second sets of constraints are not in the form of second order cone constraints. They can, however, be converted into equivalent second order cone constraints [3].

The worst case complexity of an SOCP formulation in general is $O(v^2\sqrt{N}\sum_{i=1}^{N} n_i)$, where v is the number of decision variables, N is the number of cone constraints, and n_i is the dimension of the i^{th} cone constraint [105]. In (SOCP-CCP), there are n and m decision variables representing the square of the maximum distances of polytopes and disks from the cluster centroid, respectively, m decision variables denoting the maximum distances of disks from the cluster centroid, and d decision variables expressing the coordinates of the cluster centroid. Thus, there are $v = n + 2m + d$ decision variables in total. The number of cone constraints related with polytope i is equal to the number of corners of the polytope, i.e, $|\ell^i|$. Therefore there are $\sum_{i=1}^{n} |\ell^i|$ cone constraints for polytopes. For each disk, there are 2 cone constraints. Therefore, the total number of cone constraints in the formulation is $N = \sum_{i=1}^{n} |\ell^i| + 2m$. Since each cone constraint is of at most dimension $d + 1$, the term $\sum_{i=1}^{N} n_i$ can be rewritten as $N(d + 1)$. Then, the worst case complexity of the SOCP formulation is $O((n+m+d)^2 d(\sum_{i=1}^{n} |\ell^i|+m)^{3/2})$. It should be noted that the worst case complexity increases with the number of regional data objects, the number of corners of the polytopes, and the dimension of the regional data objects. For more on the time and space complexity of the SOCP problems, the reader is referred to [3, 105].

2.4.2 Subgradient Algorithm for the CCP

The CCP is a convex but nonsmooth minimization problem and therefore the subgradient method that was originally proposed by Shor [132] would be a suitable solution alternative. The method, at each iteration, takes a step in the direction of a negative subgradient as

$$\mu^{s+1} = \mu^s - \tau_s g(\mu^s). \tag{2.3}$$

Here, μ^s is the centroid at the s^{st} iteration, τ_s is the step size at the s^{st} iteration, and $g(\mu^s)$ is a subgradient of ϕ (as defined in Equation (2.2)) at μ^s.

We first characterize the subdifferential of ϕ, $\partial\phi$, at any solution $\mu \in \mathbf{R}^d$, which is the set of all subgradients at μ. Let $\phi^1(\mu) = \sum_{i=1}^{n} \alpha_i (\max_{p_i \in \ell_i}(\|p_i - \mu\|))^2$ and $\phi^2(\mu) = \sum_{i=n+1}^{n+m} \alpha_i (\|o_i - \mu\| + r_i)^2$. Then, we have

$$\partial\phi(\mu) = \partial\phi^1(\mu) + \partial\phi^2(\mu). \tag{2.4}$$

Although ϕ^1 is nondifferentiable, it can be written as the maximum of a certain number of differentiable functions as

$$\phi^1(\mu) = \max_{p_i \in \ell^i,\ i=1,\dots,n} \left(\sum_{i=1}^{n} \alpha_i \|p_i - \mu\|^2 \right) = \max_{\Omega=1,\dots,\prod_{i=1}^{n}|\ell^i|} \phi^1_\Omega(\mu). \tag{2.5}$$

The function ϕ^1_Ω is said to be 'active' at μ if $\phi^1_\Omega(\mu) = \phi^1(\mu)$. Let $I(\mu)$ be the set of indices of all 'active' functions at μ and $I(\mu) = \{\Omega | \phi^1_\Omega(\mu) = \phi^1(\mu)\}$. Note that $\partial\phi^1_\Omega(\mu)$ is equal to the gradient of ϕ^1_Ω at μ. Thus, we have by [24]

$$\partial\phi^1(\mu) = \text{conv} \bigcup_{\Omega \in I(\mu)} \partial\phi^1_\Omega(\mu). \tag{2.6}$$

In our computational experiments, to obtain an 'active' function at μ we take one of the equidistant farthest points of each polytope data object from μ.

Now, we characterize $\partial\phi^2$. Let $\phi^2_i(\mu) = \alpha_i(\|o_i - \mu\| + r_i)^2 = \alpha_i(\|o_i - \mu\|^2 + 2r_i\|o_i - \mu\| + r_i^2)$ which is nondifferentiable for each $i \in \{n+1, n+2, \ldots, n+m\}$. Since $\phi^2(\mu) = \sum_{i=n+1}^{n+m} \phi^2_i(\mu)$, we have

$$\partial\phi^2(\mu) = \sum_{i=n+1}^{n+m} \partial\phi^2_i(\mu), \tag{2.7}$$

where

$$\partial\phi^2_i(\mu) = \begin{cases} \alpha_i \left(2(\mu - o_i) + 2r_i \dfrac{(o_i - \mu)}{\|o_i - \mu\|} \right), & \text{if } \mu \neq o_i \\ \alpha_i \left(2(\mu - o_i) + 2r_i\{u : \|u\| \leq 1\} \right), & \text{otherwise.} \end{cases} \tag{2.8}$$

An important aspect of the subgradient method is the size of the step taken at each iteration. We conduct some preliminary computational experiments to make a selection among different step size rules presented in the literature. The step size rules used in the experiments are:

- Square summable but nonsummable step size rule [23]:
  Any step size satisfying $\sum_{s=1}^{\infty} \tau_s^2 < \infty$, $\sum_{s=1}^{\infty} \tau_s = \infty$,
- Nonsummable diminishing step size rule [23]:
  Any step size satisfying $\lim_{s\to\infty} \tau_s = 0$, $\sum_{s=1}^{\infty} \tau_s = \infty$,
- Polyak step size rule [124]:
  $\tau_s = \frac{\phi(\mu^s) - \phi^{(s)}_{best} + \gamma_s}{\|g(\mu^s)\|^2}$ where $\phi^{(s)}_{best} = \min\{\phi(\mu^1), \ldots, \phi(\mu^s)\}$, $\sum_{s=1}^{\infty} \gamma_s^2 < \infty$ and $\sum_{s=1}^{\infty} \gamma_s = \infty$. $\phi^{(s)}_{best} - \gamma_s$ is an estimation of ϕ^*, the optimal objective function value of the problem, at iteration s.

All of the mentioned rules guarantee convergence to the optimal solution of the CCP.

Setting a reasonable stopping criteria is another decision to be made. A common stopping criterion is to stop the algorithm when the difference between upper and lower bounds on ϕ^* is less than a threshold value, ϵ. $\phi^{(s)}_{best}$ can be used as an upper bound on ϕ^*. For a lower bound on ϕ^*, we implement the formula proposed in [23]. A lower bound on ϕ^* at iteration s, $\underline{\phi}^{(s)}$, is

$$\underline{\phi}^{(s)} = \frac{2\sum_{i=1}^{s} \tau_i \phi(\mu^i) - D^2 - \sum_{i=1}^{s} \tau_i^2 \left\| g(\mu^i) \right\|^2}{2\sum_{i=1}^{s} \tau_i}, \tag{2.9}$$

where D is an upper bound on the distance between the initial solution and the optimal solution. As the sequence $\left\{\underline{\phi}^{(s)}\right\}$ is not necessarily nondecreasing, the best lower bound on ϕ^* till iteration s, $\phi^{(s)}_{worst}$, is computed as $\phi^{(s)}_{worst} = \max\left\{\underline{\phi}^{(1)}, \ldots, \underline{\phi}^{(s)}\right\}$. Another stopping criterion is to stop the algorithm when the number of iterations reaches a predefined number. Combining the two stopping criteria, we terminate the algorithm when $\phi^{(s)}_{best} - \phi^{(s)}_{worst} \leq \epsilon$ or the maximum number of iterations is achieved.

We design a full factorial experiment with the following design factors

- Step size rule (4 levels : Polyak step size rule with $\gamma_s = \frac{5}{s}$, Polyak step size rule with $\gamma_s = \frac{10}{s}$, square summable but nonsummable step size rule with $\tau_s = 1/s$, nonsummable diminishing step size rule with $\tau_s = 1/\sqrt{s}$),
- ϵ (3 levels : 0.001, 0.0001, 0.00001),
- Maximum number of iterations (3 levels : 100, 500, 1000).

Considering the trade off between the solution quality and solution time, we select nonsummable diminishing step size rule with $\tau_s = 1/\sqrt{s}$, $\epsilon = 0.00001$ and maximum number of iterations=100.

The steps of the subgradient method for a given instance of the CCP with polytopes and closed disks are given in Algorithm 3, named as SM.

Solving the CCP in a time efficient manner is crucial since it is solved as a subproblem in the algorithms discussed in the next section. The proposed solution methods, namely, the SOCP formulation and SM, are compared in terms of the solution times

Algorithm 3 SM

```
1: Initialization: Start with an initial cluster centroid, $\mu^0 \in \mathbf{R}^d$ and set iteration number $s = 0$.
2: loop
3:     Find a farthest point of each polytope $R_i,\ i \in \{1, 2, \dots, n\}$ from the cluster centroid $\mu^s$ (ties are arbitrarily broken) and for each disk $R_i,\ i \in \{n+1, n+2, \dots, n+m\}$ check whether $\mu^s = o_i$ or not.
4:     Find a subgradient of $\phi$ at $\mu^s$, $g(\mu^s)$, using Equation (2.4).
5:     Calculate the step size, $\tau_s = 1/\sqrt{s}$.
6:     Set $\mu^{s+1} = \mu^s - \tau_s g(\mu^s)$.
7:     Calculate the objective function value at $\mu^{s+1}$, $\phi(\mu^{s+1})$, by Equation (2.2).
8:     Find lower bound on $\phi^*$ at $\mu^{s+1}$, $\underline{\phi}^{(s+1)}$, by Equation (2.9).
9:     Find $\phi_{best}^{(s+1)}$ and $\phi_{worst}^{(s+1)}$.
10:    if $\phi_{best}^{(s+1)} - \phi_{worst}^{(s+1)} \leq \epsilon$ or $s + 1 = 100$ then
11:        return Optimal cluster centroid, $\mu^{s+1}$.
12:    else
13:        Set $s = s + 1$.
14:    end if
15: end loop
```

for a number of randomly generated problem instances with only rectangular data objects in the plane and the results are shown in Table 2.2. The performance of SM is better than that of the SOCP formulation. Thus, to solve the CCP appearing as subproblems in the algorithms discussed in Section 2.5, we prefer SM over the SOCP formulation. The subgradient algorithm needs $O(1/\epsilon^2)$ iterations [141]. The details of the time complexity for an iteration are as follows. In Step 3 of the algorithm, to find the farthest point of a polygon from the cluster centroid, the distances between the cluster centroid and each corner of the corresponding polytope need to be computed. Calculating the distance between two points in the Euclidean space $\mathbf{R}^d$ takes $O(d)$ time. Thus, in the worst case, the time required to find the farthest point of a polytope from the cluster centroid is $O(d\ell)$ where $\ell = max\{|\ell^1|, \dots, |\ell^n|\}$. Checking whether the cluster centroid and the center of a disk are the same or not requires $O(d)$ time. Therefore the total time required in Step 3 is $O(nd\ell + md)$. In Step 4, finding

Table 2.2: Comparison of SM and SOCP in terms of computational time (in seconds)

n	SM	SOCP
5	0.023	0.484
10	0.017	0.511
20	0.033	0.852
50	0.079	1.894
100	0.150	3.753
200	0.300	7.361
500	0.776	18.661

$\partial\phi^1(\mu)$ and $\partial\phi^2(\mu)$ takes $O(nd)$ and $O(md)$ time, respectively. Step 5 and Step 6 require $O(1)$ and $O(d)$ time, respectively. Time required in Step 7 is the same as that in Step 3 since the same operations are executed, i.e., finding maximum distances between data objects and the cluster centroid. Step 8 takes $O(d)$ time. Finally, each one of the remaining steps requires $O(1)$ time. In total, each iteration of SM requires $O(d(n\ell + m))$ time where $\ell = max\{|\ell^1|, \ldots, |\ell^n|\}$. The space complexity of the algorithm is linear in the size of the input.

2.5 Semi-supervised Clustering with Reginal Data

In this section, we first give a mathematical programming formulation for the considered clustering problem, namely, the semi-supervised clustering problem (SSC), which is only able to solve small size instances to optimality. After reviewing some algorithms from the literature proposed mainly for semi-supervised clustering problems in which the data objects are represented as fixed points, we provide some modifications to make them applicable to the SSC.

2.5.1 An MISOCP Formulation for the SSC

Let a_{ij} be a binary variable introduced to assign R_i to C_j, $i \in \{1, 2, \ldots, n+m\}$, $j \in \{1, 2, \ldots, k\}$. The variable will take the value 1 if and only if R_i is assigned to

C_j. Also, let $v^1_{i_1 i_2}$ be a binary variable defined to check whether a must-link constraint $\{i_1, i_2\} \in ML$ is violated or not. The variable will take the value 1 if and only if R_{i_1} and R_{i_2} are placed into different clusters. Similarly, let $v^2_{i_1 i_2}$ be a binary variable defined for a cannot-link constraint $\{i_1, i_2\} \in CL$ and it will take the value 1 if and only if R_{i_1} and R_{i_2} are placed into the same cluster.

Redefining e_i and f_i as the square of the farthest distance between polytope R_i, $i \in \{1, 2, \ldots, n\}$, and the nearest cluster centroid, and the square of the farthest distance between disk R_i, $i \in \{n + 1, n + 2, \ldots, n + m\}$, and the nearest cluster centroid, respectively, an MISOCP formulation for the SSC is given below.

$$\begin{array}{ll}
\text{minimize} & \sum_{i=1}^{n} \alpha_i e_i + \sum_{i=n+1}^{n+m} \alpha_i f_i + c_1 \sum_{\{i_1,i_2\}\in ML} v^1_{i_1 i_2} + c_2 \sum_{\{i_1,i_2\}\in CL} v^2_{i_1 i_2} \\
\text{subject to} & e_i + (1 - a_{ij})M \geq \|p_i - \mu_j\|^2, i = 1, 2, \ldots, n, p_i \in \ell_i, j = 1, 2, \ldots, k, \\
& f_i \geq \delta_i^2, i = n + 1, n + 2, \ldots, n + m, \\
& \delta_i - r_i + (1 - a_{ij})M \geq \|o_i - \mu_j\|, i = n + 1, \ldots, n + m, j = 1, \ldots, k, \\
& \sum_{j=1}^{k} a_{ij} = 1, i = 1, 2, \ldots, n + m, \\
& a_{i_1 j} - a_{i_2 j} \leq v^1_{i_1 i_2}, \{i_1, i_2\} \in ML, j = 1, 2, \ldots, k, \qquad \text{(MISOCP-SSC)} \\
& a_{i_2 j} - a_{i_1 j} \leq v^1_{i_1 i_2}, \{i_1, i_2\} \in ML, j = 1, 2, \ldots, k, \\
& a_{i_1 j} + a_{i_2 j} \leq v^2_{i_1 i_2} + 1, \{i_1, i_2\} \in CL, j = 1, 2, \ldots, k, \\
& a_{ij} \in \{0, 1\}, i = 1, 2, \ldots, n + m, j = 1, 2, \ldots, k, \\
& v^1_{i_1 i_2} \in \{0, 1\}, \{i_1, i_2\} \in ML, \\
& v^2_{i_1 i_2} \in \{0, 1\}, \{i_1, i_2\} \in CL,
\end{array}$$

where M is a big number. The objective function minimizes the total of the sum of the violation costs of instance level constraints and the weighted sum of squared maximum Euclidean distances between the data objects and the centroids of the clusters they are assigned to. The first set of constraints together with the sense of the objective function and the positive coefficient α_i imply that e_i is equal to the squared maximum Euclidean distance between R_i and the centroid of the cluster R_i is assigned to for every $i \in \{1, 2, \ldots, n\}$. Similarly, the second and the third sets of constraints ensure that f_i is forced to be equal to the squared maximum Euclidean distance between R_i and the centroid of the cluster R_i is assigned to for every $i \in \{n + 1, n + 2, \ldots, n + m\}$ in an optimal solution by the objective function. The fourth set of constraints indicates that each data object is assigned to exactly one cluster. The fifth and sixth sets of constraints together with the objective function impose that the variable $v^1_{i_1 i_2}$ takes the value 1 in an optimal solution if and only if the data objects R_{i_1} and R_{i_2} for $\{i_1, i_2\} \in ML$ are assigned to different clusters. Similarly, the seventh set of constraints and the objective function make sure that the variable $v^2_{i_1 i_2}$ takes the value 1 in an optimal solution if and only if the data objects R_{i_1} and R_{i_2} for $\{i_1, i_2\} \in CL$ are assigned to the same cluster. The other constraints are integrality constraints.

The first three sets of constraints can be converted into second order cone constraints, see e.g., [3]. The remaining sets of constraints are linear constraints that an MISOCP formulation can handle.

This formulation is weak mainly due to the presence of the big-M in the formulation. It can be used to solve small size problem instances to optimality, but it becomes time and memory inefficient for medium or large size problem instances. Therefore, we need time and memory efficient algorithms to solve the SSC.

Note that the above MISOCP formulation is proposed for the soft instance level constraints case. However, it can be easily modified for the hard instance level constraints case in two ways. The first way is setting c_1 and c_2 to very big numbers and checking the objective function value or the values of the binary variables obtained after solving the above MISOCP formulation. The second way of making instance level constraints hard is by modifying the decision variables, the objective function, and the constraints of the model. In that case, the decision variables $v^1_{i_1 i_2}$, $\{i_1, i_2\} \in ML$ and $v^2_{i_1 i_2}$, $\{i_1, i_2\} \in CL$ are not required. Thus, the terms related with those variables in the objective function and the integrality constraints related with those variables are deleted from the model. Moreover, instead of the fifth, sixth, and seventh sets of constraints, the following two sets of constraints are added to the model to ensure that must-link and cannot-link constraints are certainly satisfied:

$$\begin{aligned} &a_{i_1 j} = a_{i_2 j}, \{i_1, i_2\} \in ML, j = 1, 2, \ldots, k, \\ &a_{i_1 j} + a_{i_2 j} \leq 1, \{i_1, i_2\} \in CL, j = 1, 2, \ldots, k. \end{aligned}$$

In the next two subsections, we describe five algorithms that are modifications of the traditional k-means algorithm and an agglomerative hierarchical clustering algorithm for the solution of the SSC. These algorithms are extensions of the algorithms proposed in the literature for the solution of the semi-supervised clustering problems with point data objects.

2.5.2 K-means based Algorithms for the SSC

In this subsection, we provide the details of COP-k-means [145], PC-k-means [11], and CVQE [44] which are k-means based algorithms proposed for semi-supervised clustering problems with point data objects in the presence of instance level constraints. Moreover, we review two k-means based algorithms proposed for semi-supervised clustering problems with point data objects in the presence of a few number of labeled data, namely Seeded-k-means and Constrained-k-means [10]. These algorithms are also applicable in the existence of instance level constraints after small modifications. All of these algorithms are modified for the solution of the SSC and the modified versions are called as UCOP-k-means, UPC-k-means, UCVQE, USeeded-k-means and UConstrained-k-means, respectively. We next provide the details of the original and the modified versions of the algorithms.

2.5.2.1 UCOP-k-means

The initialization and update steps of COP-k-means are the same as those of the k-means. The assignment step of COP-k-means, however, is different. Each data object is assigned to the closest 'appropriate' cluster. For a data object, a cluster is 'appropriate' if its placement in the cluster does not violate any instance level constraint considering the data objects that have already been assigned to the clusters. In this sense, the instance level constraints are handled as hard constraints. As the authors of COP-k-means noted, the assignment step of the algorithm is order-dependent. The algorithm may return an empty partition when an 'appropriate' cluster cannot be found for a data object.

We now provide the details of UCOP-k-means which is a modified version of COP-k-means for the SSC. The first modification is in the assignment step. We measure closeness by the maximum distance between the regional (polytopes or disks) data objects and the cluster centroids. At the very beginning of the algorithm we randomly order the data objects in order to lessen the effect of order-dependence and keep that ordering until the end of the algorithm. If an ‘appropriate’ cluster is not found for a given data object in the first iteration, we re-order the data objects and re-start the assignment step until a partition is obtained for the first iteration. In the next iterations, if an empty partition is returned, we stop the algorithm and return the best solution found so far considering the objective function of the MISOCP formulation. The second modification is in the update step of the algorithm. We use the subgradient method proposed in Section 2.4.2. The assignment and update steps of UCOP-k-means alternate until assignments found in two successive iterations are the same or the number of iterations reaches a predefined maximum number of iterations. The details of UCOP-k-means are given in Algorithm 4.

2.5.2.2 UPC-k-means

The update step of PC-k-means is the same with that of the k-means. The instance level constraints are considered in the initialization and assignment steps of PC-k-means. In the initialization step of PC-k-means, equivalence classes (chunklets) are formed by taking the transitive closure of the must-link constraints. Let the number of such chunklets be ζ. If $\zeta \geq k$, then the centroids of the largest k of them (in terms of size) are used as the initial centroids. If $\zeta < k$, then using the centroids of all the chunklets, ζ centroids are initialized. For the $(\zeta + 1)^{th}$ cluster centroid, a data object which has a cannot-link constraint with every chunklet is used if such an object exists. The remaining centroids are initialized by randomly perturbing the global centroid. A modified objective function that is the same with the objective function of the MISOCP formulation in Section 2.5.1, consisting of the minimum-sum-of-squares objective function used in the k-means and the constraints violation cost, is considered in the assignment step of PC-k-means. Each data object is assigned to the cluster that minimizes the contribution to the objective function value. Since some constraints may be violated, constraints are handled as soft. Note that the objective function value obtained after an assignment step is highly order-dependent.

Algorithm 4 UCOP-k-means

```
1: Initialization: Start with initial cluster centroids, $\mu_j$, $j \in \{1, \ldots, k\}$.
2: Order the data objects randomly and let $R_{i'}$, $i' \in \{1, 2, \ldots, n+m\}$ be the ordered data objects.
3: repeat
4:   Assignment:
5:   for $i' = 1$ to $n + m$ do
6:     Assign the data object $R_{i'}$ to the cluster $j^*$ such that following conditions are satisfied:
       (1) $j^* = \text{argmin}_j \max_{p_{i'} \in \ell^{i'}} \|p_{i'} - \mu_j\|$ for polytopes while
           $j^* = \text{argmin}_j \|o_{i'} - \mu_j\|$ for disks, subject to:
       (2) $\nexists\, i'_1$, $i'_1 \in \{1, \ldots, i' - 1\}$, such that $\{i', i'_1\} \in ML$ and $R_{i'_1} \in C_k$
           where $k \neq j^*$,
       (3) $\nexists\, i'_2$, $i'_2 \in \{1, \ldots, i' - 1\}$, such that $\{i', i'_2\} \in CL$ and $R_{i'_2} \in C_{j^*}$,
       and let $R_{i'} \in C_{j^*}$.
7:     if $\nexists\, j^*$ satisfying the above conditions AND It is the first iteration then
8:       Go to Step 2.
9:     else if $\nexists\, j^*$ satisfying the above conditions then
10:      Go to Step 18.
11:    end if
12:  end for
13:  Update:
14:  for $j = 1$ to $k$ do
15:    Update $\mu_j$ by SM considering the data objects $R_{i'} \in C_j$, $i' \in \{1, \ldots, n+m\}$.
16:  end for
17: until Convergence is achieved.
18: return Best partition of the data objects found so far.
```

We made some modifications to solve the SSC with PC-k-means and name the modified algorithm as UPC-k-means. Starting with the initial cluster centroids obtained by using the given instance level constraints, we order the data objects randomly at the beginning of the algorithm to reduce the effect of the order-dependence and keep that ordering through the iterations similar to UCOP-k-means. We use the maximum distance between the data objects and the cluster centroids in the assignment step and the subgradient method in the update step of the algorithm. UPC-k-means terminates with the best solution found, considering the objective function of the MISOCP formulation, when the same convergence condition used in UCOP-k-means is satisfied. The details of UPC-k-means are given in Algorithm 5.

Algorithm 5 UPC-k-means

```
Initialization: Start with initial cluster centroids, $\mu_j$, $j \in \{1, \ldots, k\}$, obtained using the instance level constraints.
Order the data objects randomly and let $R_{i'}$, $i' \in \{1, 2, \ldots, n+m\}$ be the ordered data objects.
repeat
    Assignment:
    for $i' = 1$ to $n + m$ do
        Assign the data object $R_{i'}$ to the cluster $j^*$ such that the contribution to the objective function is minimized, where
            $j^* = \text{argmin}_j \alpha_{i'} (\max_{p_{i'} \in \ell^{i'}} (\|p_{i'} - \mu_j\|))^2 + \sum_{\substack{\{i', i'_1\} \in ML, \\ i'_1 \in \{1, \ldots, i'-1\} \\ R_{i'_1} \notin C_j}} c_1 + \sum_{\substack{\{i', i'_2\} \in CL, \\ i'_2 \in \{1, \ldots, i'-1\} \\ R_{i'_2} \in C_j}} c_2$
        for polytopes and
            $j^* = \text{argmin}_j \alpha_{i'} (\|o_{i'} - \mu_j\| + r_{i'})^2 + \sum_{\substack{\{i', i'_1\} \in ML, \\ i'_1 \in \{1, \ldots, i'-1\} \\ R_{i'_1} \notin C_j}} c_1 + \sum_{\substack{\{i', i'_2\} \in CL, \\ i'_2 \in \{1, \ldots, i'-1\} \\ R_{i'_2} \in C_j}} c_2$
        for disks and let $R_{i'} \in C_{j^*}$.
    end for
    Update:
    for $j = 1$ to $k$ do
        Update $\mu_j$ by SM considering the data objects $R_{i'} \in C_j$, $i' \in \{1, \ldots, n+m\}$.
    end for
until Convergence is achieved.
return Best partition of the data objects found so far.
```

2.5.2.3 UCVQE

The initialization step of the CVQE (Constrained Vector Quantization Error) algorithm is the same with that of the k-means. To improve the clustering performance, instance level constraints are taken into account in both the assignment and update steps of CVQE. Let $oi_{ML}(t)$ and $oi'_{ML}(t)$ be the indices of the first and second data objects of the t^{th} must-link constraint, respectively, and $oi_{CL}(t)$ and $oi'_{CL}(t)$ be the indices of the first and second data objects of the t^{th} cannot-link constraint, respectively. Furthermore, let $ci_{ML}(t)$ and $ci'_{ML}(t)$ denote the indices of the clusters to which $oi_{ML}(t)$ and $oi'_{ML}(t)$ are assigned, respectively, and $ci_{CL}(t)$ and $ci'_{CL}(t)$ denote the indices of the clusters to which $oi_{CL}(t)$ and $oi'_{CL}(t)$ are assigned, respectively. Also, by V_{ML} and V_{CL} we represent the sets of indices of the violated must-link and cannot-link constraints, respectively. In the assignment step of CVQE, the following objective function E is considered.

$$E = \sum_{j=1}^{k} E_j, \tag{2.10}$$

where

$$\begin{aligned} E_j = \sum_{i:x_i \in C_j} \alpha_i \left\| x_i - \mu_j \right\|^2 &+ \sum_{t \in V_{ML},\; ci_{ML}(t)=j} \left\| \mu_j - \mu_{ci'_{ML}(t)} \right\|^2 \\ &+ \sum_{t \in V_{CL},\; ci_{CL}(t)=j} \left\| \mu_j - \mu_{h(ci'_{CL}(t))} \right\|^2 . \end{aligned}$$

Here $h(j)$ returns the index of the nearest cluster to cluster j, where the distance between two clusters is measured by the distance between their centroids. Objective function E consists of two main terms; namely, a distortion term and a constraint violation term. Similar to the k-means, the distortion term is a weighted sum of squared Euclidean distances between the data objects and the centroids of the clusters they are assigned to. The constraint violation term on the other hand is computed by considering the distances between the cluster centroids. The cost of violating a must-link constraint is the distance between the cluster centroids to which the must linked data objects are assigned. The cost of violating a cannot-link constraint is the distance between the centroid of the cluster to which the cannot linked data objects are assigned and the nearest cluster centroid.

In CVQE, each pair of data objects belonging to a constraint is considered simultaneously. After checking all k^2 possible assignments, the two data objects in a constraint are assigned to the clusters that minimize the objective function E in Equation 2.10. The data objects which do not belong to any instance level constraint are assigned to the closest cluster.

In the update step of CVQE, the centroids are updated with the formula

$$\mu_j = \frac{y_j}{z_j} \tag{2.11}$$

where

$$y_j = \sum_{i:x_i \in C_j} \alpha_i x_i + \sum_{t \in V_{ML},\, ci_{ML}(t)=j} \mu_{ci'_{ML}(t)} + \sum_{t \in V_{CL},\, ci_{CL}(t)=j} \mu_{h(ci'_{CL}(t))}$$

and

$$z_j = \sum_{i:x_i \in C_j} \alpha_i + \sum_{t \in V_{ML},\, ci_{ML}(t)=j} 1 + \sum_{t \in V_{CL},\, ci_{CL}(t)=j} 1.$$

With Equation 2.11, the cluster centroid that holds the first data object belonging to a violated must-link constraint is moved towards the cluster centroid that holds the second data object of the constraint. Moreover, the cluster centroid that holds both of the data objects belonging to a violated cannot-link constraint is moved towards the nearest cluster centroid.

In the final partition found by CVQE, some of the constraints may be violated, i.e., constraints are handled as soft. Note that the assignment step is highly order-dependent. Both the order of the constraints and the order of the data objects in a constraint affect the assignments.

For the solution of the SSC problem, we made some modifications to CVQE and called this modified version as UCVQE. First of all, we order the instance level constraints randomly to reduce the effect of the order-dependence and keep that ordering through the iterations. The next modification is in the assignment step. To find the distance between a regional data object and a cluster centroid, we used the maximum distance. The updated objective function is

$$UE = \sum_{j=1}^{k} UE_j \tag{2.12}$$

where

$$UE_j = \sum_{i \in \{1,\dots,n\}: R_i \in C_j} \alpha_i (\max_{p_i \in \ell^i} \|p_i - \mu_j\|)^2 + \sum_{i \in \{n+1,\dots,n+m\}: R_i \in C_j} \alpha_i (\|o_i - \mu_j\| + r_i)^2$$
$$+ \sum_{t \in V_{ML},\ ci_{ML}(t)=j} \left\|\mu_j - \mu_{ci'_{ML}(t)}\right\|^2 + \sum_{t \in V_{CL},\ ci_{CL}(t)=j} \left\|\mu_j - \mu_{h(ci'_{CL}(t))}\right\|^2 .$$

Finally, in the update step, to be able to utilize the update function in 2.11, we require the summation $\sum_{i:x_i \in C_j} \alpha_i x_i$. Since our data objects are regions instead of points, finding the required summation is not straightforward. To resolve this issue, we first use the proposed subgradient algorithm, SM, to update the cluster centroids and then "correct" the centroids by Equation 2.11 using $\mu_j \sum_{i:x_i \in C_j} \alpha_i$ instead of $\sum_{i:x_i \in C_j} \alpha_i x_i$. The algorithm iterates until the convergence condition used in UCOP-k-means and UPC-k-means is satisfied and then returns the best solution found, in terms of the objective function of the MISOCP formulation. The details of UCVQE are given in Algorithm 6.

2.5.2.4 USeeded-k-means and UConstrained-k-means

Seeded-k-means and Constrained-k-means are originally proposed for semi-supervised clustering problems with point data objects in the existence of labeled data. The initialization steps of these algorithms are different than that of the k-means. The algorithms start with an initial grouping (partition) of the data objects that are associated with a label and the centroids of these groups are used as the initial centroids. The authors assume that at least one labeled data is available for each cluster and thus there will be exactly k groups in the initialization. Seeded-k-means does not consider the prior information after the initialization step and assigns each data object to the closest cluster. Thus the prior information is considered as soft. Different than Seeded-k-means, Constrained-k-means considers the prior information in the assignment step in addition to the initialization step. Only unlabeled data objects are reassigned while the assignments of labeled data are kept unchanged during the assignment step. Thus, Constrained-k-means considers the prior information as hard. The update steps of both algorithms are the same with that of the k-means.

Algorithm 6 UCVQE

```
Initialization: Start with initial cluster centroids, $\mu_j, j \in \{1, \ldots, k\}$.
Order the instance level constraints randomly and let $OR_i, i \in \{1, 2, \ldots, |ML| + |CL|\}$ denote the $i^{th}$ constraint in the ordered list.
repeat
    Assignment:
    Let $\pi$ and $\omega$ be the sets of indices of the polytopes and disks assigned to a cluster. Let $\pi = \{\ \}$, $\omega = \{\ \}$, $V_{ML} = \{\ \}$, and $V_{CL} = \{\ \}$.
    for $i = 1$ to $|ML| + |CL|$ do
        if $OR_i \in ML$ then
            Update $\pi$ and $\omega$ by including $R_{oi_{ML}(OR_i)}$ and $R_{oi'_{ML}(OR_i)}$ in the corresponding sets. Assign $R_{oi_{ML}(OR_i)}$ and $R_{oi'_{ML}(OR_i)}$ to the clusters $j_1^*$ and $j_2^*$, respectively, satisfying $\{j_1^*, j_2^*\} = \text{argmin}_{\{j_1, j_2\}} UE_{j_1} + UE_{j_2}$.
            if $j_1 \neq j_2$ then
                $V_{ML} = V_{ML} \cup \{OR_i\}$
            end if
        end if
        if $OR_i \in CL$ then
            Update $\pi$ and $\omega$ by including $R_{oi_{CL}(OR_i)}$ and $R_{oi'_{CL}(OR_i)}$ in the corresponding sets. Assign $R_{oi_{CL}(OR_i)}$ and $R_{oi'_{CL}(OR_i)}$ to the clusters $j_1^*$ and $j_2^*$, respectively, satisfying $\{j_1^*, j_2^*\} = \text{argmin}_{\{j_1, j_2\}} UE_{j_1} + UE_{j_2}$.
            if $j_1 = j_2$ then
                $V_{CL} = V_{CL} \cup \{OR_i\}$
            end if
        end if
    end for
    Assign the data objects that do not belong to any instance level constraint to the closest cluster in terms of the maximum distance.
    Update:
    for $j = 1$ to $k$ do
        Update $\mu_j$ by SM considering the data objects $R_i \in C_j, i \in \{1, 2, \ldots, n + m\}$.
    end for
    for $j = 1$ to $k$ do
        Correct $\mu_j$ using $\mu_j \sum_{i:x_i \in C_j} \alpha_i$ instead of $\sum_{i:x_i \in C_j} \alpha_i x_i$ in Equation 2.11.
    end for
until Convergence is achieved.
return Best partition of the data objects found so far.
```

Since the labeled data can easily be transformed into instance level constraints, these two algorithms can also be used for semi-supervised clustering problems in the existence of instance level constraints after small modifications. The first modification for both algorithms is in the initialization steps. For the instance level constraints case, the number of initial groups may not be equal to k. We can use, for example, the procedure proposed in PC-k-means [11] for the initialization of Seeded-k-means and Constrained-k-means with instance level constraints. For Constrained-k-means, we modify the assignment step in addition to the initialization step. We form chunklets by using only must-link constraints and consider each chunklet as a single data object (by this way the data objects belonging to the same chunklet will never split). Then, each chunklet is assigned to the closest cluster which is the cluster minimizing the weighted sum of squares of the distances between the cluster centroid and the data objects of the chunklet. The remaining data objects that do not belong to any chunklet are assigned to the closest 'appropriate' cluster. Note that the assignment step of Constrained-k-means for instance level constraints is order-dependent. It may not be possible to find an 'appropriate' cluster for all the data objects in all iterations. Thus we utilize the same random ordering process used in UCOP-k-means. We order the data objects that do not belong to any chunklet randomly at the beginning of the algorithm and use that ordering during the iterations. But if an 'appropriate' cluster is not found for a data object in the first iteration, we re-order those data objects randomly and re-start the assignment of them until a feasible partition for the first iteration is obtained. If an 'appropriate' cluster is not found for a data object in the following iterations, we return the best solution found so far. While Seeded-k-means for instance level constraints considers constraints as soft, Constrained k-means for instance level constraints handles them as hard constraints.

After foregoing modifications for instance level constraints, in order to use Seeded-k-means and Constrained-k-means for the SSC, we made additional changes and call these changed versions as USeeded-k-means and UConstrained-k-means, respectively. As in the previously mentioned algorithms we use the maximum distance between the data objects and the cluster centroids in the assignment steps to measure closeness and apply the proposed subgradient method for the update steps of the algorithms. The algorithms iterate until the convergence condition used in UCOP-k-means and UPC-k-means is satisfied and then return the best solution found, in terms of the objective function of the MISOCP formulation. The details of USeeded-k-means and UConstrained-k-means are given in Algorithm 7 and Algorithm 8, respectively.

Algorithm 7 USeeded-k-means

```
Initialization: Start with initial cluster centroids, $\mu_j$, $j \in \{1, \ldots, k\}$, obtained
using the instance level constraints.
repeat
    Assignment:
    for $i = 1$ to $n + m$ do
        Assign the data object $R_i$ to the cluster $j^*$ such that
        $j^* = \text{argmin}_j \max_{p_i \in \ell^i} \|p_i - \mu_j\|$ for polytopes while
        $j^* = \text{argmin}_j \|o_i - \mu_j\|$ for disks and let $R_i \in C_{j^*}$.
    end for
    Update:
    for $j = 1$ to $k$ do
        Update $\mu_j$ by SM considering the data objects $R_i \in C_j, i \in \{1, \ldots, n+m\}$.
    end for
until Convergence is achieved.
return Best partition of the data objects found so far.
```

2.5.3 Agglomerative Hiearchical Clustering based Algorithm for the SSC

In [116], the authors propose an agglomerative hierarchical clustering algorithm, AHCP, in the existence of instance level constraints. The classical agglomerative hierarchical clustering algorithms begin with each data object as a separate cluster and merge the two least dissimilar clusters in each iteration until all the data objects are in the same cluster. Dissimilarity can be measured in different ways, e.g., using the centroid method or the Ward method. In the centroid method, the dissimilarity of two clusters is measured by the distance between their centroids. For a cluster C_j with centroid μ_j and the data objects in that cluster, $y \in C_j$, let $E(C_j) = \sum_{y \in C_j} \|y - \mu_j\|$. In the Ward method, the dissimilarity of C_{j_1} and C_{j_2} is

Algorithm 8 UConstrained-k-means

```
1:  Initialization: Start with initial cluster centroids, $\mu_j$, $j \in \{1, \ldots, k\}$, obtained using the instance level constraints.
2:  Form chunklets. Let $EC_\psi$, $\psi \in \{1, 2, \ldots, \zeta\}$, be the $\psi^{th}$ chunklet and $\pi^\psi$ and $\omega^\psi$ be the set of polytopes and disks belonging to $EC_\psi$, respectively.
3:  repeat
4:      Assignment:
5:      for $\psi = 1$ to $\zeta$ do
6:          Assign chunklet $EC_\psi$ to the cluster $j^*$ such that
              $$j^* = \operatorname*{argmin}_j \sum_{i \in \pi^\psi} \alpha_i (\max_{p_i \in \ell^i} \|p_i - \mu_j\|)^2 + \sum_{i \in \omega^\psi} \alpha_i (\|o_i - \mu_j\| + r_i)^2$$
          and let $R_i \in C_{j^*}$, $i \in \pi^\psi$ or $i \in \omega^\psi$ .
7:      end for
8:      Let $R_i$, $i \in \{1, 2, \ldots, \nu\}$, be the $i^{th}$ data object that does not belonging to any $EC_\psi$, $\psi \in \{1, 2, \ldots, \zeta\}$.
9:      Order $R_i, i \in \{1, 2, \ldots, \nu\}$ randomly and let $R_{i'}, i' \in \{1, 2, \ldots, \nu\}$ be the ordered data objects.

10:     for $i' = 1$ to $\nu$ do
11:         Assign the data object $R_{i'}$ to the cluster $j^*$ such that the following conditions are satisfied:
            (1) $j^* = \operatorname{argmin}_j \max_{p_{i'} \in \ell^{i'}} \|p_{i'} - \mu_j\|$ for polytopes and $j^* = \operatorname{argmin}_j \|o_{i'} - \mu_j\|$ for disks, subject to:
            (2) $\nexists\, i'_2, i'_2 \in \{1, \ldots, i' - 1\}$, such that $\{i', i'_2\} \in CL$ and $R_{i'_2} \in C_{j^*}$,
            and let $R_{i'} \in C_{j^*}$.
12:         if $\nexists\, j^*$ satisfying the above conditions AND It is the first iteration then
13:             Go to Step 9.
14:         else if $\nexists\, j^*$ satisfying the above conditions then
15:             Go to Step 23.
16:         end if
17:     end for
18:     Update:
19:     for $j = 1$ to $k$ do
20:         Update $\mu_j$ by SM considering the data objects $R_i \in C_j$, $i \in \{1, 2, \ldots, n + m\}$.
21:     end for
22: until Convergence is achieved.
23: return Best partition of the data objects found so far.
```

defined as $E(C_{j_1} \cup C_{j_2}) - E(C_{j_1}) - E(C_{j_2})$.

AHCP starts with forming chunklets by using the must link constraints. Then, it considers each chunklet and each data object that is not placed into any chunklet as separate clusters and merge the two least dissimilar clusters until k clusters are obtained. Thus, in the final partition, all of the must link constraints are satisfied. To measure the dissimilarity of two clusters, the authors, in addition to the classical dissimilarity measures, use a violation term related to the cannot link constraints. Note that AHCP considers cannot link constraints as soft constraints although must link constraints are handled as hard constraints.

For the SSC, we modify AHCP and name it as UAHCP. Let us redefine $E(C_j)$ as

$$E(C_j) = \sum_{\substack{R_i \in C_j, \\ R_i \text{ is a polytope}}} \alpha_i (\max_{p_i \in \ell_i}(\|p_i - \mu_j\|))^2 + \sum_{\substack{R_i \in C_j, \\ R_i \text{ is a disk}}} \alpha_i (\|o_i - \mu_j\| + r_i)^2. \qquad (2.13)$$

Then, we define the dissimilarity of C_{j_1} and C_{j_2}, $\varpi(C_{j_1}, C_{j_2})$, as

$$\varpi(C_{j_1}, C_{j_2}) = E(C_{j_1} \cup C_{j_2}) - E(C_{j_1}) - E(C_{j_2}) + \sum_{\substack{R_{i_1} \in C_{j_1}, \\ R_{i_2} \in C_{j_2}, \\ \{i_1, i_2\} \in CL}} c_2 \ . \qquad (2.14)$$

Moreover, to find the centroids of the clusters obtained throughout the algorithm we implement the proposed subgradient method. The details of UAHCP are given in Algorithm 9.

2.5.4 More on Literature of Semi-supervised Clustering

We have conducted an extensive literature review on semi-supervised clustering in order to find algorithms different than the aforementioned algorithms. We have found some different semi-supervised clustering algorithms for point data objects. When we extend them for the problem considered in this chapter, the extensions turned out to be not significantly different from the already considered extensions.

In [131], authors take must-link and cannot-link constraints into consideration while clustering point data objects. To solve the problem, they propose a k-means based algorithm. The initialization and update steps of this algorithm are the same as those of

Algorithm 9 UAHCP

```
Form the chunklets. Let $EC_\psi$, $\psi \in \{1, 2, \ldots, \zeta\}$ be the $\psi^{th}$ chunklet and $\pi^\psi$ and $\omega^\psi$ be the set of polytopes and disks belonging to $EC_\psi$, respectively.
Let $R_i$, $i \in \{1, 2, \ldots, \nu\}$, be the $i^{th}$ data object that does not belong to any $EC_\psi$, $\psi \in \{1, 2, \ldots, \zeta\}$.
Start with initial clusters $C = \{C_j\}$, $j \in \{1, 2, \ldots, \zeta + \nu\}$ with centers $\mu_j$, where $\mu_j$, $j \in \{1, 2, \ldots, \zeta\}$, is found by SM considering the data objects in $EC_j$, and $\mu_j$, $j \in \{\zeta + 1, \zeta + 2, \ldots, \zeta + \nu\}$, is the geometric center of the data object $R_{j-\zeta}$.
repeat
    Merge $C_u$ and $C_v$ where
        $(u, v) = \underset{\substack{j_1 \neq j_2, \\ C_{j_1} \in C, \\ C_{j_2} \in C}}{\operatorname{argmin}} \varpi(C_{j_1}, C_{j_2}).$
    Let $C_u = C_u \bigcup C_v$ and $C = C \setminus C_v$.
until $k$ clusters remained.
return Partition $\{C_1, \ldots, C_k\}$ of the data objects.
```

the k-means. In the first assignment step, chunklets which are formed by using must-link constraints are assigned to the nearest cluster centroids and these assignments are not changed during the algorithm. The remaining data objects which do not belong to any must-link constraints are reassigned to the closest 'appropriate' clusters throughout the iterations. When we try to modify this algorithm for the solution of our problem, it resembles to UConstrained-k-means.

Soft-seeded-k-means [78] is another k-means based algorithm proposed in the literature for semi-supervised clustering of point data objects. The algorithm considers some number of labeled data as prior information. The update step of the algorithm is the same as that of the k-means. In the initialization step, centroids (obtained by taking average of data points) of initial partition of the labeled objects are used. If the number of those centroids is smaller than the number of clusters to be formed, the remaining centroids are randomly selected among the unlabeled data objects. In the assignment step, an objective function which is the sum of reassignment penalty and the distance between the data object and the cluster centroid is considered. Each data object is assigned to the cluster centroid which minimizes the objective function. When we try to modify this algorithm for the solution of our problem, it becomes very similar to UPC-k-means.

In [148], labeled data is used as prior information and a k-means based algorithm is proposed for the solution of semi-supervised clustering of point data objects. The update step of the algorithm is the same as that of the k-means. In the initialization step, authors consider both complete seed (there exists at least one labeled data object for each cluster) and incomplete seed cases. For the complete seed version, centroids of initial partition of the labeled objects are used. For the incomplete seed version, authors use two different methods to obtain centroids for which no seed (data object) is provided. In the first method, the data object which is farthest to previously obtained centroids is selected as next centroid and the procedure is repeated until the required number of centroids is obtained. In the second method, centroids are randomly selected among the unlabeled data objects. In the assignment step of the algorithm, only unlabeled data objects are reassigned to the closest centroids. After the modification of this algorithm for the solution of our problem, it resembles to UConstrained-k-means.

A k-means based algorithm is proposed for the solution of semi-supervised clustering of point data objects in the existence of only must-link constraints in [149]. The initialization and update steps of the algorithm are the same as those of the k-means. In the assignment step, chunklets are formed and they are considered as a single data object. Then, each data object (some are chunklets) is assigned to the closest cluster. The modified version of this algorithm for the solution of our problem is very similar to UConstrained-k-means.

2.6 Computational Studies

All of the algorithms are coded with MATLAB R2015a and run on a computer with Intel Core i7, 3.4 GHz processor and 8 GB RAM. To solve the SOCP problems (given in Section 2.4), SDPT3 solver of CVX [41] with MATLAB interface is utilized. For the solution of the MISOCP problems, Gurobi solver [73] with MATLAB interface is used.

To compare the performances of the proposed algorithms for the SSC, we make use of the problem instances from the literature. For each selected problem instance, the following four sets of instance level constraints are generated:

1. Only must-link constraints,
2. Only cannot-link constraints,
3. Random constraints,
4. Constraints by active learning [11].

While generating a constraint, we select two data objects. If their labels are the same, a must-link constraint, otherwise a cannot-link constraint is generated. To observe which type of instance level constraint is more informative, we created the first three sets of constraints. Pairs of data objects are picked randomly until the required number of constraints, which is changed during the computational studies to see the effects of it on the clustering performance, is achieved.

Instead of randomly generating the constraints, the fourth set of constraints is formed with the method proposed by Basu et al. [11]. The method actively selects informative constraints in two phases, namely the explore and consolidate phases. It is assumed that we can make a given number of pairwise comparisons of labels of the data objects, say Q. In the explore phase, farthest-first traversal scheme that tries to select data objects that are far from each other is utilized. Firstly, a data object is chosen at random and placed into the first neighborhood. At the beginning of each step, the data object which is farthest from the data objects placed into the neighborhoods is found (i.e., maximizing the minimum distance). Then, the new data object is placed in an existing neighborhood if a must-link constraint can be generated after checking the labels of the new data object and a data object from each neighborhood. Otherwise, a new neighborhood is formed and the new data object is placed into it. The explore phase proceeds until k neighborhoods are generated or the maximum number of pairwise comparisons, Q, is made. If there exists extra pairwise comparisons at the end of the explore phase, the method continues with the consolidate phase. At the beginning of this phase, the centroid of each of the k neighborhoods is found. Then, a data object that is not placed into a neighborhood is randomly selected and the distance between this data object and the centroids of the neighborhoods are calculated. Starting with the neighborhood with the closest centroid, pairwise comparisons of labels of the new data object and a data object from the neighborhood is made until the new data object is placed into a neighborhood, i.e., a must-link constraint is generated. The consolidate phase continues until Q pairwise comparisons are made. Note that the distance between a (regional) data object and another (regional) data object or a cluster centroid is calculated by the maximum distance between them and the centroids of neighborhoods are found by the subgradient method proposed in Section 2.4.2.

The choice of the violation costs, c_1 and c_2, is an important aspect of semi-supervised clustering. If these costs are set to 0, then the algorithms act as unsupervised clustering algorithms. If they are set to a positive value, a tradeoff between constraints violations and distances between the data objects and the centroids is included into the problem. Constraints are handled as soft constraints if the violation costs are not chosen too high. When the violation costs are set to infinity (or to a large enough number), constraints become hard constraints. c_1 and c_2 can be defined by the user based on the degree of importance of the constraints. In [11], the authors set the values of c_1 and c_2 to a constant which is the average distance between pairs of data objects in the problem instance. In our experimental studies, for a given problem instance, we fixed c_1 and c_2 to the average maximum distance between pairs of data objects.

Similar to the traditional k-means, the final partition of the data objects by all of the methods mentioned in Section 2.5.2 is highly depended on the initial cluster centroids. Thus, the algorithms may be initialized with different sets of initial centroids and the best solution among them (in terms of the objective function value) may be returned to the user or a method can be developed to find good initial centroids. In our experimental studies, the following three different initialization procedures are used:

1. Random initialization,

2. Initialization by the method in [11],
3. Initialization by means of the agglomerative hierarchical clustering.

In the first procedure, the initial cluster centroids are randomly generated from the convex hull of the data objects. As the solution returned is highly dependent on the set of initial cluster centroids, we make 20 replications and the best solution found among these 20 replications is taken as the solution returned. Note that with this initialization procedure, USeeded-k-means is just an extension of the traditional k-means for regional data objects. The instance level constraints are not utilized in any step of USeeded-k-means and thus it behaves as an unsupervised clustering algorithm for regional data objects.

We apply the initialization scheme proposed in [11] as the second initialization procedure. The details of the procedure are given in Section 2.5.2.2. In this procedure, ζ chunklets are formed using the must-link constraints. If $\zeta \geq k$, then the centroids of the largest k chunklets, which are found by SM, are used as the initial centroids. If $\zeta < k$, then the centroids of all the chunklets are utilized as first ζ initial centroids. Next, a data object which has a cannot-link constraint with every chunklet is searched and if one is found, its geometric center is set as the $(\zeta + 1)^{th}$ cluster centroid. The remaining centroids are initialized by random perturbations of the global centroid. If at least one of the cluster centroids is required to be generated by random perturbation, the procedure is repeated 20 times (i.e., 20 replications) to obtain different sets of cluster centroids. Otherwise, there is only one set of initial centroids.

UAHCP proposed in Section 2.5.3 is a myopic algorithm. Merging two clusters that minimizes the dissimilarity measure in Equation 2.14 in an iteration may result in facing more constraint violation costs in the following iterations. Thus, the partition found by UAHCP may not be a good partition considering the objective function value of the MISOCP formulation. As a third initialization procedure, to improve upon the solution found by UAHCP, we give the cluster centroids found by UAHCP to the algorithms proposed in Section 2.5.2 with the goal of starting with a good set of initial centroids. By this procedure, only one set of initial centroids is provided to the algorithms (i.e., one replication).

2.6.1 Performance Measures

We use six performance measures to evaluate the algorithms. Given a problem instance and a fixed number of constraints to be generated (and the method that they will be generated by), we generate 20 sets of constraints and find the (best) partition of the data objects by the proposed algorithms for each set of constraints (and obtain the values of the performance measures). The averages of the values of the 20 performance measures are then reported.

Our performance measures are global and local objective function value, solution time, rand index, pairwise F-measure, and mutual information. Since we have an optimization problem, objective function values as performance measures came to our minds naturally. But we did not restrict ourselves with those since objective function values are difficult for direct interpretation. Although by looking objective function values we can see the effects of constraint generation type, initialization procedure and the number of constraints on the solution quality, they do not provide direct information on agreement between found partition and desired partition unless we do not know cost of violating a single constraint and the weighted sum of squared maximum Euclidean distances in the optimal solution. Thus, we have rand index, pairwise F-measure, and mutual information which are widely used in the literature to quantify agreement. Each performance measure has own advantages and an algorithm performing best in one measure may not be the best for another. Therefore, having many performance measures instead of one is beneficial for preventing bias against the algorithms.

As seen in the MISOCP formulation, the objective function is the total of the violation costs of the unsatisfied instance level constraints and the weighted sum of squared maximum Euclidean distances. We name this value as the local objective function value. This way of calculating the objective function value may be misleading in certain comparisons because of the differences between the algorithms in handling the given instance level constraints. UCOP-k-means and UConstrained-k-means handle the given constraints as hard while the MISOCP formulation, UPC-k-means, UCVQE, and USeeded-k-means consider them as soft. Also, UAHCP considers the given must-link constraints as hard and the given cannot-link constraints as soft. Therefore, the partition found by the algorithms for the given instance level constraints should be compared with the desired/correct partition of the data objects. In addition to the violation of the given constraints, violation of the constraints obtained by the pairwise comparisons of all the data objects should be included in the calculation of the objective function value of the partition obtained. We call this value as the global objective function value. The lower the objective function values (both local and global), the better the performance.

The rand index is generally used to measure the agreement between the found partition and the desired partition of the data objects [116, 145]. Higher the rand index, the better the performance. Let ρ_1 be the number of pairs of data objects placed into the same cluster in the found partition while they are in the same cluster at the desired partition, ρ_2 be the number of pairs of data objects placed into different clusters in the found partition while they are in different clusters at the desired partition. With the total number of $(n+m)(n+m-1)/2$ pairs of data objects in a problem instance, rand index is calculated as

$$Rand = \frac{\rho_1 + \rho_2}{(n+m)(n+m-1)/2}. \tag{2.15}$$

Pairwise F-measure is another performance evaluation metric used in the literature [11]. It is the harmonic mean of pairwise precision and recall. Higher the pairwise F-measure, the better the performance. Let η be the number of pairs of data objects placed into the same cluster in the found partition and ϱ be the number of pairs of data objects belonging to the same cluster in the desired partition. Then, the pairwise precision, recall, and F-measure are calculated as

$$Precision = \frac{\rho_1}{\eta}, \tag{2.16}$$

$$Recall = \frac{\rho_1}{\varrho}, \tag{2.17}$$

$$F - measure = \frac{2(Precision)(Recall)}{Precision + Recall}. \tag{2.18}$$

Similar to the rand index, the mutual information determines the amount of similarity between the found partition and the desired partition of the data objects [11, 134]. Thus, the higher the mutual information, the better the performance. Let $C = \{C_1, C_2, \ldots, C_k\}$ be the found partition, $\mathit{\Gamma} = \{\mathit{\Gamma}_1, \mathit{\Gamma}_2, \ldots, \mathit{\Gamma}_k\}$ be the correct/desired partition, $\varsigma_{j_f}^{j_c}$ be the number of data objects clustered into C_{j_f} while they actually belong to $\mathit{\Gamma}_{j_c}$. Then, the mutual information is calculated as

$$Mutual = \frac{1}{(n+m)} \sum_{j_f=1}^{k} \sum_{j_c=1}^{k} \varsigma_{j_f}^{j_c} \frac{log(\frac{\varsigma_{j_f}^{j_c}(n+m)}{\sum_{\iota=1}^{k} \varsigma_{\iota}^{j_c} \sum_{\iota=1}^{k} \varsigma_{j_f}^{\iota}})}{log(k^2)}. \tag{2.19}$$

The maximum possible values of the rand index, pairwise F-measure, and mutual information are $1, 1$, and 0.5, respectively. Note that the mutual information can take values less than 0.5 even when the found partition is exactly the same as the desired partition.

2.6.2 Computational Results

2.6.2.1 Artificial Datasets

To compare the performances of the proposed algorithms for the SSC, we make use of the datasets generated in [54]. In that paper, the authors consider a multi-facility location problem with polygonal demand regions that aims to minimize the weighted sum of squared maximum Euclidean distances between the facilities and the demand regions. The problem considered is equivalent to the unsupervised clustering problem with polygonal uncertainty. We selected six problem instances with rectangular data objects whose sides are parallel to the standard coordinate axes used in [54], namely problem instances with $n = 10$ and $k = 2$, $n = 10$ and $k = 3$, $n = 20$ and $k = 2$, $n = 20$ and $k = 3$, $n = 50$ and $k = 3$, $n = 100$ and $k = 5$, and $n = 200$ and $k = 5$, where n is the number of rectangular data objects and k is the number of clusters to be formed.

While constructing a rectangular region, the authors first randomly generate the coordinates of the southern west corner and the side lengths of the region from the discrete uniform distributions with specified parameters. Then, they construct the other corners of the region by using the coordinates of southern west corner and the side lengths. The maximum values used for the discrete uniform distributions to generate

Table 2.3: Parameters for rectangular region genaration

Number of regions	Maximum value for x and y coordinate of southern west corners	Maximum value for side lengths
10	100	10
20	150	10
50	200	8
100	300	8
200	500	8

the problem instances are provided in Table 2.3. For all distributions 1 is used as the minimum value. The details of the problem instances (data, objective function values, and the assignments of the demand regions to the facilities for the best (or optimal) solutions) can be found at the website "http://tol.ie.metu.edu.tr/test-instances" (date accessed: April 20^{th}, 2019).

The best solutions (assignments of the demand regions/data objects to the facilities/ clusters) found for the problem instances by the methods proposed in [54] are utilized as a priori information for the SSC problem. Note that the best solutions given are not necessarily the optimal solutions of the considered facility location problem. An example problem instance with $n = 50$ and $k = 3$, and its solution utilized as a priori information is given in Figure 2.1.

As it is noted, the MISOCP formulation provided in Section 2.5.1 is weak mainly due to the presence of the big-M's in the formulation. None of the large size problem instances (where $n = 100$ and $n = 200$) is solved within reasonable times with the MISOCP formulation. Even the medium size problem instance, the instance with $n = 50$ and $k = 3$, is not solved to optimality within an hour by the MISOCP formulation. The optimality gap for this problem instance with 5 actively selected constraints is 54.3% after 21 hours of computation. To show the usability of the MISOCP formulation, we solve the small size problem instances (where $n = 10$ and $n = 20$). The solution times for these instances are reported in Table 2.4. $P_{n,k,nc}$ in the table represents a problem instance with n rectangular data objects, k clusters and nc instance level constraints generated by the active learning scheme described in Section 3.5. To

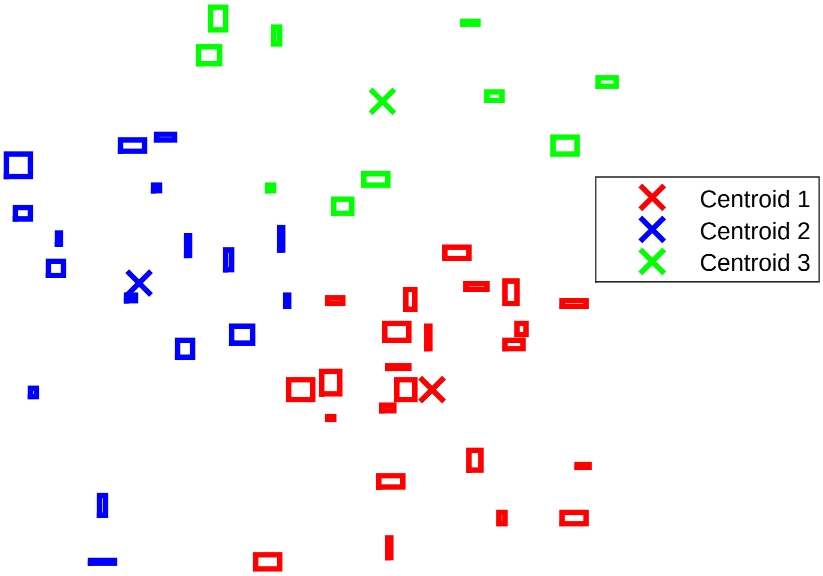


Figure 2.1: An example problem instance with $n = 50$ and $k = 3$, and its solution

initialize the k-means based algorithms, we utilize the second initialization procedure mentioned in Section 3.5. Results in Table 2.4 indicate that even though the MISOCP formulation solves small size instances to optimality in reasonable times (around 1 minute), it is the slowest solution approach for all the problem instances given in the table. To see which factors, namely the number of data objects, the number of clusters, and the number of instance level constraints, have significant effect on the solution times, we analyze the data in Table 2.4 as a factorial design. Using a 10% significance level, the solution time of all the algorithms increases with the increase in the number of data objects in the problem instance. Also, the solution time decreases as the number of instance level constraints provided increases. The effect of the number of clusters depends on the algorithm. While solution time increases when the number of clusters increases for the MISOCP formulation, UPC-k-means, and UAHCP, the effect of the number of clusters on solution time is not statistically significant for UCOP-k-means, UCVQE, USeeded-k-means, and UConstrained-k-means.

Table 2.4: Solution time (in seconds) for small size instances with actively generated constraints

Problem instance	MISOCP	UCOP-k-means*	UPC-k-means*	UCVQE*	USeeded-k-means*	UConstrained-k-means*	UAHCP
$P_{10,2,5}$	8.232	0.048	0.038	0.023	0.039	0.041	0.287
$P_{10,2,10}$	4.470	0.040	0.035	0.017	0.034	0.036	0.050
$P_{10,2,20}$	4.202	0.041	0.035	0.016	0.034	0.035	0.018
$P_{10,2,40}$	4.197	0.041	0.035	0.016	0.034	0.035	0.018
$P_{10,3,5}$	27.837	0.047	0.041	0.018	0.039	0.043	0.379
$P_{10,3,10}$	18.908	0.041	0.033	0.019	0.032	0.034	0.084
$P_{10,3,20}$	14.692	0.040	0.032	0.015	0.031	0.033	0.016
$P_{10,3,40}$	15.778	0.038	0.033	0.015	0.031	0.033	0.016
$P_{20,2,10}$	71.335	0.075	0.064	0.030	0.063	0.070	1.148
$P_{20,2,20}$	20.211	0.075	0.065	0.030	0.063	0.066	0.116
$P_{20,2,50}$	19.582	0.075	0.064	0.029	0.063	0.066	0.034
$P_{20,2,100}$	20.075	0.075	0.064	0.029	0.063	0.066	0.033
$P_{20,3,10}$	67.184	0.080	0.069	0.032	0.081	0.073	1.348
$P_{20,3,20}$	65.040	0.078	0.068	0.031	0.066	0.069	0.170
$P_{20,3,50}$	47.693	0.078	0.068	0.031	0.066	0.069	0.035
$P_{20,3,100}$	47.682	0.078	0.068	0.030	0.066	0.069	0.035

* Algorithm is initialized by the second initialization procedure given in Section 3.5.

In Table 2.5, we provide the values of the performance measures of the algorithms for a selected problem instance, $P_{20,2,10}$, where the constraints are generated by the active learning scheme and the k-means based algorithms are initialized by the second initialization procedure mentioned in Section 3.5. The MISOCP formulation is the best solution approach since the partition found by it is the same with the desired partition. This is due to the fact that the desired partition given for this instance is the optimal solution of the unsupervised clustering problem considered in [54]. Note that the desired partitions for the larger problem instances are not necessarily the optimal solutions of the unsupervised clustering problem considered in [54]. UAHCP is the second best solution approach. The worst approach is USeeded-k-means. This is expected as USeeded-k-means does not consider instance level constraints after the initialization step. The performances of the remaining algorithms are similar. Although the best performer is the MISOCP formulation, we do not provide the results

Table 2.5: Performance measures (percent deviation of global and local objective function value from the best global and local value found (*ObjG* and *ObjL*), rand index (*Rand*), pairwise F-measure (*FM*), and mutual information (*MI*)) for $P_{20,2,10}$ where constraints are generated actively

Algorithms	ObjG	ObjL	Rand	FM	MI
MISOCP	0.000	0.00	1.000	1.000	0.500
UCOP-k-means*	265.262	16.64	0.944	0.941	0.439
UPC-k-means*	233.351	11.03	0.951	0.949	0.444
UCVQE*	255.883	11.03	0.946	0.944	0.438
USeeded-k-means*	278.428	19.05	0.941	0.938	0.432
UConstrained-k-means*	233.351	11.03	0.951	0.949	0.444
UAHCP	218.731	16.73	0.956	0.953	0.448

* Algorithm is initialized by the second initialization procedure given in Section 3.5.

of this approach for larger instances due to its time ineffectiveness.

In Figure 2.2, we compare all the algorithms except the MISOCP formulation for a selected problem instance, problem instance with 200 rectangular data objects and 5 centroids, where the instance level constraints are generated by the active learning procedure. Also, all three initialization methods are utilized for the k-means based algorithms. In the figure legend, $Algorithm/init$ represents the results for the specified algorithm initialized by the $init^{th}$ initialization method. As it can be seen from the figure, all of the performance measures tend to improve as the number of instance level constraints increases for all (or most of) the algorithms (and for all type of initialization methods). Similar results are observed for the other problem instances as well. Note that, however, the improvements obtained by the random initializations is not as large as the improvements obtained by the other initializations. Therefore we do not report the results for random initializations in the rest of the study.

In Tables 2.6, 2.7, and 2.8, we report the values of the performance measures of the algorithms for the problem instance with 100 rectangular data objects and 5 centroids. In these tables, $H_{const,init,nc}$ represents that the problem instance is solved with nc instance level constraints generated by $const^{th}$ constraint generation method and the

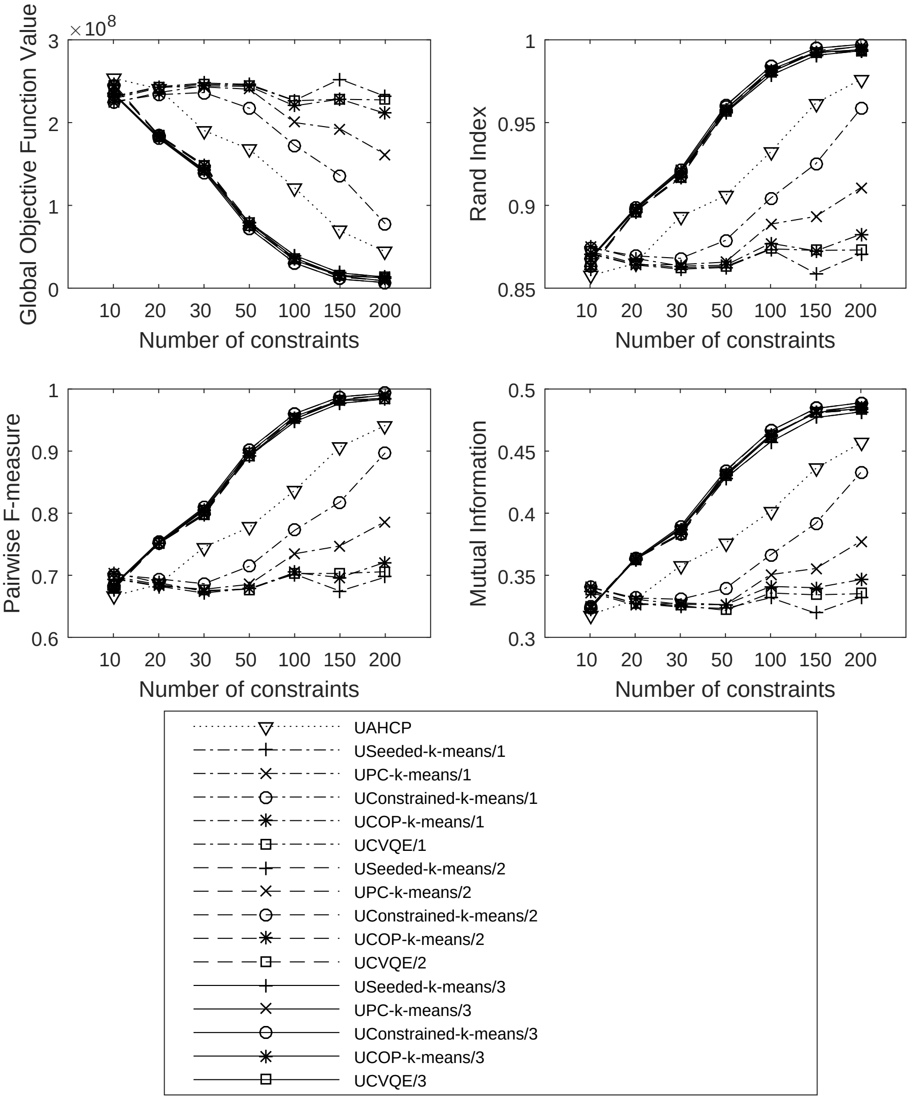


Figure 2.2: Performance measures (global objective function value, rand index, pairwise F-measure and mutual information) for the problem instance with 200 rectangular data objects and 5 centroids where constraints are generated actively

k-means based algorithms are initialized with the $init^{th}$ initialization method. Note that when we provide only cannot-link constraints to the problem, the initialization by the method in [11] is the same with the random initialization. Thus, for the cannot-link constraints case, we only report the results with the initialization by means of the agglomerative hierarchical clustering.

In Table 2.6, percent deviations of global and local objective function values are provided. Percent deviations of global (local) objective function values in each row are calculated by using the minimum global (local) objective function value found in the corresponding row. The best value in each row for global and local objective function values are highlighted in the table. Based on the values in Table 2.6, the following conclusions can be drawn.

- When we consider the percent deviations of global objective function values, UConstrained-k-means is the best algorithm when either only must-link constraints are provided and the k-means based algorithms are initialized by means of the agglomerative hierarchical clustering, or the constraints are generated actively and either the method in [11] or the agglomerative hierarchical clustering is used to initialize the k-means based algorithms. For the remaining combinations of constraint generation and initialization procedures, UAHCP performs better than the other algorithms.
- Considering deviations of local objective function values, it can be seen that UAHCP outperforms the others when the method in [11] is utilized for the initialization of the k-means based algorithms and either constraints are generated randomly or only must-link constraints are available. When only cannot-link constraints are provided and the agglomerative hierarchical clustering is used for the initialization of the k-means based algorithms, UPC-kmeans beats the other algorithms. UConstrained-k-means results in the best performance for the remaining combinations of constraint generation and initialization procedures.

Row based percent deviations of global and local objective function values (as calculated in Table 2.6) are useful in determining which algorithm is the best for a given combination of constraint generation and initialization procedures. In Table 2.7, we provide the percent deviations of global (local) objective function values from the best global (local) objective function value found for the problem instance among all combinations of constraint generation and initialization procedures. Note that since an initialization procedure is not required for UAHCP, the blocks for different initialization procedures, for a given instance level constraint generation type, contain the same values. In each column, for each number of instance level constraints, the best values are highlighted in the table. By this way we can decide on which combination of constraint generation and initialization procedures results in better performance for a given algorithm. The following results are inferred from Table 2.7.

- In terms of the percent deviation of global objective function value, the best combination is to generate instance level constraints by active learning and to initialize the algorithm by the method in [11] for UCOP-k-means and USeeded-k-means. For UPC-k-means, providing only must-link constraints and initializing the algorithm with the agglomerative hierarchical clustering or generating constraints actively and initializing the algorithm by the method in [11] are better than the other combinations of constraint generation and initialization procedures. Obtaining constraints by active learning is the best constraint generation procedure for UCVQE. When the active learning is used for the algorithm, both of the initialization procedures (the method in [11] and the agglomerative hierarchical clustering) are good choices. For UConstrained-k-means, providing only must-link constraints and utilizing the agglomerative hierarchical clustering to initialize the algorithm outperforms the other combinations of constraint generation and initialization procedures.
- According to the percent deviations of local objective function values, providing only must-link constraints and using agglomerative hierarchical clustering in initialization is the best combination for UCOP-k-means. For UPC-k-means, USeeded-k-means, and UConstrained-k-means, the best combination is to generate instance level constraints by active learning and to initialize the algorithm by the method in [11]. Like in the percent deviations of global objective function values, obtaining constraints by active learning is the best constraint generation procedure and using the method in [11] or agglomerative hierarchical clustering in initialization are good alternatives for UCVQE.
- For UAHCP, providing only must-link constraints is the best way of improving the performance in terms of the percent deviation of both global and local objective function values.
- Must-link constraints and the actively generated constraints are more informative than cannot-link constraints and randomly generated constraints.

- Although both initialization procedures have similar performances, initialization by the method in [11] is slightly better than initialization by agglomerative hierarchical clustering.

Rand index, pairwise F-measure, and mutual information values are reported in Table 2.8. Similar to Table 2.7, the blocks for different initialization procedures, for a given instance level constraint generation type, contain the same values for UAHCP since the algorithm does not require an initialization procedure. In each row, the best value for each performance measure is highlighted to see which algorithm is better for a given constraint generation and initialization combination. Also, in each column, for each number of instance level constraints, the best values are boldfaced to determine which constraint generation and initialization combination leads to better performance for a given algorithm. The following results are deduced from the table.

- UConstrained-k-means and UAHCP perform better than the other algorithms in terms of all performance measures. When we investigate each combination of constraint generation and initialization procedures, it can be seen that UConstrained-k-means is the best algorithm when instance level constraints are generated actively (selecting method in [11] or agglomerative hierarchical clustering as the initialization procedure makes no difference), or only must-link constraints are provided and the algorithm is initialized by agglomerative hierarchical clustering. For the remaining combinations of constraint generation and initialization procedures, UAHCP is better than UConstrained-k-means.
- For UCOP-k-means and USeeded-k-means, the best combination of constraint generation and initialization procedures is obtained by generating constraints actively and initializing the algorithms with the method in [11] according to all of the performance measures.
- For UPC-k-means, providing only-must link constraints and initializing the algorithm by agglomerative hierarchical clustering, or generating constraints by the active learning procedure and initializing the algorithm by the method in [11] result in better performance in terms of all performance measures.
- For UCVQE, the best constraint generation procedure is active learning. The

initialization procedures (method in [11] and agglomerative hierarchical clustering) do not make much difference for actively learned constraints.

- Generating only must-link constraints and using agglomerative hierarchical clustering based initialization is the best constraint generation and initialization combination for UConstrained-k-means considering all performance measures.
- Providing only must-link constraints to UAHCP outdoes other constraint generation alternatives in all performance measures.

According to solution time, USeeded-k-means is the best algorithm for all combinations of constraint generation and initialization procedures. But it should be noted that all of the algorithms except UAHCP solve a problem instance in less than a second. Thus, in terms of the solution time, all k-means based algorithms perform good. UAHCP's solution times are also acceptable since its solution times are around 2 minutes for a problem instance with any combination of constraint generation and initialization procedures.

Table 2.6: Performance measures (percent deviation of global and local objective function value from the best global and local value found in the corresponding row (*ObjG* and *ObjL*)) for the problem instance with 100 rectangular data objects and 5 centroids

Problem	UCOP-k-means		UPC-k-means		UCVQE		USeeded-k-means		UConstrained-k-means		UAHCP	
instance	ObjG	ObjL	ObjG	ObjL	ObjG	ObjL	ObjG	ObjL	ObjG	ObjL	ObjG	ObjL
$H_{1,2,10}$	61.86	39.86	63.01	38.70	64.08	57.12	63.73	52.28	61.30	34.82	0.00	0.00
$H_{1,2,30}$	151.63	90.14	151.06	76.06	162.89	140.16	156.53	133.99	138.09	64.25	0.00	0.00
$H_{1,2,50}$	183.35	77.47	164.83	64.20	183.27	112.30	183.65	112.14	150.19	43.69	0.00	0.00
$H_{1,2,80}$	201.08	56.64	127.04	29.55	184.39	68.77	184.32	70.29	124.77	19.54	0.00	0.00
$H_{1,2,100}$	111.30	17.57	64.89	10.98	95.63	24.84	95.63	24.84	0.00	1.57	59.62	0.00
$H_{1,3,10}$	4.79	1.11	2.54	1.18	3.23	6.85	3.32	7.23	0.00	0.00	2.02	15.29
$H_{1,3,30}$	55.85	12.09	34.11	6.20	43.71	27.67	46.38	28.91	0.00	0.00	52.67	8.76
$H_{1,3,50}$	97.37	21.77	57.32	7.32	118.96	40.69	102.39	36.19	0.00	0.00	90.21	3.92
$H_{1,3,80}$	155.01	18.73	143.18	10.21	165.08	23.62	177.93	26.70	0.00	0.00	143.22	1.61
$H_{1,3,100}$	180.75	9.06	73.34	2.82	186.97	18.58	186.97	18.58	0.00	0.00	229.54	1.64
$H_{2,3,10}$	11.01	0.65	10.51	0.00	9.60	3.53	10.99	4.99	10.94	0.27	0.00	55.37
$H_{2,3,30}$	26.16	0.41	26.31	0.00	25.10	14.93	27.60	13.89	27.34	0.44	0.00	36.53
$H_{2,3,50}$	21.84	0.97	19.07	0.00	19.00	17.55	21.93	19.81	21.13	1.07	0.00	24.24
$H_{2,3,80}$	29.61	0.00	27.50	1.36	31.35	29.33	32.65	29.65	33.15	0.84	0.00	19.51
$H_{2,3,100}$	32.06	3.10	21.29	0.00	32.73	32.26	34.50	33.91	37.69	3.91	0.00	25.43
$H_{3,2,10}$	1.05	0.00	1.29	0.55	0.00	2.63	1.99	3.19	1.79	0.36	26.72	6.72
$H_{3,2,30}$	48.23	21.35	49.27	17.95	47.43	38.25	46.48	32.55	45.92	15.53	0.00	0.00
$H_{3,2,50}$	73.15	33.69	75.67	30.59	80.09	65.87	74.14	60.50	76.33	26.06	0.00	0.00
$H_{3,2,80}$	126.49	50.52	133.84	45.88	128.33	104.02	129.68	101.28	127.40	37.37	0.00	0.00
$H_{3,2,100}$	157.50	81.38	162.27	79.63	147.13	145.66	155.25	152.31	144.88	63.75	0.00	0.00
$H_{3,3,10}$	8.23	0.38	8.09	0.55	5.04	4.02	7.46	6.16	7.79	0.00	0.00	25.76
$H_{3,3,30}$	14.95	1.19	14.51	0.00	16.51	16.05	14.48	12.15	15.76	0.21	0.00	17.92
$H_{3,3,50}$	28.25	5.37	22.35	2.19	23.50	22.58	25.46	26.31	19.55	0.00	0.00	17.16
$H_{3,3,80}$	36.54	7.25	33.51	4.58	48.61	50.90	33.38	30.00	21.43	0.00	0.00	15.42
$H_{3,3,100}$	24.67	9.91	17.81	9.96	42.57	60.95	19.23	37.43	1.59	0.00	0.00	10.33
$H_{4,2,10}$	1.40	0.16	3.07	0.00	1.67	4.29	1.83	4.01	0.00	6.30	34.60	23.21
$H_{4,2,30}$	7.49	14.93	2.21	1.72	8.78	7.88	7.48	7.98	0.00	0.00	70.79	17.33
$H_{4,2,50}$	16.63	28.47	17.41	8.25	20.77	11.68	22.31	12.70	0.00	0.00	130.80	9.57
$H_{4,2,80}$	57.72	44.68	9.98	1.49	25.59	5.47	29.04	6.60	0.00	0.00	215.78	5.84
$H_{4,2,100}$	20.28	16.24	25.34	3.65	35.83	4.69	35.83	4.69	0.00	0.00	139.20	3.90
$H_{4,3,10}$	30.25	3.58	28.24	0.00	26.59	1.23	28.55	3.88	26.99	0.08	0.00	26.94
$H_{4,3,30}$	11.18	12.63	6.02	4.27	6.03	9.77	10.12	14.18	0.00	0.00	51.97	14.66
$H_{4,3,50}$	22.16	29.56	21.69	9.58	18.99	10.64	22.05	12.68	0.00	0.00	128.07	9.47
$H_{4,3,80}$	29.63	51.93	13.34	3.10	22.17	5.05	29.04	6.60	0.00	0.00	215.78	5.84
$H_{4,3,100}$	23.09	22.89	25.75	3.14	35.83	4.69	35.83	4.69	0.00	0.00	139.20	3.90

Table 2.7: Performance measures (percent deviation of global and local objective function value from the best global and local value found for the problem instance (*ObjG'* and *ObjL'*)) for the problem instance with 100 rectangular data objects and 5 centroids

Problem	UCOP-k-means		UPC-k-means		UCVQE		USeeded-k-means		UConstrained-k-means		UAHCP	
instance	ObjG'	ObjL'	ObjG'	ObjL'	ObjG'	ObjL'	ObjG'	ObjL'	ObjG'	ObjL'	ObjG'	ObjL'
$H_{1,2,10}$	4334.12	123.04	4365.86	121.19	4395.09	150.56	4385.42	142.84	4318.89	115.01	2639.55	59.47
$H_{1,2,30}$	3894.08	145.59	3884.95	127.40	4072.86	210.20	3971.82	202.22	3679.11	112.15	1487.27	29.16
$H_{1,2,50}$	2493.50	101.48	2323.98	86.42	2492.74	141.02	2496.25	140.84	2190.03	63.13	815.30	13.53
$H_{1,2,80}$	1086.75	63.16	794.91	34.95	1020.95	75.80	1020.69	77.38	785.97	24.52	294.16	4.16
$H_{1,2,100}$	336.22	20.32	240.41	13.58	303.87	27.77	303.87	27.77	106.44	3.95	229.54	2.34
$H_{1,3,10}$	2714.02	39.86	2653.41	39.95	2671.89	47.80	2674.46	48.32	2585.28	38.32	2639.55	59.47
$H_{1,3,30}$	1520.30	33.11	1294.33	26.12	1394.14	51.61	1421.85	53.09	939.67	18.76	1487.27	29.16
$H_{1,3,50}$	849.74	33.03	657.00	17.25	953.63	53.70	873.87	48.78	381.19	9.25	815.30	13.53
$H_{1,3,80}$	313.26	21.72	294.10	12.99	329.58	26.73	350.40	29.89	62.06	2.52	294.16	4.16
$H_{1,3,100}$	180.75	9.81	73.34	3.53	186.97	19.40	186.97	19.40	0.00	0.69	229.54	2.34
$H_{2,3,10}$	4963.71	74.44	4941.06	73.31	4899.59	79.44	4962.75	81.95	4960.57	73.78	4461.53	169.28
$H_{2,3,30}$	4618.83	81.84	4624.63	81.10	4579.23	108.14	4672.89	106.26	4663.06	81.89	3640.46	147.25
$H_{2,3,50}$	4089.58	89.05	3994.37	87.22	3991.89	120.08	4092.65	124.31	4065.26	89.24	3338.54	132.61
$H_{2,3,80}$	3685.12	96.03	3623.71	98.69	3736.04	153.53	3773.87	154.15	3788.73	97.68	2820.47	134.28
$H_{2,3,100}$	3323.99	101.21	3044.76	95.15	3341.46	158.10	3387.25	161.33	3469.97	102.79	2492.77	144.78
$H_{3,2,10}$	2956.76	95.99	2963.97	97.06	2925.10	101.14	2985.25	102.24	2979.21	96.70	3733.42	109.17
$H_{3,2,30}$	4295.19	128.59	4325.99	122.17	4271.40	160.43	4243.29	149.69	4226.62	117.62	2865.15	88.37
$H_{3,2,50}$	3987.53	130.62	4046.86	125.28	4151.34	186.15	4010.86	176.87	4062.58	117.46	2260.66	72.51
$H_{3,2,80}$	4317.96	166.31	4461.22	158.09	4353.85	260.96	4380.01	256.10	4335.65	143.04	1850.58	76.92
$H_{3,2,100}$	4524.59	203.20	4610.29	200.28	4338.51	310.65	4484.25	321.77	4297.94	173.73	1695.99	67.16
$H_{3,3,10}$	4048.73	66.95	4043.47	67.23	3926.79	73.01	4019.24	76.57	4031.98	66.31	3733.42	109.17
$H_{3,3,30}$	3308.55	61.65	3295.29	59.74	3354.57	85.38	3294.48	79.16	3332.46	60.08	2865.15	88.37
$H_{3,3,50}$	2927.59	55.15	2788.33	50.47	2815.31	80.49	2861.67	85.97	2722.24	47.24	2260.66	72.51
$H_{3,3,80}$	2563.31	64.40	2504.26	60.31	2798.71	131.30	2501.67	99.27	2268.50	53.28	1850.58	76.92
$H_{3,3,100}$	2139.01	66.53	2015.94	66.60	2460.46	143.86	2041.38	108.23	1724.60	51.52	1695.99	67.16
$H_{4,2,10}$	2888.58	86.35	2937.78	86.05	2896.70	94.03	2901.33	93.51	2847.38	97.77	3867.08	129.23
$H_{4,2,30}$	1172.20	32.87	1109.75	17.59	1187.50	24.72	1172.13	24.83	1083.59	15.60	1921.42	35.64
$H_{4,2,50}$	656.71	35.09	661.76	13.83	683.56	17.43	693.59	18.50	548.82	5.15	1397.46	15.21
$H_{4,2,80}$	395.03	46.57	245.19	2.82	294.19	6.85	305.02	7.99	213.87	1.31	891.12	7.22
$H_{4,2,100}$	126.83	16.24	136.37	3.65	156.15	4.69	156.15	4.69	88.58	0.00	351.09	3.90
$H_{4,3,10}$	5067.26	87.04	4987.52	80.57	4921.79	82.80	4999.68	87.59	4937.99	80.71	3867.08	129.23
$H_{4,3,30}$	1378.76	33.25	1310.19	23.35	1310.29	29.86	1364.74	35.07	1230.12	18.30	1921.42	35.64
$H_{4,3,50}$	702.05	36.36	698.98	15.33	681.29	16.45	701.35	18.59	556.57	5.25	1397.46	15.21
$H_{4,3,80}$	306.86	53.91	255.75	4.44	283.44	6.42	305.02	7.99	213.87	1.31	891.12	7.22
$H_{4,3,100}$	132.13	22.89	137.15	3.14	156.15	4.69	156.15	4.69	88.58	0.00	351.09	3.90

Table 2.8: Performance measures (rand index (*Rand*), F-measure (*FM*), and mutual information (*MI*)) for the problem instance with 100 rectangular data objects and 5 centroids

Problem	UCOP-k-means			UPC-k-means			UCVQE			USeeded-k-means			UConstrained-k-means			UAHCP		
instance	Rand	FM	MI	Rand	FM	MI	Rand	FM	MI	Rand	FM	MI	Rand	FM	MI	Rand	FM	MI
$H_{1,2,10}$	0.83	0.61	0.31	0.83	0.61	0.31	0.83	0.61	0.31	0.83	0.61	0.31	0.83	0.62	0.31	**0.90**	**0.75**	**0.37**
$H_{1,2,30}$	0.85	0.64	0.33	0.85	0.65	0.33	0.84	0.63	0.32	0.84	0.64	0.33	0.86	0.67	0.34	**0.94**	**0.85**	**0.42**
$H_{1,2,50}$	0.90	0.76	0.39	0.91	0.78	0.39	0.90	0.76	0.38	0.90	0.76	0.38	0.91	0.80	0.40	**0.97**	**0.92**	**0.45**
$H_{1,2,80}$	0.96	0.89	0.45	0.97	0.92	0.46	0.96	0.90	0.45	0.96	0.90	0.45	0.97	0.92	0.46	**0.99**	**0.97**	**0.48**
$H_{1,2,100}$	0.98	0.96	0.48	0.99	0.97	0.48	0.99	0.97	0.48	0.99	0.97	0.48	0.99	0.99	0.49	**0.99**	**0.97**	**0.48**
$H_{1,3,10}$	**0.89**	**0.74**	**0.37**	**0.89**	**0.74**	**0.37**	**0.89**	**0.74**	**0.37**	**0.89**	**0.74**	**0.37**	**0.90**	**0.75**	**0.37**	**0.90**	**0.75**	**0.37**
$H_{1,3,30}$	0.94	0.85	0.42	0.95	0.87	0.43	0.94	0.86	0.42	0.94	0.86	0.42	**0.96**	**0.91**	**0.45**	**0.94**	**0.85**	**0.42**
$H_{1,3,50}$	0.96	0.91	0.45	0.97	**0.93**	**0.46**	0.96	0.90	0.44	0.96	0.91	0.44	**0.98**	**0.96**	**0.47**	**0.97**	**0.92**	**0.45**
$H_{1,3,80}$	0.99	0.96	**0.48**	0.99	0.97	0.48	0.99	0.96	0.47	0.98	0.96	0.47	**1.00**	**0.99**	**0.49**	**0.99**	**0.97**	**0.48**
$H_{1,3,100}$	0.99	0.98	0.48	**1.00**	**0.99**	**0.49**	0.99	0.98	0.48	0.99	0.98	0.48	**1.00**	**0.99**	**0.49**	**0.99**	**0.97**	**0.48**
$H_{2,3,10}$	0.80	0.52	0.27	0.81	0.52	0.27	0.81	0.53	0.27	0.80	0.52	0.27	0.80	0.52	0.27	0.83	0.59	0.30
$H_{2,3,30}$	0.82	0.55	0.28	0.82	0.55	0.28	0.82	0.56	0.28	0.82	0.54	0.28	0.82	0.54	0.28	0.86	0.66	0.33
$H_{2,3,50}$	0.84	0.60	0.30	0.84	0.61	0.30	0.84	0.61	0.31	0.84	0.60	0.30	0.84	0.60	0.30	0.87	0.68	0.34
$H_{2,3,80}$	0.85	0.64	0.32	0.86	0.65	0.32	0.85	0.64	0.32	0.85	0.64	0.32	0.85	0.63	0.31	0.89	0.72	0.36
$H_{2,3,100}$	0.87	0.67	0.33	0.88	0.70	0.34	0.87	0.67	0.33	0.87	0.67	0.33	0.86	0.66	0.32	0.90	0.76	0.37
$H_{3,2,10}$	0.88	0.72	0.35	0.88	0.71	0.35	0.88	0.72	0.36	0.88	0.71	0.35	0.88	0.71	0.35	0.85	0.64	0.33
$H_{3,2,30}$	0.83	0.60	0.30	0.83	0.60	0.30	0.83	0.62	0.31	0.83	0.62	0.31	0.83	0.61	0.31	0.89	0.72	0.36
$H_{3,2,50}$	0.84	0.62	0.31	0.84	0.62	0.31	0.84	0.62	0.31	0.84	0.63	0.32	0.84	0.62	0.31	0.91	0.78	0.39
$H_{3,2,80}$	0.83	0.59	0.30	0.83	0.58	0.29	0.83	0.60	0.30	0.83	0.60	0.31	0.83	0.59	0.30	0.93	0.82	0.40
$H_{3,2,100}$	0.82	0.56	0.29	0.82	0.57	0.28	0.83	0.61	0.31	0.82	0.60	0.30	0.83	0.60	0.29	0.93	0.83	0.41
$H_{3,3,10}$	0.84	0.60	0.31	0.84	0.60	0.31	0.84	0.61	0.31	0.84	0.61	0.31	0.84	0.60	0.31	0.85	0.64	0.33
$H_{3,3,30}$	0.87	0.67	0.34	0.87	0.68	0.34	0.87	0.67	0.34	0.87	0.68	0.34	0.87	0.67	0.34	0.89	0.72	0.36
$H_{3,3,50}$	0.88	0.71	0.35	0.89	0.72	0.36	0.89	0.72	0.36	0.89	0.72	0.35	0.89	0.73	0.36	0.91	0.78	0.39
$H_{3,3,80}$	0.90	0.75	0.37	0.90	0.75	0.37	0.89	0.75	0.37	0.90	0.76	0.37	0.91	0.78	0.38	0.93	0.82	0.40
$H_{3,3,100}$	0.91	0.79	0.39	0.92	0.80	0.39	0.90	0.77	0.38	0.92	0.80	0.39	0.93	0.83	0.41	0.93	0.83	0.41
$H_{4,2,10}$	0.89	0.72	0.36	0.88	0.72	0.36	0.89	0.72	0.36	0.89	0.72	0.36	0.89	0.73	0.36	0.85	0.63	0.33
$H_{4,2,30}$	**0.95**	**0.88**	**0.43**	**0.95**	**0.89**	**0.43**	**0.95**	**0.88**	**0.43**	**0.95**	**0.88**	**0.43**	0.96	0.89	0.44	0.92	0.81	0.40
$H_{4,2,50}$	**0.97**	**0.93**	**0.46**	**0.97**	0.93	0.46	0.97	0.93	0.46	**0.97**	**0.93**	**0.45**	0.98	0.94	0.46	0.94	0.86	0.42
$H_{4,2,80}$	0.98	0.96	0.47	**0.99**	**0.97**	**0.48**	0.99	0.97	0.48	**0.99**	**0.96**	**0.47**	0.99	0.97	0.48	0.96	0.91	0.44
$H_{4,2,100}$	**0.99**	**0.98**	**0.48**	0.99	0.98	0.48	**0.99**	**0.98**	**0.48**	**0.99**	**0.98**	**0.48**	0.99	0.99	0.49	0.98	0.96	0.47
$H_{4,3,10}$	0.80	0.51	0.26	0.80	0.52	0.26	0.81	0.52	0.27	0.80	0.52	0.26	0.81	0.52	0.27	0.85	0.63	0.33
$H_{4,3,30}$	0.94	0.86	0.42	0.95	0.87	0.42	0.95	0.87	0.43	0.94	0.86	0.42	0.95	0.87	0.43	0.92	0.81	0.40
$H_{4,3,50}$	0.97	0.93	0.46	0.97	0.93	0.46	**0.97**	**0.93**	**0.46**	0.97	0.93	0.45	0.98	0.94	0.46	0.94	0.86	0.42
$H_{4,3,80}$	**0.99**	**0.96**	0.48	0.99	0.97	0.48	**0.99**	**0.97**	**0.48**	**0.99**	**0.96**	**0.47**	0.99	0.97	0.48	0.96	0.91	0.44
$H_{4,3,100}$	0.99	0.98	0.48	0.99	0.98	0.48	**0.99**	**0.98**	**0.48**	**0.99**	**0.98**	**0.48**	0.99	0.99	0.49	0.98	0.96	0.47

2.6.2.2 Real Life Datasets

To test the performances of the proposed algorithms on higher dimensional regional data objects, we make use of four real life datasets.

The first one is the Fat and Oil dataset [20] which can be seen in Table 2.9. It consists of features of oil and fat collected from 6 plants and 2 animals, i.e., there are 8 data objects. In [121], the authors cluster these data objects into 3 categories based on the values of specific gravity (GRA), freezing point (FRE), Iodine value (IOD), and saponification (SAP). Note that each data object in the dataset is a hyper-rectangle in $\mathbf{R}^4$.

Table 2.9: Fat and Oil dataset

Observation	Name	GRA	FRE	IOD	SAP	Label
1	Linseed	[0.930 , 0.935]	[-27 , -18]	[170 , 204]	[118 , 196]	1
2	Perilla	[0.930 , 0.937]	[-5 , -4]	[192 , 208]	[188 , 197]	1
3	Cotton	[0.916 , 0.918]	[-6 , -1]	[99 , 113]	[189 , 198]	2
4	Sesame	[0.920 , 0.926]	[-6 , -4]	[104 , 116]	[187 , 193]	2
5	Camellia	[0.916 , 0.917]	[-21 , -15]	[80 , 82]	[189 , 193]	2
6	Olive	[0.914 , 0.919]	[0 , 6]	[79 , 90]	[187 , 196]	2
7	Beef	[0.860 , 0.870]	[30 , 38]	[40 , 48]	[190 , 199]	3
8	Hog	[0.858 , 0.864]	[22 , 32]	[53 , 77]	[190 , 202]	3

Table 2.10 reports rand index, pairwise F-measure, and mutual information values for Fat and Oil dataset. $F_{const,init,nc}$ represents that the fat and oil data is solved with nc instance level constraints generated by $const^{th}$ constraint generation method and the k-means based algorithms are initialized with the $init^{th}$ initialization method. Based on the results obtained in Section 2.6.2.1, we provide the values of the performance measures for only two constraint generation techniques in this section, namely must link constraints and actively generated constraints, since they are the most informative ones. As can be seen from the table, all of the algorithms find the optimal clustering for all constraint generation and initialization combinations; and all different number of constraints used, except UPC-k-means for $F_{1,2,5}$. This can be attributed to the small size of the dataset and we therefore also considered another high dimensional real life dataset consisting of 30 data objects.

Table 2.10: Performance measures (rand index (*Rand*), F-measure (*FM*), and mutual information (*MI*)) for the Fat and Oil dataset

Problem	UCOP-k-means			UPC-k-means			UCVQE			USeeded-k-means			UConstrained-k-means			UAHCP		
instance	Rand	FM	MI	Rand	FM	MI	Rand	FM	MI	Rand	FM	MI	Rand	FM	MI	Rand	FM	MI
$F_{1,2,1}$	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47
$F_{1,2,2}$	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47
$F_{1,2,3}$	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47
$F_{1,2,5}$	1.00	1.00	0.47	0.99	0.98	0.46	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47
$F_{1,2,10}$	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47
$F_{1,3,1}$	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47
$F_{1,3,2}$	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47
$F_{1,3,3}$	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47
$F_{1,3,5}$	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47
$F_{1,3,10}$	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47
$F_{4,2,1}$	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47
$F_{4,2,2}$	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47
$F_{4,2,3}$	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47
$F_{4,2,5}$	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47
$F_{4,2,10}$	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47
$F_{4,3,1}$	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47
$F_{4,3,2}$	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47
$F_{4,3,3}$	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47
$F_{4,3,5}$	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47
$F_{4,3,10}$	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47	1.00	1.00	0.47

For the objective function values (both global and local), the same results are obtained. According to solution time, all of the k-means based algorithms are better than UAHCP and they perform similarly. It should be noted that even UAHCP solves an instance of fat and oil data in less than one third of a second. Note that we do not report all the values for these performance measures for the datasets in this section. We just provide a short summary since we believe that the external performance measures (rand index, F-measure and mutual information) are more important than these ones in real life.

The second dataset that we use in our computational experiments is the well-known Iris dataset [64]. The original dataset consists of 150 individual observation classified into three classes (50 observations in each class). The classification task is performed based on 4 attributes; namely sepal length, sepal width, petal length, and petal width. In [18], the interval valued 30 observations of irises are provided. Each interval valued observation is generated by aggregating consecutive 5 individual observations (taking the minimum and maximum values of those 5 individuals). The authors state that such a transformation may be valid when the aggregated observations are collected under similar conditions. For example, in this case, the aggregated irises may be five different flowers of the same plant. Table 2.11 provides the interval valued iris data where each data object is a hyper-rectangle in $\mathbf{R}^4$.

Table 2.12 reports rand index, pairwise F-measure, and mutual information values for the interval valued Iris dataset. Similar to the previous notation, $I_{const,init,nc}$ represents that the iris data is solved with nc instance level constraints generated by $const^{th}$ constraint generation method and the k-means based algorithms are initialized with the $init^{th}$ initialization method. The following conclusions can be drawn from the table.

- Except the combination of provision of only must link constraints and initialization by the method in [11], all of the algorithms perform well for all other constraint generation and initialization combinations.
- For the combination of constraint generation by only must link constraints and initialization by the method in [11], the best algorithm is UAHCP. All the remaining algorithms have similar performances. It should be noted that, there

is an improvement in all the performance measures for all k-means based algorithms when the number of instance level constraints increases.

- UAHCP is able to find the optimal clustering in all cases.
- The actively generated constraints are more informative than must link constraints.
- Initialization by hierarchical clustering is better than initialization by the method in [11].

The same results are obtained for the objective function values (both global and local). According to solution time, although UAHCP is worse than all k-means based algorithms, it is still able to solve an instance of the iris data in less than 5 seconds.

In Figure 2.3, partitioning of the iris data is provided. Since each observation is on $\mathbf{R}^4$, we first applied principal component analysis (PCA), which is a well known dimensionality reduction technique, to data and then used the first three principal components (PC), which explain 99.48% of the variability in the data, to obtain 3-dimensional plot in Figure 2.3. Note that the data objects in the figure are points in $\mathbf{R}^3$. After finding assignments for interval valued observations with the solution methods proposed, we assume that the individual observations used to obtain the corresponding interval valued observations have the same assignments. The best partition found by the algorithms is the same with the desired (true) partition and it is provided in the figure.

Table 2.11: Interval valued Iris dataset

Observation	Sepal length	Sepal Width	Petal length	Petal Width	Label
1	[4.6 , 5.1]	[3.0 , 3.6]	[1.3 , 1.5]	[0.2 , 0.2]	Setosa
2	[4.4 , 5.4]	[2.9 , 3.9]	[1.4 , 1.7]	[0.1 , 0.4]	Setosa
3	[4.3 , 5.8]	[3.0 , 4.0]	[1.1 , 1.6]	[0.1 , 0.2]	Setosa
4	[5.1 , 5.7]	[3.5 , 4.4]	[1.3 , 1.7]	[0.3 , 0.4]	Setosa
5	[4.6 , 5.4]	[3.3 , 3.7]	[1.0 , 1.9]	[0.2 , 0.5]	Setosa
6	[4.7 , 5.2]	[3.0 , 3.5]	[1.4 , 1.6]	[0.2 , 0.4]	Setosa
7	[4.8 , 5.5]	[3.1 , 4.2]	[1.4 , 1.6]	[0.1 , 0.4]	Setosa
8	[4.4 , 5.5]	[3.0 , 3.5]	[1.2 , 1.5]	[0.1 , 0.2]	Setosa
9	[4.4 , 5.1]	[2.3 , 3.8]	[1.3 , 1.9]	[0.2 , 0.6]	Setosa
10	[4.6 , 5.3]	[3.0 , 3.8]	[1.4 , 1.6]	[0.2 , 0.3]	Setosa
11	[5.5 , 7.0]	[2.3 , 3.2]	[4.0 , 4.9]	[1.3 , 1.5]	Versicolor
12	[4.9 , 6.6]	[2.4 , 3.3]	[3.3 , 4.7]	[1.0 , 1.6]	Versicolor
13	[5.0 , 6.1]	[2.0 , 3.0]	[3.5 , 4.7]	[1.0 , 1.5]	Versicolor
14	[5.6 , 6.7]	[2.2 , 3.1]	[3.9 , 4.5]	[1.0 , 1.5]	Versicolor
15	[5.9 , 6.4]	[2.5 , 3.2]	[4.0 , 4.9]	[1.2 , 1.8]	Versicolor
16	[5.7 , 6.8]	[2.6 , 3.0]	[3.5 , 5.0]	[1.0 , 1.7]	Versicolor
17	[5.4 , 6.0]	[2.4 , 3.0]	[3.7 , 5.1]	[1.0 , 1.6]	Versicolor
18	[5.5 , 6.7]	[2.3 , 3.4]	[4.0 , 4.7]	[1.3 , 1.6]	Versicolor
19	[5.0 , 6.1]	[2.3 , 3.0]	[3.3 , 4.6]	[1.0 , 1.4]	Versicolor
20	[5.1 , 6.2]	[2.5 , 3.0]	[3.0 , 4.3]	[1.1 , 1.3]	Versicolor
21	[5.8 , 7.1]	[2.7 , 3.3]	[5.1 , 6.0]	[1.8 , 2.5]	Virginica
22	[4.9 , 7.6]	[2.5 , 3.6]	[4.5 , 6.6]	[1.7 , 2.5]	Virginica
23	[5.7 , 6.8]	[2.5 , 3.2]	[5.0 , 5.5]	[1.9 , 2.4]	Virginica
24	[6.0 , 7.7]	[2.2 , 3.8]	[5.0 , 6.9]	[1.5 , 2.3]	Virginica
25	[5.6 , 7.7]	[2.7 , 3.3]	[4.9 , 6.7]	[1.8 , 2.3]	Virginica
26	[6.1 , 7.2]	[2.8 , 3.2]	[4.8 , 6.0]	[1.6 , 2.1]	Virginica
27	[6.1 , 7.9]	[2.6 , 3.8]	[5.1 , 6.4]	[1.4 , 2.2]	Virginica
28	[6.0 , 7.7]	[3.0 , 3.4]	[4.8 , 6.1]	[1.8 , 2.4]	Virginica
29	[5.8 , 6.9]	[2.7 , 3.3]	[5.1 , 5.9]	[1.9 , 2.5]	Virginica
30	[5.9 , 6.7]	[2.5 , 3.4]	[5.0 , 5.4]	[1.8 , 2.3]	Virginica

Table 2.12: Performance measures (rand index (*Rand*), F-measure (*FM*), and mutual information (*MI*)) for interval valued Iris dataset

Problem	UCOP-k-means			UPC-k-means			UCVQE			USeeded-k-means			UConstrained-k-means			UAHCP		
instance	Rand	FM	MI	Rand	FM	MI	Rand	FM	MI	Rand	FM	MI	Rand	FM	MI	Rand	FM	MI
$I_{1,2,5}$	0.86	0.81	0.38	0.85	0.81	0.38	0.85	0.81	0.38	0.85	0.81	0.38	0.85	0.81	0.38	1.00	1.00	0.50
$I_{1,2,10}$	0.84	0.79	0.37	0.84	0.78	0.37	0.82	0.77	0.36	0.83	0.78	0.37	0.84	0.79	0.37	1.00	1.00	0.50
$I_{1,2,20}$	0.97	0.96	0.48	0.97	0.96	0.48	0.95	0.94	0.46	0.97	0.96	0.47	0.97	0.96	0.47	1.00	1.00	0.50
$I_{1,2,30}$	0.98	0.98	0.49	0.99	0.99	0.49	0.99	0.98	0.49	0.99	0.98	0.49	0.99	0.99	0.49	1.00	1.00	0.50
$I_{1,2,40}$	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50
$I_{1,3,5}$	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50
$I_{1,3,10}$	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50
$I_{1,3,20}$	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50
$I_{1,3,30}$	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50
$I_{1,3,40}$	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50
$I_{4,2,5}$	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50
$I_{4,2,10}$	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50
$I_{4,2,20}$	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50
$I_{4,2,30}$	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50
$I_{4,2,40}$	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50
$I_{4,3,5}$	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50
$I_{4,3,10}$	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50
$I_{4,3,20}$	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50
$I_{4,3,30}$	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50
$I_{4,3,40}$	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50

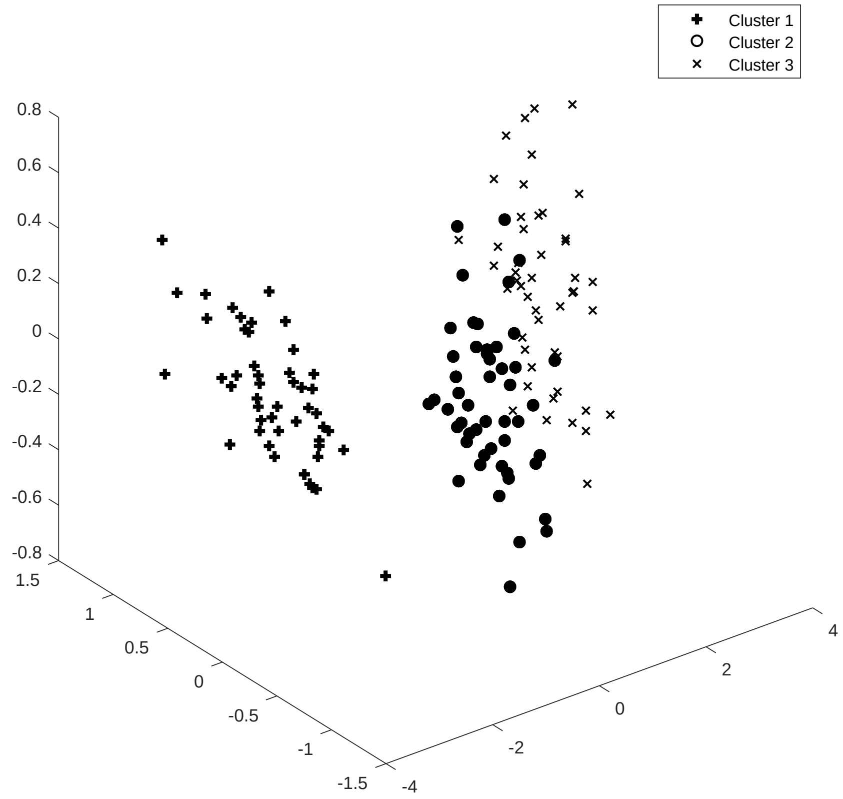


Figure 2.3: The best partition found by the algorithms (also the desired partition) for the Iris dataset

The third real life dataset is Wine dataset from UCI repository [103]. The dataset contains 178 individual (point) observations classified into three classes. Each observation corresponds to 13 continuous attributes of a wine which is cultivated from one of the three cultivars. The attributes are malic acid, ash, alcalinity of ash, magnesium, total phenols, flavanoids, nonflavanoid phenols, proanthocyanins, color intensity, hue, OD280/OD315 of diluted wines and proline. Similar to procedure used to obtain interval valued Iris dataset, we obtained interval valued Wine dataset by aggregating 2 consecutive observations. Such a transformation may be valid when the aggregated observations are two measurements taken from the same wine. Note that each data object in this dataset is a hyper-rectangle in $\mathbf{R}^{13}$.

Table 2.13 reports rand index, pairwise F-measure, and mutual information values for the interval valued Wine dataset. $W_{const,init,nc}$ in the table represents that the wine data is solved with nc instance level constraints generated by $const^{th}$ constraint generation method and the k-means based algorithms are initialized with the $init^{th}$ initialization method. To see which algorithm is better for a given constraint generation and initialization combination, the best value for each performance measure is highlighted in each row. Also, one-way ANOVA is performed to see the effect of algorithm on performance measures for each constraint generation and initialization combination. The p-values are reported in Table 2.14 and the best algorithm(s) provided in Table 2.15. Similarly, to determine which constraint generation and initialization combination leads to better performance for a given algorithm, the best values are boldfaced in each column for each number of instance level constraints. Also, we analyzed the results as a full factorial experiment for each performance measure of each algorithm by setting constraint generation method, initialization procedure and the number of constraints as factors. The p-values are reported in Table 2.16.

The following conclusions can be obtained.

- Based on Table 2.14, it is clear that the algorithm selected has significant effect on performance measures for different constraint generation and initialization combinations in general when we accept the significance level as 0.1. Based on Table 2.15, although there is no significant difference between UConstrained-k-means and UAHCP, they perform better than the other algorithms in terms of all performance measures for all constraint generation and initialization combinations. Note that the algorithms in each cell of Table 2.15 are provided in descending order with respect to performance.

- Based on Table 2.16, constraint generation methods do not significantly differs from each other for different performance measures of different algorithms in general. On the other hand, initialization procedure selected has significant effect on performance most of the time and it seems that initialization with method in [11] is better based on Table 2.13. Moreover, the number of constraints has significant positive effect on performance in general. It can be seen from Table 2.13, providing more constraints results in better performance.

- For all k-means based algorithms, the best combination of constraint generation and initialization is the combination of providing must-link constraints and initializing the algorithms with the method in [11] in general.
- Providing only must-link constraints to UAHCP is better than generating constraints actively.

For the objective function values (both global and local), similar results are obtained. According to solution time, k-means based algorithms solve an instance of the wine data in approximately 20 seconds when UAHCP requires approximately 15 minutes to solve an instance.

The desired partition and the best partition found for the wine data are provided in Figure 2.4. Similar to the procedure followed to plot partitioning of the iris data, we applied PCA and used first three PCs, which explain 99.99% of the variability in the data, to obtain the plots. Also, we assumed that the assignments of the individual observations are the same with the assignments of the interval valued observations for which they are used. The best partition was found by UConstrained-k-means where we provided 80 must-link constraints and used the method in [11] as the initialization procedure.

Table 2.13: Performance measures (rand index (*Rand*), F-measure (*FM*), and mutual information (*MI*)) for interval valued Wine dataset

Problem	UCOP-k-means			UPC-k-means			UCVQE			USeeded-k-means			UConstrained-k-means			UAHCP		
instance	Rand	FM	MI	Rand	FM	MI	Rand	FM	MI	Rand	FM	MI	Rand	FM	MI	Rand	FM	MI
$W_{1,2,5}$	**0.73**	**0.63**	**0.24**	**0.73**	**0.63**	**0.24**	**0.73**	**0.63**	**0.24**	**0.73**	**0.63**	**0.24**	0.73	**0.64**	0.25	0.71	**0.64**	0.24
$W_{1,2,10}$	0.72	**0.63**	0.24	0.72	0.62	0.23	0.72	0.62	0.23	0.72	0.62	0.23	0.73	0.64	0.25	0.72	**0.65**	0.25
$W_{1,2,20}$	**0.76**	**0.66**	**0.26**	0.74	**0.63**	**0.25**	0.74	**0.63**	**0.24**	0.74	**0.63**	**0.25**	0.77	**0.68**	**0.28**	**0.74**	**0.66**	**0.26**
$W_{1,2,50}$	0.76	**0.64**	**0.27**	**0.74**	**0.62**	**0.24**	**0.75**	0.63	0.24	**0.74**	**0.62**	**0.24**	**0.85**	**0.78**	**0.34**	0.77	**0.71**	**0.29**
$W_{1,2,80}$	0.77	0.65	**0.30**	**0.74**	0.61	0.23	**0.74**	0.61	0.23	**0.74**	0.61	0.23	**0.93**	**0.89**	**0.41**	**0.86**	**0.82**	**0.36**
$W_{1,3,5}$	0.71	0.62	0.23	0.71	0.62	0.23	0.71	0.62	0.23	0.71	0.62	0.23	0.71	0.63	0.24	0.71	**0.64**	0.24
$W_{1,3,10}$	0.71	0.62	0.23	0.71	**0.62**	**0.23**	0.71	0.62	0.23	0.71	**0.62**	**0.23**	0.72	0.64	0.24	0.72	**0.65**	0.25
$W_{1,3,20}$	0.72	0.62	0.24	0.72	0.62	0.23	0.71	0.62	0.23	0.72	0.62	0.23	0.74	0.66	0.25	**0.74**	**0.66**	**0.26**
$W_{1,3,50}$	0.73	0.64	0.26	0.72	0.62	0.23	0.72	0.62	0.23	0.72	0.62	0.23	0.77	0.71	0.29	0.77	**0.71**	**0.29**
$W_{1,3,80}$	0.75	0.64	0.28	0.73	0.61	0.23	0.73	0.62	0.23	0.73	0.61	0.23	0.86	0.82	0.36	**0.86**	**0.82**	**0.36**
$W_{4,2,5}$	0.71	0.62	0.24	0.71	0.62	0.23	0.72	0.62	0.24	0.71	0.62	0.23	**0.74**	0.63	**0.25**	**0.73**	0.63	**0.24**
$W_{4,2,10}$	0.73	0.62	0.24	0.72	0.62	0.23	0.72	**0.62**	**0.24**	0.72	0.62	0.23	**0.76**	**0.65**	**0.25**	**0.74**	0.64	**0.25**
$W_{4,2,20}$	0.75	0.63	0.25	**0.74**	0.62	0.24	**0.75**	0.63	0.24	**0.74**	0.62	0.24	**0.77**	0.66	0.26	0.74	0.65	0.25
$W_{4,2,50}$	**0.76**	0.64	0.26	0.74	0.61	0.23	0.74	0.62	0.23	0.74	0.61	0.23	0.83	0.74	0.31	**0.78**	0.70	0.28
$W_{4,2,80}$	**0.80**	**0.68**	0.29	0.74	0.61	0.23	0.74	0.63	**0.24**	0.74	0.61	0.23	0.91	0.86	0.39	0.82	0.77	0.32
$W_{4,3,5}$	0.72	0.62	0.23	0.72	0.62	0.23	0.73	0.62	0.23	0.72	0.62	0.23	0.73	0.63	0.24	**0.73**	0.63	**0.24**
$W_{4,3,10}$	**0.74**	0.62	**0.24**	**0.73**	0.61	0.23	**0.73**	0.62	0.23	**0.73**	0.61	0.23	0.75	0.64	0.24	**0.74**	0.64	**0.25**
$W_{4,3,20}$	0.73	0.63	0.24	0.72	0.63	0.24	0.73	0.63	0.24	0.72	0.63	0.24	0.73	0.64	0.25	0.74	0.65	0.25
$W_{4,3,50}$	0.75	0.63	0.26	0.73	0.62	0.24	0.75	**0.64**	**0.25**	0.73	0.62	0.24	0.79	0.70	0.29	**0.78**	0.70	0.28
$W_{4,3,80}$	0.75	0.63	0.26	0.73	**0.62**	**0.23**	0.74	**0.63**	0.24	0.73	**0.62**	**0.23**	0.82	0.77	0.32	0.82	0.77	0.32

Table 2.14: P-values for the effect of algorithm on different performance measures for each constraint generation and initialization combination

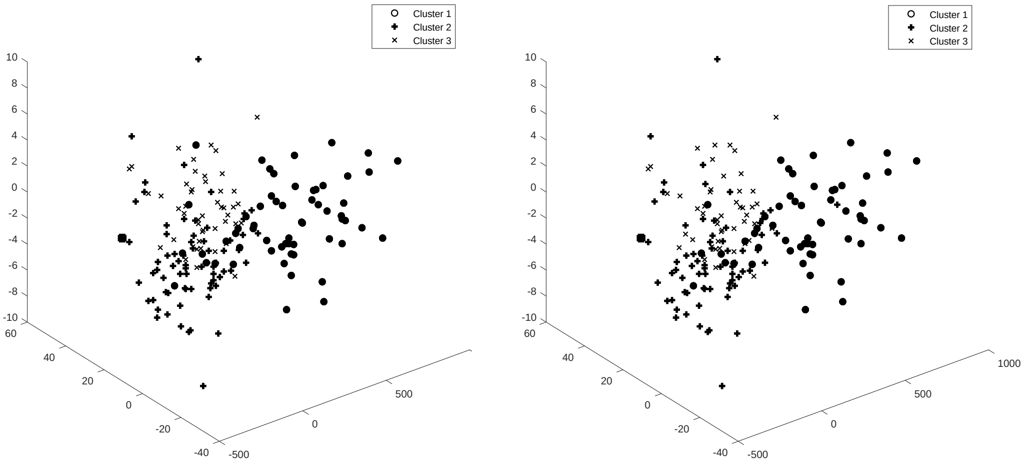
Constraint type & Initialization	Performance measures		
	Rand	FM	MI
Must-link & Method in [11]	0.18	0.02	0.05
Must-link & Agglomerative hierarchical clustering	0.16	0.02	0.06
Active & Method in [11]	0.04	0.02	0.03
Active & Agglomerative hierarchical clustering	0.05	0.01	0.02

Table 2.15: Best algorithm(s) for different performance measures for each constraint generation and initialization combinations

Constraint type & Initialization	Performance measures		
	Rand	FM	MI
Must-link & Method in [11]	UConstrained-k-means UAHCP	UConstrained-k-means UAHCP	UConstrained-k-means UAHCP
Must-link & AHC	UAHCP UConstrained-k-means	UAHCP UConstrained-k-means	UAHCP UConstrained-k-means
Active & Method in [11]	UConstrained-k-means	UConstrained-k-means UAHCP	UConstrained-k-means UAHCP
Active & AHC	UConstrained-k-means UAHCP	UAHCP UConstrained-k-means	UAHCP UConstrained-k-means

Table 2.16: P-values for the effects of constraint generation, initialization and the number of constraints on different performance measures of algorithms

Performance Measure	Constraint type	Initialization	Number of constraints
Rand of UCOP-k-means	0.15	0.00	0.00
FM of UCOP-k-means	0.68	0.03	0.03
MI of UCOP-k-means	0.12	0.00	0.00
Rand of UPC-k-means	0.34	0.01	0.03
FM of UPC-k-means	0.08	0.94	0.05
MI of UPC-k-means	0.19	0.37	0.12
Rand of UCVQE	0.10	0.05	0.03
FM of UCVQE	0.19	0.93	0.55
MI of UCVQE	0.21	0.42	0.52
Rand of USeeded-k-means	0.32	0.01	0.03
FM of USeeded-k-means	0.10	0.99	0.04
MI of USeeded-k-means	0.21	0.41	0.11
Rand of UConstrained-k-means	0.80	0.00	0.00
FM of UConstrained-k-means	0.19	0.01	0.00
MI of UConstrained-k-means	0.14	0.01	0.00
Rand of UAHCP	0.74	-	0.00
FM of UAHCP	0.01	-	0.00
MI of UAHCP	0.10	-	0.00



(a) Desired partition

(b) Best partition found

Figure 2.4: Partitions for the Wine dataset

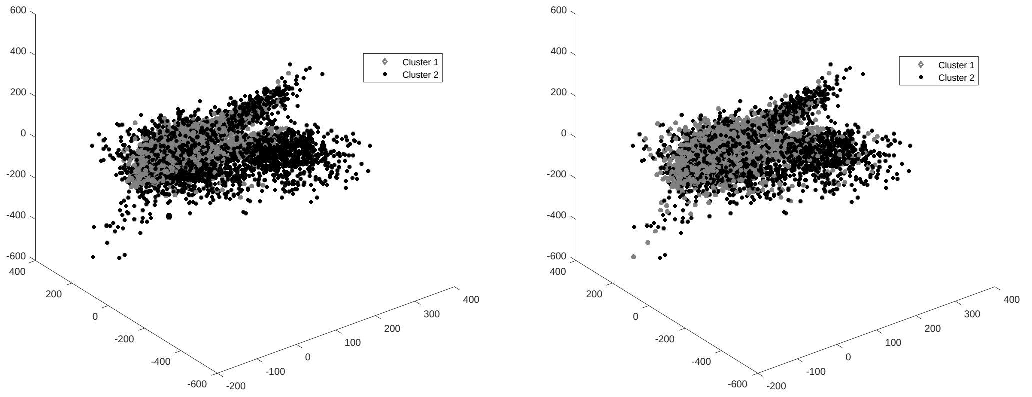
The last real life dataset we used is the interval valued Gamma dataset. The original version of the dataset, MAGIC Gamma Telescope dataset, is obtained from the UCI repository [103]. It contains 19020 individual observations classified into two classes. Each observation has 10 attributes. To obtain the interval valued Gamma dataset, we aggregated 5 consecutive observations. Note that each of the 3803 data objects in the obtained interval dataset is a hyper-rectangle in $\mathbf{R}^{10}$.

For this dataset, we utilized, based on the previous results, the best two algorithms, namely, UConstrained-k-means and UAHCP, and the best constraint generation and initialization pair, namely must-link constraints and initialization by the method in [11]. UConstrained-k-means solves an instance of the interval valued Gamma dataset in about 25 minutes, while UAHCP could not solve an instance within the given 3 hour time limit. This is expected as UAHCP considers the possibility of merging $O((n+m)^3)$ clusters in the generation of the dendrogram making it not suitable for large scale instances.

Table 2.17 reports rand index, pairwise F-measure, and mutual information values obtained with UConstrained-k-means. $G_{1,2,nc}$ in Table 2.17 represents that the gamma data is solved with nc must-link constraints by utilizing the method in [11] as the initialization procedure. Values of performance measures increase with the number of constraints indicating that the agreement between the found partition and the desired partition improves.

Table 2.17: Performance measures (rand index (*Rand*), F-measure (*FM*), and mutual information (*MI*)) of UConstrained k-means for the interval valued Gamma dataset

Problem	UConstrained-k-means		
instance	Rand	FM	MI
$G_{1,2,100}$	0.68	0.73	0.11
$G_{1,2,500}$	0.69	0.74	0.12
$G_{1,2,1000}$	0.70	0.74	0.13
$G_{1,2,2000}$	0.76	0.80	0.18
$G_{1,2,3000}$	0.83	0.85	0.26



(a) Desired partition (b) Best partition found

Figure 2.5: Partitions for Gamma dataset

Figure 2.5 shows the desired partition and the best partition found for the gamma data. Again, we used the first three PCs, which explain 83.34% of the variability in the data, to obtain the plots and assumed that the assignments of the individual observations are the same with the assignments of the interval valued observations for which they are used. The best partition was found in the case where we provided 3000 constraints.

2.7 Conclusion and Future Work

We considered a semi-supervised clustering problem with regional data objects that aims to minimize the total of the sum of the violation costs of the unsatisfied instance level constraints and a weighted sum of squared maximum Euclidean distances between the locations of the data objects and the centroids of the clusters they are assigned to. It is assumed that the data objects are closed convex bounded polytopes and/or closed disks.

We first considered the problem of computing the centroid of a given cluster (CCP) and formulated the problem as an SOCP which can be solved in polynomial time.

We then adapted the subgradient method to the problem which turned out to be faster than the SOCP formulation.

For the solution of the considered semi-supervised clustering problem (SSC), seven solution approaches, namely MISOCP formulation, UCOP-k-means, UPC-k-means, UCVQE, USeeded-k-means, UConstrained-k-means, and UAHCP are proposed and compared using six performance measures on both artificial and real life datasets. The MISOCP formulation turned out to be applicable only for small size instances. For medium size instances, UConstrained-k-means and UAHCP perform better than the other algorithms in most of the cases. For large size instances, UAHCP becomes time inefficient making UConstrained-k-means the overall best. Each iteration of the UConstrained-k-means takes $O\left((1/\epsilon^2)kd(n\ell + m) + k|CL|(n+m)\right)$ time and linear space. The details are as follows. In Steps 5 - 7 of the algorithm, for each polytope and disk in the chunklets, farthest distances to the nearest cluster centroid are calculated. Time required to find farthest distance to a cluster centroid is $O(d\ell)$ and $O(d)$ for a polytope and for a disk, respectively. Note that to determine the nearest cluster, we need to check farthest distances to all clusters. As the number of polytopes and disks in the chunklets is at most the number of polytopes and disks in the problem instance, Steps 5 - 7 take $O(nkd\ell + mkd)$ time in total. In Step 9, random ordering of the data objects not belonging to any chunklet is required. Since the maximum number of such data objects can be the total number of data objects in the problem instance, i.e., $n + m$, Step 9 needs $O(n + m)$ time in the worst case. In Steps 10 - 17, each data object not belonging to any chunklet is assigned to the nearest appropriate cluster. For a polytope, in the search of the nearest appropriate cluster, finding the farthest distances to all cluster centroids takes $O(kd\ell)$ time and checking the appropriateness of a cluster requires at most $|CL|$ comparisons. In the worst case, the appropriate cluster for a data object can be the k^{th} nearest cluster and so $k|CL|$ comparisons can be required. Thus, for a polytope, $O(kd\ell + k|CL|)$ time is needed. Similarly, for a disk, $O(kd + k|CL|)$ time is required. Note that the number of polytopes and disks not belonging to any chunklet can be at most the number of polytopes and disks in the problem instance. Therefore, Steps 10 - 17 require $O(n(kd\ell + k|CL|) + m(kd + k|CL|))$ time in the worst case. Finally, in Step 20, SM requiring $O((1/\epsilon^2)d(n\ell + m))$ time is run k times. In total, each iteration of the UConstrained-k-means takes $O\left((1/\epsilon^2)kd(n\ell + m) + k|CL|(n+m)\right)$ time.

To solve each problem instance, we implemented different combinations of constraint generation and initialization procedures and considered different numbers of constraints. Eventually, we obtained more than a thousand results to compare for each performance measure and provided a summary of the results.

As a future work, one can consider regional representatives instead of point representatives (i.e, centroids). Also, fuzzy versions of the problem, where the assumption of assigning each regional data object to a single cluster is relaxed, can be studied.

The publications related with this chapter can be found in [51, 52].

CHAPTER 3

DATA MINING FOR CENTRAL NODES IN A GRAPH-STRUCTURED DATA OBJECT

3.1 Introduction

As it is stated before, traditional data mining techniques deal with data points, i.e., data objects which are represented by numerical vectors in the space. But improving technology and measurement capabilities, and the need for deeper analyses result in collecting more complex datasets as well as larger datasets [113]. Such complex datasets may include images, shapes and graphs. In Chapter 3 and Chapter 4, we will focus on data mining for graph-structured data objects.

Graph mining problems arise in many areas such as biology, neuroscience, medical imaging, computer or social networks [2]. They are of two types. In the first type, there is only one graph and aim is to mine the graph based on the edge behavior to find the dense regions on the graph or to identify the central nodes of the graph etc. In the second type, there are many graphs and aim is to mine the graphs based on the overall structural behavior. In this chapter, we will address the first type while the second type is considered in the next chapter.

Identification of central nodes in a graph is an essential problem in network analysis. Applications include finding the most influential nodes in a social network where the nodes are usually the individuals and the edges between them represent interactions, collaborations, or influences [114] and identifying nodes of high importance in the transmission of information, viruses etc. in a computer network where the nodes could be the individual devices on the network and the edges represent connections between them [32].

Which node/group of nodes is the most central in a graph? Answer is purpose and context dependent. Numerous measures have been introduced in the literature in order to identify the most central vertex in a network including degree centrality, closeness centrality, betweenness centrality and eigenvector centrality [114], [150]. Most of these measures have been extended to groups of vertices with the aim of identifying the most central groups of vertices in a network, e.g., group degree centrality, group closeness centrality, and group betweenness centrality [62].

Group centrality measures have found numerous applications. Ni et al. [120] used group degree, closeness, and betweenness centralities to explore the role of disciplines in knowledge communication and interaction networks. Kchiche and Kamoun [89] studied group centrality based deployment strategies of roadside units in vehicular networks. In [56], the authors proposed a method for finding candidate locations for additional deployment of network monitors using group betweenness centrality.

Given a network and a group size k, the problem of finding a subset of the vertices of size k that is the most central (according to a predefined criterion) is a combinatorial problem. The number of possible subsets to explore may grow too much necessitating efficient computation or approximation of centrality of several groups successively.

In this chapter, our focus is on the successive computation of the bounds on the group betweenness centrality of several groups efficiently. Betweenness centrality of a vertex in a network represents the total influence it has on communications between every pair of vertices in the network assuming that information flows through the shortest paths. Group betweenness centrality (GBC) of a group of vertices measures the influence the group has on communications collectively.

Given a group size k, the problem of finding k vertices in a graph with the highest group betweenness centrality is an NP-hard problem [126]. Therefore heuristic approaches [126] and approximation algorithms [56] have been proposed in the literature. Kolaczyk et al. [92] proposed lower and upper bounds on the group betweenness centrality and therefore their approach can be used to estimate group betweenness centrality of any given subset of the vertices. The proposed upper bound (in its strongest form) requires the solution of an NP-hard problem, namely the traveling salesman problem, and hence is not practical. In this chapter, we propose improvements on their method. We replace the upper and lower bounds proposed in [92] with stronger bounds from the literature. The upper bound, in addition to being stronger, is much faster to compute. Moreover, we introduce a modification to their method which enables faster computation of the bounds for several groups successively.

The organization of this chapter is as follows. In Section 3.2, the notation is provided. Betweenness and group betweenness centralities are introduced in Section 3.3. Section 3.4 details the bounds proposed for the group betweenness centrality and their faster computation for several groups successively. An illustrative example is also provided in Section 3.4. Section 3.5 contains the results of the computational experiments performed on randomly generated and real-life networks. Finally, conclusion is provided in Section 3.6.

3.2 Notation

Let $G = (V, E)$ be a simple, unweighted, undirected, and connected graph, where V is the set of its vertices and $E \subseteq V \times V$ is the set of its edges. We assume that $n = |V|$ and $m = |E|$. We say that vertices u and w are adjacent if $\{u, w\} \in E$. A walk from vertex u to w is a sequence of vertices $u_0 = u, u_1, \ldots, u_\ell = w$ such that u_i and u_{i+1} are adjacent for every $i \in \{0, 1, \ldots, \ell - 1\}$. The length of this walk is equal to ℓ. A walk can also be represented by a sequence of edges $\{u_0, u_1\}, \{u_1, u_2\}, \ldots, \{u_{\ell-1}, u_\ell\}$ with the ending vertices of the edges being the same as the starting vertices of the following edges. If all the edges are different in a walk, then the walk is called a path. By $d(u, w)$, we denote the length of a shortest path from u to w. Clearly, we have that $d(u, w) = d(w, u)$ for every pair of vertices $u, w \in V$. We assume that $d(u, u) = 0$ for every vertex $u \in V$.

For two vertices u and $w \in V$, σ_{uw} represents the number of shortest paths from u to w and is equal to σ_{wu}. It is assumed that $\sigma_{uu} = 1$ for every vertex $u \in V$. The number of shortest paths from u to w that pass through $v \in V$ is denoted by $\sigma_{uw}(v)$. We have that

$$\sigma_{uw}(v) = \begin{cases} \sigma_{uv}\sigma_{vw}, & \text{if } d(u, w) = d(u, v) + d(v, w), \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

3.3 Betweenness and Group Betweenness Centrality

Betweenness centrality of a vertex $v \in V$, $BC(v)$, represents the total influence v has on communications between every pair of vertices in G assuming that information flows through shortest paths. More formally, we have that

$$BC(v) = \sum_{\{u,w\} \in V \times V | u,w \neq v | u < w} \frac{\sigma_{uw}(v)}{\sigma_{uw}}. \tag{3.2}$$

In the above sum, the condition that u and w have to be different from v can be removed. In this case the betweenness centrality of each vertex v increases by the constant $n-1$ and is denoted by $\overline{BC}(v)$. We have that

$$\overline{BC}(v) = \sum_{\{u,w\} \in V \times V | u < w} \frac{\sigma_{uw}(v)}{\sigma_{uw}} = BC(v) + n - 1. \tag{3.3}$$

$BC(v)$ and $\overline{BC}(v)$ can be normalized by dividing them, respectively, by $\binom{n-1}{2}$ and $\binom{n}{2}$ to make them into measures between 0 and 1. By this normalization, the betweenness centrality can also be stated probabilistically. For instance, $\overline{BC}(v)/\binom{n}{2}$ represents the probability that the information sent between a pair of distinct vertices chosen uniformly at random passes through v if the information flows through one of the shortest paths selected uniformly at random. $BC(v), \overline{BC}(v)$, and their normalized versions are all equivalent in the sense that having the value of one of the measures for a vertex, one can immediately obtain the values of the others for the same vertex by multiplication/division by a constant and/or addition/subtraction of a constant. Therefore in terms of the complexity of computing the betweenness centrality of a single vertex or betweenness centralities of all the vertices in a graph, the four measures are all equivalent. Furthermore, if one orders the vertices by non-increasing value of betweenness centrality, the four measures give exactly the same ordering.

Until the paper by Brandes [26], the fastest algorithms for the computation of the betweenness centralities of all the vertices in an undirected graph required $O(n^3)$ time and $O(n^2)$ space. Brandes [26] introduced the notion of the dependency of a vertex $u \in V$ on a vertex $v \in V$, $\delta_u(v)$, defined as

$$\delta_u(v) = \sum_{w \in V | w \neq u} \frac{\sigma_{uw}(v)}{\sigma_{uw}}, \tag{3.4}$$

which represents the total fraction of shortest paths that start at u and pass through v. After exploiting an observation that the partial sums in Equation 3.4 obey a recursive relation, he showed that $\delta_u(v)$ values for all $u, v \in V$ can be computed in $O(nm)$ time. In terms of the $\delta_u(v)$'s, the betweenness centrality of a vertex v can be written as

$$\overline{BC}(v) = \sum_{u \in V} \frac{\delta_u(v)}{2}. \tag{3.5}$$

After the computation of all the dependencies which take $O(nm)$ time, taking the above sum for all the vertices in V requires $O(n^2)$ time resulting in the $O(nm)$ overall time complexity (and a space requirement of $O(n + m)$) for the computation of all betweenness centrality values.

Betweenness centrality of individual vertices can be extended to groups of vertices in different ways. Let S be a subset of the vertices V. One way to extend betweenness centrality to groups of vertices is by counting the shortest paths that pass through all the vertices in S and using this quantity in Equations 3.2 or 3.3 in place of $\sigma_{uw}(v)$. This notion was introduced by Kolaczyk et al. [92] and called as the co-betweenness centrality. Formally, letting $\sigma^{\#}_{uw}(S)$ represent the number of shortest paths between u and w that pass through all the vertices in S, the co-betweenness centrality of S, $C(S)$, is defined as

$$C(S) = \sum_{\{u,w\} \in V \times V | u,w \notin S | u<w} \frac{\sigma^{\#}_{uw}(S)}{\sigma_{uw}}. \tag{3.6}$$

If the end vertices of the shortest paths are allowed to be in S, then we have

$$\overline{C}(S) = \sum_{\{u,w\} \in V \times V | u<w} \frac{\sigma^{\#}_{uw}(S)}{\sigma_{uw}}. \tag{3.7}$$

When $S = \{u\}$ for some $u \in V$ or $S = \{u, w\}$ for two vertices $u, w \in V$, we use the notations $\overline{C}(u)$ and $\overline{C}(u, w)$, respectively, for $\overline{C}(S)$. $C(S)$ and $\overline{C}(S)$ in a similar fashion can be normalized to obtain measures whose values are between 0 and 1.

Another way to extend betweenness centrality to groups of vertices is by counting the shortest paths that pass through at least one of the vertices in S. Considering a road network with roadside equipments to be placed on certain vertices, the group betweenness centrality of a set of vertices S defined in this manner represents the likelihood that a vehicle traveling between a random pair of vertices through one of the shortest paths selected uniformly at random meets at least one roadside equipment along its way to destination.

Letting $\sigma^*_{uw}(S)$ denote the number of shortest paths between u and w that pass through at least one vertex in S, the group betweenness centrality (GBC) of a vertex set S, $GB(S)$, is defined as

$$GB(S) = \sum_{\{u,w\} \in V \times V | u,w \notin S | u<w} \frac{\sigma^*_{uw}(S)}{\sigma_{uw}}. \tag{3.8}$$

Similarly, allowing the end vertices of the shortest paths to be in S, we define, $\overline{GB}(S)$ as

$$\overline{GB}(S) = \sum_{\{u,w\} \in V \times V | u<w} \frac{\sigma^*_{uw}(S)}{\sigma_{uw}}. \tag{3.9}$$

Brandes [27] proposed a method to find the GBC of a single subset of vertices of size k in $O(nm)$ time. His method is not practical for finding the group (of size k) with the highest GBC value as this would take $O\left(nm\binom{n}{k}\right)$ time. Puzis et al. [125] proposed an algorithm to compute successively the GBC of subsets of vertices of a given size k. The algorithm has a preprocessing step taking $O(n^3)$ time after which the computation of each GBC of a set of vertices of size k takes $O(k^3)$ time. In a recent paper [142], the authors introduced mixed integer programming formulations for the problem of finding a group of k vertices with the highest GBC value and demonstrated the applicability of their method on medium size sparse instances. In this study, we do not compute the exact GBC values. Similar to [125], a preprocessing step computes certain quantities. After the preprocessing step, lower and upper bounds on the GBC value of a set of vertices of size k are computed and this step takes $O(k^2)$ time for each group of size k. We show in the computational experiments that with these bounds, several groups can be eliminated from further consideration in the search for a group with the highest GBC value.

Path betweenness centrality is another generalization of the betweenness centrality from a single vertex to a sequence of vertices. The reader is referred to [125] for more details on the path betweenness centrality. For the special case in which we have a subset $S = \{s_1, s_2\}$ of size two, the path betweenness centrality of (s_1, s_2) is equal to the co-betweenness centrality of S.

3.3.1 Bounds on the Group Betweenness Centrality

Kolaczyk et al. [92] proposed a method providing lower and upper bounds on the GBC of a subset S of V. For a subset T of S, they introduced the co-betweenness of T with respect to S, $C_S(T)$, as

$$C_S(T) = \sum_{\{u,w\} \in V \times V | u,w \notin S | u<w} \frac{\sigma^{\#}_{uw}(T)}{\sigma_{uw}}. \tag{3.10}$$

With this definition, they showed that the group betweenness centrality of S, $GB(S)$, can be bounded as

$$\sum_{s \in S} C_S(s) - \sum_{\{s^1,s^2\} \in S \times S | s^1 < s^2} C_S(s^1, s^2) \leq GB(S) \leq \sum_{s \in S} C_S(s) - \sum_{j=1}^{k-1} C_S(s_j, s_{j+1}), \tag{3.11}$$

where $s_1, s_2, \ldots, s_k$ is any ordering of the vertices in S. Computation of the best upper bound, i.e., finding the ordering $s_1, s_2, \ldots, s_k$ such that the upper bound is the smallest, is in general an NP-hard problem as it can be transformed into the traveling salesman problem. Computation of the co-betweenness centrality of each vertex and the co-betweenness centrality of every pair of distinct vertices with respect to S takes $O(n^3)$ time. Computation of the lower bound in Equation 3.11 takes $O(k^2)$ time and the computation of the best upper bound takes exponential time in k. When the ordering of the vertices is fixed a priori, the computation of the upper bound takes $O(k)$ time, however in this case the bound usually becomes very loose.

When one wants to compute successively the bounds on the GBC of several groups, say $S_1, \ldots, S_\ell$, all the quantities in Equation 3.11 need to be recomputed with respect to all these groups. This is because there is no simple relation between $C_S(u, v)$ and $C_W(u, v)$ for two different groups S and W containing the same vertices u and v.

We propose the following modifications in computing the bounds on the GBC.

1. An upper bound, which is stronger than the strongest one in Equation 3.11, is

proposed that is faster to compute than finding the strongest upper bound in Equation 3.11.

2. A lower bound, which is stronger than the one in Equation 3.11, is proposed. The upper and lower bounds proposed are adaptations of the existing bounds in probability theory literature to the problem considered in this study.

3. A preprocessing step, which eliminates the dependency to the group in consideration, is proposed that computes all the necessary quantities (betweenness and co-betweenness values) in $O(n^3)$ time. After the preprocessing step, there is no need to recompute the co-betweenness values for the computation of the GBC of different groups. This results in faster computation of the bounds for successive groups. For each group of size k, the computation of the new bounds take $O(k^2)$ time whereas, in the method proposed in [92], the computation of the bounds take $O(n^3 + k^2)$ time if the ordering of the vertices is fixed a priori or $O(n^3 + f(k))$ time if the best upper bound in Equation 3.11 is computed where $f(k)$ is not bounded by any polynomial of k.

3.4 Computation of the New Bounds

In this section, we explain the modifications we propose to bound the GBC of several groups successively. First, to make things easier to explain, we will use the following normalized version of the GBC.

$$\overline{GBN}(S) = \sum_{\{u,w\} \in V \times V | u < w} \frac{\sigma^*_{uw}(S)}{\sigma_{uw}} / \binom{n}{2}. \tag{3.12}$$

The ratio $\dfrac{\sigma^*_{uw}(S)}{\sigma_{uw}}$ represents the probability that the information sent between u and w passes through at least one of the vertices in S (under the assumptions that the information flows through the shortest paths and if there are more than one shortest path between u and w, the information flows through one of the shortest paths chosen uniformly at random from all shortest paths between u and w.). Therefore $\overline{GBN}(S)$ can be interpreted as the probability that the information sent between a pair of distinct vertices chosen uniformly at random from V passes through at least one of the vertices in S if the information flows through one of the shortest paths selected uniformly at random. This probabilistic interpretation already exists in the literature, see e.g. [63]. For a vertex $u \in V$, let $E(u)$ denote the event that the information sent between a pair of distinct vertices chosen uniformly at random passes through u. Similarly, let $E(u, w)$ be the event that the information sent between a pair of distinct vertices chosen uniformly at random passes through both u and w, i.e., $E(u, w) = E(u) \cap E(w)$. We denote by $P(u)$ and $P(u, w)$ the probabilities that events $E(u)$ and $E(u, w)$ occur, respectively, i.e., $P(u) = prob(E(u))$ and $P(u, w) = prob(E(u, w))$, where $prob(E)$ represents the probability that event E occurs.

With these definitions, we have that

$$\overline{GBN}(S) = prob(\cup_{v \in S} E(v)). \tag{3.13}$$

In probability theory, there is a vast literature on bounding the probability of union of events in terms of the individual probabilities and joint probabilities of up to $q < |S|$ events. Such bounds are known as the bounds of order q. As the value of q increases, the quality of the bounds and the complexity of their computation increase in general. With these considerations, we consider using bounds of order 2 that employ $P(u)$ and $P(u, w)$ values. So, once $P(u)$ values for every $u \in V$ and $P(u, w)$ values for every pair of distinct vertices $u, w \in V$ are computed in a preprocessing step, one can use these quantities to compute successively bounds on the GBC of several groups. Note that the quantities used to bound the GBC of a group S in Equation 3.11 are dependent on the group S, and hence all the quantities need to be recomputed from scratch for every group which makes the method in Kolaczyk et al. [92] less efficient.

For a vertex $u \in V$, we have that $P(u) = \overline{BC}(u)/\binom{n}{2}$ and for a distinct pair of vertices $u, w \in V$, we have that $P(u, w) = \overline{C}(u, w)/\binom{n}{2}$. Note that, by definition, we have that $P(u) = P(u, u)$ for every vertex $u \in V$. In the preprocessing step of our algorithm, all betweenness centrality values $\overline{BC}(u), u \in V$ and all co-betweenness centrality values $\overline{C}(u, w), u, w \in V, u \neq w$ are computed.

3.4.1 Preprocessing Step

In the preprocessing step, we start with computing for every pair of distinct vertices $u, w \in V$, the distance $d(u, w)$ and the number of shortest paths σ_{uw} between them. These values can be computed by breadth first search in $O(nm)$ time [26]. We then compute $\delta_u(v)$ values for every $u, v \in V$ which can be done in $O(nm)$ time. For each vertex $u \in V$, the betweenness centrality value $\overline{BC}(u)$ is computed using Equation 3.5 and $P(u)$ is obtained by $P(u) = \overline{BC}(u)/\binom{n}{2}$. Computation of all $\overline{BC}(u)$ and $P(u)$ values from the computed $\delta_u(v)$ values take $O(n^2)$ time.

Finally, in the end of the preprocessing step, $P(u, w)$ values are computed by using $P(u, w) = \overline{C}(u, w)/\binom{n}{2}$ for every pair of distinct vertices $u, w \in V$. To be able to compute $\overline{C}(u, w)$ values, let us consider two distinct vertices $s, t \in V$. Let us compute the fraction of the shortest paths from s to t that pass first through u and then through w. This fraction can be calculated by the product of two fractions: the fraction of the shortest paths from s to t that pass through w and the fraction of the shortest paths from s to w that pass through u. Adding all such fractions for every pair of distinct vertices $s, t \in V$, we obtain $\overline{C}(u, w)$. Mathematically, we have

$$\overline{C}(u, w) = \sum_{s \in V} \left(\sum_{t \in V | t \neq s} \frac{\sigma_{st}(w)}{\sigma_{st}} \right) \frac{\sigma_{sw}(u)}{\sigma_{sw}}$$

from which we obtain

$$\overline{C}(u, w) = \sum_{s \in V} \delta_s(w) \frac{\sigma_{sw}(u)}{\sigma_{sw}}. \tag{3.14}$$

Every $\overline{C}(u, w)$ can be computed by Equation 3.14 in $O(n)$ time. Therefore the total time to compute $\overline{C}(u, w)$ and $P(u, w)$ values for all pairs of distinct vertices $u, w \in V$ is $O(n^3)$. Overall the preprocessing step takes $O(n^3)$ time.

3.4.2 Upper Bounds

The upper bound in Equation 3.11 for the GBC of a subset $S \subseteq V$ of size k used in [92] can be rewritten for the normalized group betweenness centrality defined in Equation 3.12 as a bound of order two as

$$\overline{GBN}(S) \leq \textstyle\sum_{i \in S} P(i) - \sum_{j=1}^{k-1} P(s_j, s_{j+1}), \tag{KCBu}$$

where $s_1, s_2, \ldots, s_k$ is any ordering of the vertices in S. This upper bound is abbreviated as KCBu following the initials of the last names of the authors in [92] (for all of the following upper bounds, we use the initials of the last names of the authors who proposed them for the abbreviation). Even though this upper bound is equivalent to the upper bound in Equation 3.11, the dependencies of the betweenness and co-betweenness values on the group in consideration is successfully eliminated.

Consider a complete graph K_k of order k with vertices $s_1, s_2, \ldots, s_k$. Let the weight of the edge $\{u, w\}$ be $P(u, w)$. Finding the best upper bound in KCBu amounts to finding the maximum weighted subgraph that is a cycle containing all the vertices of K_k but from which exactly one edge is deleted. This problem can be transformed into the traveling salesman problem and is therefore hard to solve. The authors in [92] used complete enumeration in their computational experiments to find the ordering resulting in the maximum weight, i.e., resulting in the smallest upper bound. Note that any cycle containing all the vertices of K_k but from which exactly one edge is deleted is spanning, i.e., contains all the vertices of the graph, and is connected; and hence is a spanning tree of K_k. The upper bound in KCBu can be made stronger by using an upper bound of order two from the literature that finds a maximum weighted spanning tree in K_k and uses the corresponding weight in place of the term $\sum_{j=1}^{k-1} P(s_j, s_{j+1})$ [82, 152]. Clearly the weight of a maximum weighted spanning tree is never less than the weight of a maximum weighted cycle containing all the vertices of K_k but from which exactly one edge is deleted as the latter is also a spanning tree. Therefore the following bound, called as HW, which is based on the maximum weighted spanning tree computation is never weaker than the upper bound KCBu. For a given subset S of V, HW bound is computed as

$$\overline{GBN}(S) \leq \textstyle\sum_{i \in S} P(i) \; - \max_{T \in \Gamma} \sum_{\{i,j\} \in E_T} P(i, j) \tag{HW}$$

where Γ is the set of all spanning trees of the complete graph with vertex set S and edge weights $P(i, j)$ for $\{i, j\} \in E_T$ and E_T is the set of edges of T. Surprisingly, the problem of finding a maximum weighted spanning tree is much easier than the travelling salesman problem and the complexity of this upper bound is at most $O(k^2)$.

In [55], the authors define the following Bonferroni-type inequality for the union of a finitely many events for any $r \in \mathbb{N}$

$$(-1)^r P(\cup_{v\in S} E(v)) \geq (-1)^r \sum_{\substack{I \subseteq S \\ 0 < |I| \leq r}} (-1)^{|I|-1} P(\cap_{i\in I} E(i)) + \sum_{\substack{I \subseteq S \\ |I| = r+1}} P(\cap_{i\in I} E(i)) \frac{\sum_{i\in I} E(i)}{\sum_{v\in S} E(v)}. \tag{3.15}$$

When $r = 1$, we obtain from Inequality 3.15 the following upper bound of order two, called as DT, on the normalized GBC for a given subset S of V.

$$\overline{GBN}(S) \leq \sum_{i\in S} P(i) \; - \sum_{\{i,j\}\in S\times S, i<j} P(i,j) \; \frac{P(i)+P(j)}{\sum_{i\in S} P(i)} \tag{DT}$$

Note that the complexity of computing the upper bound DT is $O(k^2)$ for a given subset of vertices S of size k.

In [157], the authors propose two upper bounds, which we call as YATu1 and YATu2, for the probability of the union of finitely many events. For a given subset S of V, YATu1 and YATu2 are given as

$$\overline{GBN}(S) \leq \Big(\tfrac{1}{\min_{i\in S} c_i} + \tfrac{1}{\sum_{i\in S} c_i}\Big) \sum_{i\in S} c_i P(i) - \tfrac{1}{(\min_{i\in S} c_i) \sum_{i\in S} c_i} \sum_{i\in S} \sum_{j\in S} c_i c_j P(i,j), \tag{YATu1}$$

$$\begin{aligned}\overline{GBN}(S) \leq \min_{i\in S} \Big\{ \tfrac{\sum_{j\in S} c_j P(i,j) - (\min_{j\in S} c_j) P(i)}{\sum_{j\in S} c_j - \min_{j\in S} c_j} \Big\} + \Big(\tfrac{1}{\min_{i\in S} c_i} + \tfrac{1}{\sum_{i\in S} c_i - \min_{i\in S} c_i}\Big) \sum_{i\in S} c_i P(i) \\ - \tfrac{1}{(\min_{i\in S} c_i)(\sum_{i\in S} c_i - \min_{i\in S} c_i)} \sum_{i\in S} \sum_{j\in S} c_i c_j P(i,j),\end{aligned} \tag{YATu2}$$

where c_i is a positive real number for each $i \in S$. The authors prove that YATu2 dominates YATu1 for every choice of c_i's. According to some computational experiments using randomly generated c_i's, the authors claim that the best upper bounds are obtained when all c_i's are equal. Thus, we set each c_i to 1 in our computational experiments. Note that the upper bounds do not change if we multiply all c_i's by the same positive constant. The upper bounds YATu1 and YATu2 can be computed in $O(k^2)$ time.

In the search for different upper bounds of order two, it turned out that most of the bounds we found in the literature are dominated by HW in terms of quality. For instance, the upper bounds proposed in [93] and [97] are also appropriate for the problem considered in this study. However, it has been proved in [22] that these upper bounds are dominated by HW. Thus we only used KCBu, HW, DT, YATu1, and YATu2 in our computational experiments.

3.4.3 Lower Bounds

The lower bound for the GBC of a subset $S \subseteq V$ of size k used in [92] can be rewritten for the normalized group betweenness centrality as a bound of order two as

$$\textstyle\sum_{i\in S} P(i) - \sum_{\{i,j\}\in S\times S, i<j} P(i,j) \leq \overline{GBN}(S), \tag{KCBl}$$

which takes $O(k^2)$ time to compute once the betweenness and co-betweenness centrality values are given. This bound, abbreviated as KCBl, is the classical Bonferroni inequality of order two. There are stronger bounds in the literature with the same $O(k^2)$ time complexity. For all of the following lower bounds, we use the initials of the last names of the authors who proposed them for the abbreviation.

In [156], the authors propose a lower bound of order two, called as YAT, which is shown to be stronger than the classical Bonferroni bound of order two. For a given subset S of V of size k, YAT is computed as

$$\beta + \textstyle\sum_{i\in S} \alpha_i \left(\frac{1}{\chi\left(\frac{\gamma_i}{\alpha_i}\right)} - \frac{\frac{\gamma_i}{\alpha_i} - \chi\left(\frac{\gamma_i}{\alpha_i}\right)}{\left[1+\chi\left(\frac{\gamma_i}{\alpha_i}\right)\right]\left[\chi\left(\frac{\gamma_i}{\alpha_i}\right)\right]} \right) \leq \overline{GBN}(S), \tag{YAT}$$

where $\chi(.)$ is a function defined as

$$\chi(x) = \begin{cases} x-1 & \text{if } x \geq 2 \text{ is an integer,} \\ \lfloor x \rfloor & \text{otherwise,} \end{cases}$$

$\beta = \max\left\{0, \max_{i\in S}\left\{\sum_{j\in S} P(i,j) - (k-1)P(i)\right\}\right\}$, $\alpha_i = P(i) - \beta$, $\gamma_i = \sum_{j\in S} P(i,j) - k\beta$. This bound can be computed in $O(k^2)$ time.

Another bound that is stronger than the classical Bonferroni bound of order two is proposed in [96] which we call as KAT. Even though YAT is stronger than KAT in terms of quality, KAT may have better performance in terms of computational time. For a given subset S of V of size k, KAT is calculated as

$$\textstyle\sum_{i\in S} \left(\frac{\theta_i P(i)^2}{\sum_{j\in S} P(i,j) + (1-\theta_i)P(i)} + \frac{(1-\theta_i)P(i)^2}{\sum_{j\in S} P(i,j) - \theta_i P(i)} \right) \leq \overline{GBN}(S), \tag{KAT}$$

where $\theta_i = \frac{\sum_{j\in S, j\neq i} P(i,j)}{P(i)} - \left\lfloor \frac{\sum_{j\in S, j\neq i} P(i,j)}{P(i)} \right\rfloor$.

As a last lower bound, we consider the bound proposed in [66, 93], called as GK. To the best knowledge of the authors, there is no other lower bound of order two that is theoretically stronger than GK and that has a smaller time complexity. For a given subset S of V of size k, GK is computed as

$$\frac{(\sum_{i\in S}\delta_i P(i))^2}{\sum_{i,j\in S}\delta_i\delta_j P(i,j)} \leq \overline{GBN}(S), \tag{GK}$$

where $\delta_1,\ldots,\delta_k$ is the solution of the following system of linear equations

$$\begin{pmatrix} P(1) & P(1,2) & \cdots & P(1,k) \\ P(2,1) & P(2) & \cdots & P(2,k) \\ \vdots & \vdots & \vdots & \vdots \\ P(k,1) & P(k,2) & \cdots & P(k) \end{pmatrix} \begin{pmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \delta_k \end{pmatrix} = \begin{pmatrix} P(1) \\ P(2) \\ \vdots \\ P(k) \end{pmatrix}.$$

Although there are some other lower bounds of order two in the literature, we did not consider them in this study. This is because they are either computationally expensive or dominated by an already considered lower bound. For example, the lower bounds proposed in [45] and [46] are special cases of KAT and they are always less than or equal to KAT as it is proved in [96]. In [76], the author improves lower bounds proposed in [45], [46], and [96] by optimization over subsets of events. But the improved versions of these bounds are computationally expensive since they require exponential time in the group size k. Another bound is proposed in [39]. In [94], the authors proved that this bound is dominated by the bound proposed in [45]. Hence, it is dominated by KAT. In [157], the authors propose two lower bounds whose calculations require the solution of the knapsack problem. Even though these lower bounds dominate GK, we did not consider them in our computational experiments because the knapsack problem is an NP-hard problem. Therefore, we utilized only KCBl, YAT, KAT, and GK in this study.

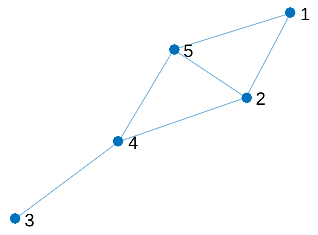


Figure 3.1: A graph with $n = 5$

3.4.4 Illustrative Example

In this subsection, we illustrate upper and lower bound calculations for the normalized GBC on a small graph with 5 vertices, $V = \{1, \dots, 5\}$, which is given in Figure 3.1. Betweenness and co-betweenness values are calculated by Equation 3.5 and Equation 3.14, respectively, are put in the matrix $\overline{C}$ which is given below.

$$\overline{C} = \begin{pmatrix} 4 & 2 & 1 & 2 & 2 \\ 2 & 5 & 1.5 & 3 & 1 \\ 1 & 1.5 & 4 & 4 & 1.5 \\ 2 & 3 & 4 & 7 & 3 \\ 2 & 1 & 1.5 & 3 & 5 \end{pmatrix}$$

Note that the diagonal entries of $\overline{C}$ are equal to the betweenness centrality values, i.e., $\overline{C}(i,i) = \overline{BC}(i)$.

The normalized values, i.e., $P(i)$ and $P(i,j)$ values, are given below in the matrix P.

$$P = \overline{C}/\binom{n}{2} = \begin{pmatrix} 0.4 & 0.2 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.5 & 0.15 & 0.3 & 0.1 \\ 0.1 & 0.15 & 0.4 & 0.4 & 0.15 \\ 0.2 & 0.3 & 0.4 & 0.7 & 0.3 \\ 0.2 & 0.1 & 0.15 & 0.3 & 0.5 \end{pmatrix}.$$

Consider the subset $S = \{2, 3, 4, 5\}$ of V. The complete graph with vertex set S and edge weights $P(i,j)$ is given in Figure 3.2. The maximum weighted subgraph that is a cycle containing all the vertices of the graph in Figure 3.2 but from which exactly one edge is deleted is 5-3-4-2. Thus,

$$KCBu = P(2) + P(3) + P(4) + P(5) - P(5,3) - P(3,4) - P(4,2) = 1.25.$$

On the other hand, the maximum weighted spanning tree consists of the edges $\{2,4\}$, $\{3,4\}$, and $\{4,5\}$. Therefore,

$$HW = P(2) + P(3) + P(4) + P(5) - P(2,4) - P(3,4) - P(4,5) = 1.1.$$

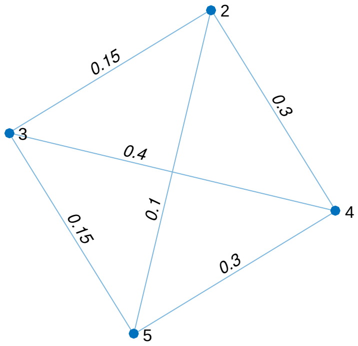


Figure 3.2: Complete graph with vertex set $S = \{2, 3, 4, 5\}$ and edge weights $P(i, j)$

We have that $\sum_{i \in S} P(i) = P(2) + P(3) + P(4) + P(5) = 2.1$ and hence

$$DT = 2.1 - P(2,3)\frac{P(2)+P(3)}{2.1} - P(2,4)\frac{P(2)+P(4)}{2.1} - \cdots - P(4,5)\frac{P(4)+P(5)}{2.1} = 1.37.$$

Also, $\sum_{i \in S} \sum_{j \in S} P(i, j) = P(2, 2) + P(2, 3) + \cdots + P(4, 5) + P(5, 5) = 4.9$. Thus,

$$YATu1 = \left(1 + \frac{1}{4}\right) 2.1 - \frac{1}{4} 4.9 = 1.4,$$

$$YATu2 = \min\left\{\frac{P(2,3)+P(2,4)+P(2,5)}{3}, \ldots, \frac{P(5,2)+P(5,3)+P(5,4)}{3}\right\} + \left(1 + \frac{1}{3}\right) 2.1 - \frac{1}{3} 4.9 = 1.35.$$

Similarly, we calculate lower bounds as $KCBl = 0.7$, $YAT = 0.93$, $KAT = 0.93$, and $GK = 0.97$. Using the upper and lower bounds, we can define the gap on the normalized GBC of S as

$$GAP = \min\{1, UpperBound\} - \max\{0, LowerBound\}. \tag{3.16}$$

For this small example HW results in the best upper bound while GK finds the best lower bound. By Equation 3.16, $GAP = 0.03$ meaning that the exact normalized GBC value of S is estimated between 0.97 and 1 with a 3% gap.

3.4.5 An Algorithm for the Group with the Highest GBC Value

For an efficient search of a group of size k with the highest GBC value from among a given set of groups of size k, we propose Algorithm 10. It is based on the idea that a group cannot be the best group in the given set, i.e., the one with the highest GBC value, if there exists another group in the set whose lower bound is greater than the upper bound of the group in consideration. Thus, the algorithm executes the following two-step procedure: (1) find lower and upper bounds on the GBC of the given groups, (2) eliminate the groups having an upper bound that is lower than the maximum lower bound and find the remaining groups which are candidates for the best group in the set. Algorithm 10 in addition to returning the gap also returns the optimality gap for each group in the candidate list. The optimality gap of a group provides an upper bound on the difference between the GBC value of the best group in the given set and the GBC value of the group in consideration. Hence, the optimality gap shows the user the quality of the approximation. The details of the algorithm are shown in Algorithm 10.

For a given graph $G = (V, E)$ and group size k, our algorithm can be applied for different purposes. For example, if k is small, then all possible subsets of V of size k can be generated with the aim of finding bounds on the GBC of each subset. On the other hand, in a road network, one may have candidate locations, $T \subseteq V$ where the roadside equipments can be installed. In this case, one may generate all subsets of T of size k for the purpose of choosing among the candidate locations. As a third example, if the user has a predetermined set of candidate groups, obtained for example as a result of another computational experiment or expert opinion, then s/he may give this set as an input to estimate their GBC values. Consider a case where there is a cost associated with each vertex and the cost of a group cannot exceed a given budget. In this case, the user may do some preliminary computations to determine the input set that will be given to the algorithm. Note that our algorithm after the preprocessing step can find bounds on the GBC of any group irrespective of its size. For the purpose of simplicity, we assume that all the subsets given as inputs to the algorithm are of the same size k.

The running time of Algorithm 10 is $O(n^3 + sk^2)$, where s is the number of groups in

Algorithm 10

```
Input: Subsets $S_1, \ldots, S_s$ of $V$ of size $k$.
Preprocessing: Compute the matrix $P = [P(i, j)]$.
for $i = 1$ to $s$ do
    Find a lower bound on the GBC of $S_i$, $LB(S_i)$.
    Find an upper bound on the GBC of $S_i$, $UB(S_i)$.
end for
Find the group $S_{i^*}$ with maximum lower bound, where $i^* = \text{argmax}_i\{LB(S_i)\}$.
Find the group $S_{j^*}$ with maximum upper bound, where $j^* = \text{argmax}_i\{UB(S_i)\}$.

Initialize the list of candidate groups for the best group, $C = \{S_{i^*}\}$.
for $i = 1$ to $s$ do
    if $UB(S_i) \geq LB(S_{i^*})$ then
        Add $S_i$ to the candidate list, $C = C \cup \{S_i\}$.
        Find gap for $S_i$, $GAP(S_i) = \min\{1, UB(S_i)\} - \max\{0, LB(S_i)\}$.
        if $i \neq j^*$ then
            Calculate the optimality gap for $S_i$,
            $optGAP(S_i) = \min\{1, UB(S_{j^*})\} - \max\{0, LB(S_i)\}$.
        else
            Find the group $S_{j^{**}}$ with the second maximum upper bound,
            where $j^{**} = \text{argmax}_{i \in \{1,2,\ldots,s\} \setminus \{j^*\}}\{UB(S_i)\}$.
            Calculate the optimality gap for $S_i$,
            $optGAP(S_i) = \max\{0, \min\{1, UB(S_{j^{**}})\} - \max\{0, LB(S_i)\}\}$.
        end if
    end if
end for
return $C$, $GAP$, $optGAP$.
```

the input set. If there are a polynomial number of groups (in n) in the input set, then the algorithm runs in polynomial time. On the other hand, if there are an exponential number of groups in the input set, then the algorithm runs in exponential time.

Algorithm 10 returns all groups that are candidates for the optimal group. One may compute the exact GBC values of these groups, for example with the method proposed by [125], in order to find the optimal group. Note that if the method proposed in [125] is used, the evaluation of the exact GBC for each group takes $O(k^3)$ time and no additional preprocessing will be needed as the preprocessing steps of the method in [125] and Algorithm 10 are the same. One may also device branch-and-bound like procedures to arrive at the optimal group without computing the exact GBC values of all the groups in the candidate list C.

3.5 Computational Experiments

3.5.1 Selection of Bounds

Selection of upper and lower bounds to be used in Algorithm 10 is crucial in order to have good estimates of the GBC values, i.e., in order to have small gaps. In this subsection, we illustrate successive lower and upper bound calculations on the GBC of several groups on three small networks. The first network is the Zachary's karate club network [159]. In his study, Zachary collected information about 34 members of a university karate club, i.e., $n = 34$. An edge between two vertices indicates that the corresponding members interact with each other. Figure 3.3 graphically displays the Zachary's karate club network.

The second network is randomly generated by implementing a preferential attachment procedure for the network growth. The procedure starts with a graph with two vertices and an edge between them and adds vertices one by one until the desired number of vertices is achieved. In each step, the procedure attaches the new vertex to e existing vertices with a probability which is proportional to the degrees of the existing vertices. The number of edges to be added in each step, e, affects the density of the generated graph. We generated a small network with 30 vertices by adding 2 edges in each step. A graphical display of the preferential attachment network generated is given in Figure 3.4.

The last network is randomly generated by the Erdös-Renyi graph model. This model takes two inputs. The first input is the number of vertices, n. The second one is the probability of adding an edge between vertex $i \in \{1, \ldots, n\}$ and vertex $j \in$

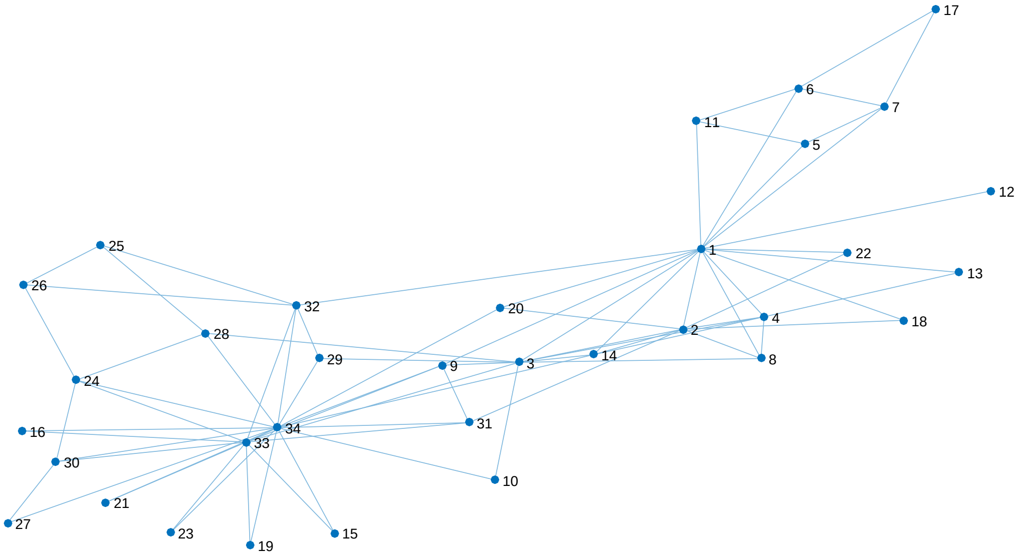


Figure 3.3: Zachary's karate club network

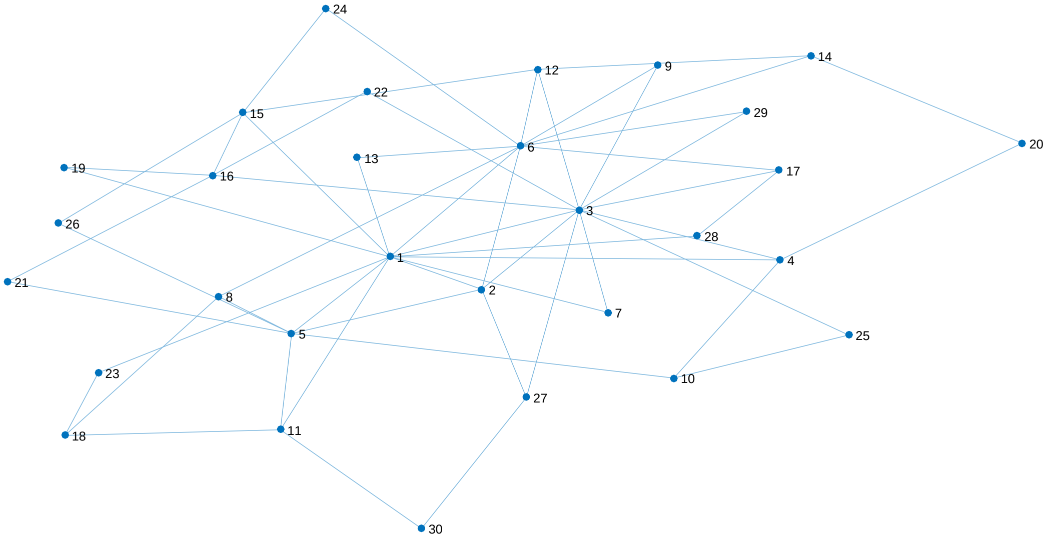


Figure 3.4: Randomly generated preferential attachment network with $n = 30$ and $e = 2$, PA

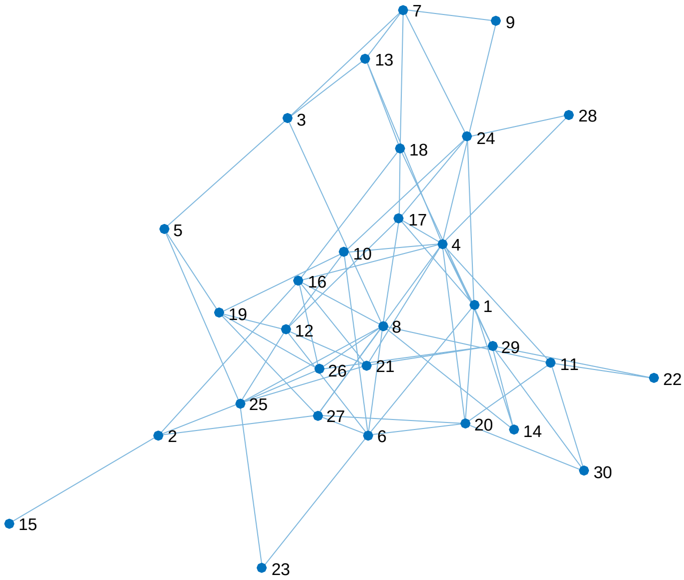


Figure 3.5: Randomly generated Erdös-Renyi graph with $n = 30$ and $p = 0.2$, ER

$\{1, \ldots, n\}$, p. This input can be considered as the density parameter. As p goes to 1 from 0, the graph becomes denser. By taking $n = 30$ and $p = 0.2$, we generated an Erdös-Renyi graph which is given in Figure 3.5.

We calculated the lower and upper bounds provided in Section 3.4 for all possible groups of size 3 through 7, i.e., $k = 3, 4, 5, 6, 7$, for three networks defined above. For each group, we found the best lower bound (i.e., the highest) and the best upper bound (i.e., the lowest). Table 3.1 reports the number of possible groups for each network and group size combination, and the performances of different bounds in terms of the percentage of finding the best bounds. Also, Table 3.2 provides the total computational time spent to find the lower and upper bounds for all of the groups for each network and group size combination. Lower bounds YAT and KAT have similar computational times and give the same bounds for almost all groups outperforming KCBl and GK. In three network and group size combination, YAT beats KAT in terms of the percentage of finding the best bound. In our further computational experiments, we will use YAT since it is both experimentally and theoretically better than KAT [156]. When the group size is 3, upper bounds KCBu and HW find the same bounds for all the groups. For other group sizes, HW outperforms KCBu and gives the strongest

Table 3.1: The performances of different bounds on the Zachary, and randomly generated preferential attachment (PA) and Erdös-Renyi (ER) graphs in terms of the percentage of finding the best bounds for different group sizes

	Group	# of	Lower bounds				Upper bounds				
Network	size	groups	KCBl	YAT	KAT	GK	KCBu	HW	DT	YATu1	YATu2
Zachary	3	5984	98.88	100.00	98.88	0.00	100.00	100.00	14.84	14.84	29.71
	4	46376	95.55	99.99	99.99	0.01	48.95	100.00	5.58	5.58	5.61
	5	278256	89.49	100.00	100.00	0.00	18.64	100.00	2.09	2.09	2.09
	6	1344904	81.63	99.99	99.99	0.01	5.93	100.00	0.75	0.75	0.75
	7	5379616	73.28	99.98	99.98	0.02	1.64	100.00	0.25	0.25	0.25
PA	3	4060	99.61	100.00	99.61	0.00	100.00	100.00	8.72	8.72	28.13
	4	27405	97.16	100.00	100.00	0.00	57.93	100.00	1.64	1.64	1.64
	5	142506	91.03	100.00	100.00	0.00	26.84	100.00	0.24	0.24	0.24
	6	593775	80.82	100.00	100.00	0.00	9.79	100.00	0.02	0.02	0.02
	7	2035800	66.88	100.00	100.00	0.00	2.80	100.00	0.00	0.00	0.00
ER	3	4060	99.21	100.00	99.21	0.00	100.00	100.00	2.66	2.66	13.72
	4	27405	98.17	100.00	100.00	0.00	67.60	100.00	0.24	0.24	0.24
	5	142506	95.50	100.00	100.00	0.00	37.12	100.00	0.01	0.01	0.01
	6	593775	88.60	100.00	100.00	0.00	17.46	100.00	0.00	0.00	0.00
	7	2035800	77.22	100.00	100.00	0.00	7.24	100.00	0.00	0.00	0.00

bounds. Moreover, computational time of HW is much better than that of KCBu. Even though computational times of other upper bounds are better than that of HW, they are worse than HW in terms of the quality. Thus we will utilize HW as the upper bound in the further computational experiments.

Following the procedure in Algorithm 10, i.e., finding bounds on the GBC of groups and eliminating groups based on the bounds, to compare the performance of the selected YAT-HW pair with that of KCBl-KCBu pair in [92], we provide the number of remaining groups after the elimination step and the total time spent to find the lower and upper bounds for all of the possible groups in Table 3.3. The columns 6 and 7 of Table 3.3, are the sum of the columns 4 and 8, and the sum of the columns 3 and 7 of Table 3.2, respectively. The preprocessing steps of Algorithm 10 take approximately 0.016, 0.012, and 0.013 seconds for Zachary, PA, and ER networks, respectively. Theoretically, YAT and HW outperform KCBl and KCBu, respectively. We confirm that YAT-HW pair outperform KCBl-KCBu pair computationally as well. The number of remaining groups is larger for KCBl-KCBu bounds. Moreover, as the group

Table 3.2: The total computational time of different bounds on the Zachary, and randomly generated preferential attachment (PA) and Erdös-Renyi (ER) graphs for all of the groups with different group sizes

	Group	Total time (in s) for lower bounds				Total time (in s) for upper bounds				
Network	size	KCBl	YAT	KAT	GK	KCBu	HW	DT	YATu1	YATu2
Zachary	3	0.02	0.03	0.02	0.02	0.15	0.05	0.02	0.01	0.02
	4	0.12	0.21	0.15	0.17	1.69	0.49	0.17	0.10	0.18
	5	0.74	1.25	0.89	1.06	15.14	3.64	1.03	0.72	1.20
	6	3.64	6.06	4.33	5.12	125.00	21.04	4.96	3.70	6.19
	7	14.97	24.90	17.80	21.10	1183.11	99.37	20.75	15.86	26.79
PA	3	0.02	0.03	0.02	0.03	0.16	0.05	0.02	0.01	0.02
	4	0.11	0.14	0.12	0.14	1.04	0.31	0.11	0.07	0.11
	5	0.40	0.72	0.48	0.57	8.10	1.97	0.57	0.39	0.66
	6	1.72	2.91	2.08	2.48	56.70	9.83	2.27	1.74	2.93
	7	5.90	9.70	6.94	8.30	440.26	36.11	7.62	5.83	9.83
ER	3	0.02	0.03	0.02	0.02	0.16	0.05	0.02	0.01	0.02
	4	0.11	0.18	0.13	0.15	1.01	0.31	0.10	0.07	0.11
	5	0.39	0.64	0.46	0.54	7.72	1.84	0.53	0.37	0.62
	6	1.63	2.69	1.92	2.28	56.33	9.25	2.26	1.65	2.81
	7	5.86	9.44	6.88	7.94	441.30	37.16	7.67	6.00	10.18
Total time		35.63	58.91	42.24	49.92	2337.88	221.46	48.08	36.55	61.68

size increases, the time for KCBl-KCBu pair increases more rapidly than the time for YAT-HW pair. This is mainly because of the complete enumeration used in the KCBu calculations. Based on Table 3.3, we can conclude that YAT-HW pair is more efficient than KCBl-KCBu pair in terms of both solution quality (i.e., the number of remaining groups) and computational time. Furthermore, in the last column of Table 3.3, the computation time to find the lower and upper bounds for all of the groups with the original method proposed in [92] is provided. Here the code shared by the authors of [92] is used. By comparing the last two columns of Table 3.3, the impact of the preprocessing step we introduced can be seen. Also, looking at the columns with the headers YAT-HW (fourth and sixth columns) and KCBl-KCBu (fifth and seventh columns), the effects of using the stronger bounds can be observed. We can conclude that both the introduction of the preprocessing step and the use of the improved bounds have significant contributions in the reduction of the computational time.

For the Zachary's network, by using YAT and HW in bound calculations, only the group {1, 33, 34} remains after the elimination step when $k = 3$. Thus, the group

Table 3.3: The number of possible groups; the number of remaining groups after elimination by YAT-HW and KCBl-KCBu bounds; the total time spent for YAT-HW and KCBl-KCBu bound calculations; and the total time spent for KCBl-KCBu bound calculations with the original method in [92] on the Zachary, and randomly generated preferential attachment (PA) and Erdös-Renyi (ER) graphs for different group sizes

Network	Group size	Number of groups	Number of remaining groups		Total time (in s)		Total time (in s) for the method in [92]
			YAT-HW	KCBl-KCBu	YAT-HW	KCBl-KCBu	
Zachary	3	5984	1	1	0.08	0.17	161.97
	4	46376	8	24	0.69	1.82	1154.53
	5	278256	202	643	4.89	15.88	7054.35
	6	1344904	3422	10301	27.10	128.65	33439.85
	7	5379616	37148	132138	124.26	1198.07	134224.18
PA	3	4060	1	1	0.08	0.17	72.45
	4	27405	1	4	0.45	1.15	469.92
	5	142506	69	193	2.69	8.50	2361.80
	6	593775	1855	4788	12.74	58.42	9764.25
	7	2035800	23475	66238	45.81	446.16	30515.88
ER	3	4060	1	1	0.08	0.18	70.81
	4	27405	10	11	0.48	1.11	478.87
	5	142506	138	170	2.49	8.11	2484.75
	6	593775	1259	2149	11.94	57.96	10359.22
	7	2035800	10828	23114	46.59	447.16	35481.35

with the highest GBC, i.e., the optimal group, consists of members 1, 33, and 34. In Figure 3.3, it can be seen that these members are among the key vertices in the graph. Also, lower and upper bounds for this group are 0.846 and 0.854, respectively. Thus, the GBC value of $\{1, 33, 34\}$ is estimated to be at least as 0.846 with 0.8% gap. By using the exact algorithm taking $O(k^3)$ time in [125], the GBC of $\{1, 33, 34\}$ is found as 0.846. When we increase the group size to 4, the results are still quite good. Only 8 groups among 46376 groups remain as candidates for the optimal group.

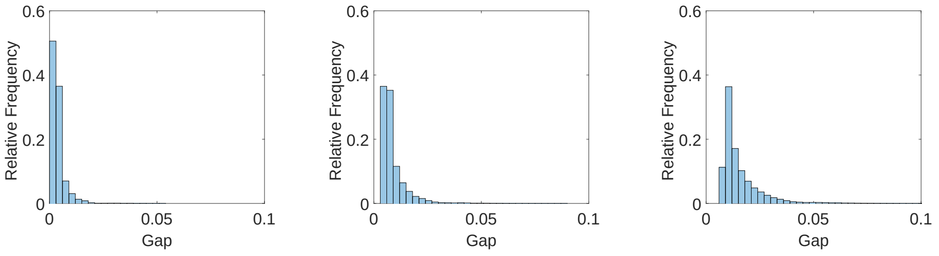
For the PA network, when $k = 3$ only the group $\{1, 3, 6\}$ remains after the elimination step and its GBC value is estimated to be at least 0.784 with 1.5% gap. With the method in [125], the GBC value is calculated as 0.786. When $k = 4$, we are also able to find the optimal group which is $\{1, 3, 5, 6\}$. The estimated GBC value is at least 0.872 with 4.5% gap where the exact value is 0.887. In Figure 3.4, the vertices 1, 3, 5, and 6 seem to be central vertices as it is expected.

Table 3.4: The mean and median gaps for YAT-HW and KCBl-KCBu bound pairs; and the mean gaps between the selected bounds (YAT and HW) and the exact values on the Zachary, and randomly generated preferential attachment (PA) and Erdös-Renyi (ER) graphs for all of the groups with different group sizes

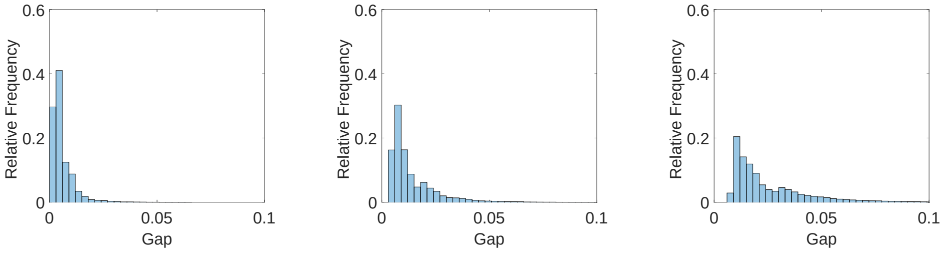
Network	Group size	Mean Gap		Median Gap		Mean Gap	
		YAT-HW	KCBl-KCBu	YAT-HW	KCBl-KCBu	YAT-Exact	HW-Exact
Zachary	3	0.002	0.003	0.002	0.002	0.000	0.002
	4	0.008	0.012	0.006	0.008	0.002	0.006
	5	0.017	0.027	0.013	0.019	0.005	0.013
	6	0.031	0.050	0.024	0.036	0.009	0.022
	7	0.048	0.080	0.040	0.060	0.015	0.033
PA	3	0.003	0.003	0.002	0.002	0.001	0.003
	4	0.010	0.013	0.008	0.009	0.002	0.008
	5	0.022	0.029	0.018	0.022	0.005	0.017
	6	0.038	0.053	0.033	0.042	0.011	0.028
	7	0.060	0.086	0.053	0.073	0.018	0.042
ER	3	0.003	0.003	0.002	0.002	0.001	0.003
	4	0.011	0.012	0.010	0.010	0.002	0.009
	5	0.024	0.027	0.022	0.024	0.005	0.018
	6	0.042	0.050	0.039	0.043	0.011	0.031
	7	0.066	0.079	0.062	0.070	0.018	0.047

The optimal group is {4, 8, 29} for the ER network when $k = 3$. The estimated GBC value is at least 0.581 with 0.4% gap where the exact GBC value is found as 0.581. The vertices 4, 8, and 29 are among the central vertices as it can be seen in Figure 3.5.

Table 3.4 shows the mean and median gaps for YAT-HW and KCBl-KCBu bound pairs for different group sizes. For a given group size and a network, median values are smaller than the mean values for both YAT-HW and KCBl-KCBu bound pairs showing that the gap distributions of both pairs are right skewed. In other words, obtaining small gaps are more likely than large gaps for both YAT-HW and KCBl-KCBu bound pairs. As an illustrative example, in Figure 3.6 the distribution of gaps for YAT-HW and KCBl-KCBu bound pairs for different group sizes on the Zachary's network is provided. It can also be verified from Figure 3.6 that the distributions of the gaps are right skewed. If we investigate the first and second rows of Figure 3.6 separately, it can be seen that the gaps tend to increase as the group size increases



(a) YAT-HW, $k = 4$ (b) YAT-HW, $k = 5$ (c) YAT-HW, $k = 6$



(d) KCBl-KCBu, $k = 4$ (e) KCBl-KCBu, $k = 5$ (f) KCBl-KCBu, $k = 6$

Figure 3.6: Distribution of gaps for YAT-HW and KCBl-KCBu bound pairs for different group sizes on the Zachary's network

regardless of the lower-upper bound pair in use. Same conclusion can also be obtained from Table 3.4. To see the effect of lower-upper bound selection, Figure 3.6 can be analyzed column by column. The gaps found by YAT-HW are smaller than those found by KCBl-KCBu for each group size. The same conclusion can also be drawn from Table 3.4. We can conclude that YAT-HW pair is better than KCBl-KCBu pair once again. Thus, we use YAT and HW in Steps 4 and 5 of Algorithm 10, respectively. The last two columns of Table 3.4 report the mean gaps between the selected lower and upper bounds and the exact GBC values calculated by the method in [125] for different networks and group sizes, respectively. The mean values corresponding to the gaps between the exact values and the YAT bounds are smaller than the values for the gaps between the exact values and the HW bounds. It is an indicator of the fact that the exact values are closer to the lower bounds for all different cases provided in Table 3.4. If the user desires Algorithm 10 to return a single group instead of a list of candidate groups we can return the group having the highest lower bound.

3.5.2 Randomly Generated Graphs

To see the effects of graph size and graph density on the computational results, we made experiments with different randomly generated graphs. We generated nine different graphs using the Erdös-Renyi graph model by selecting $n = 40$, $n = 60$, and $n = 80$; and $p = 0.1$, $p = 0.3$, and $p = 0.5$. $ER_{n,p}$ represents the Erdös-Renyi graph with n vertices whose edges are added with probability p. Also, we constructed six different graphs with the preferential attachment procedure by setting $n = 40$, $n = 60$, and $n = 80$; and $e = 2$ and $e = 5$. We will use the notation $PA_{n,e}$ to represent the preferential attachment graph with n vertices and e edges added in each step. The details of the Erdös-Renyi graph model and the preferential attachment procedure are given in Section 3.5.1.

Table 3.5 reports the number of remaining groups after the elimination step and the total time to find the lower and upper bounds for all possible groups for different group sizes on different graphs. It should be noted that the time it takes for the preprocessing step for each graph is smaller than 0.1 seconds. As it can be seen from the table, the number of remaining groups is small for all graph-group size combinations. Indeed, we can find the optimal group for some combinations. For a given graph, the number of remaining groups increases in general with the group size meaning that finding the optimal group becomes harder for larger group sizes. Moreover, it can be concluded from the table that for a given group size, the computational time is not much affected by the density of the graph, but the graph size, i.e., the number of vertices, is an important factor for the time. Also, the computational time increases with the group size.

Table 3.6 shows the mean and median gaps for all graphs and group sizes. Gaps are very small for all graph-group size combinations. Also, we can conclude that for a given graph, gaps increase with the group size as the mean and median values increase. In addition, regardless of the graph and the group size, distributions of the gaps tend to be right skewed since the median values are smaller than or equal to the mean values.

As discussed before, the lower bounds computed by YAT-HW bound pair are closer to

Table 3.5: The number of remaining groups after elimination by YAT-HW bound and the total time spent for YAT-HW bound calculations on randomly generated different Erdös-Renyi and preferential attachment graphs for all groups with different k values

	Number of remaining groups			Total time (in s)		
Graph	$k=3$	$k=4$	$k=5$	$k=3$	$k=4$	$k=5$
$ER_{40,0.1}$	3	2	18	0.33	3.05	21.07
$ER_{40,0.3}$	1	2	15	0.30	2.96	19.42
$ER_{40,0.5}$	9	78	1263	0.28	2.92	21.60
$ER_{60,0.1}$	2	1	1	1.05	15.36	172.24
$ER_{60,0.3}$	1	2	23	1.00	14.70	165.15
$ER_{60,0.5}$	2	11	227	1.04	14.49	172.52
$ER_{80,0.1}$	1	1	5	2.38	56.55	809.79
$ER_{80,0.3}$	1	2	42	2.52	50.04	851.48
$ER_{80,0.5}$	1	23	535	2.55	51.68	835.92
$PA_{40,2}$	1	12	187	0.31	2.63	20.53
$PA_{40,5}$	2	5	23	0.29	2.87	21.77
$PA_{60,2}$	1	1	29	1.01	13.67	170.09
$PA_{60,5}$	2	3	22	1.04	17.34	170.11
$PA_{80,2}$	1	77	878	2.33	45.30	709.23
$PA_{80,5}$	2	5	6	2.38	48.56	773.08

the exact GBC values than the upper bounds. With this observation, Algorithm 10 can be modified to return a single group which is the group with the highest lower bound together with the gap and optimality gap of the returned group. Table 3.7 provides the GBC values of the best group returned by (the modified) Algorithm 10 for each graph and group size combination. 'Estimated' columns are the lower bounds on the GBC values of the group returned by Algorithm 10 with the gaps (i.e., the difference between the upper bound and the lower bound) provided in the parenthesis while 'Exact' columns are the GBC values of the returned groups calculated by the method in [125]. 'optGAP' columns are the optimality gaps of the returned groups. The estimated values are very close to the exact values. Indeed, they are the same most of the time. The optimality gaps are also very small in all the cases. Also, it should be noted that the GBC values of the optimal groups are the same with the GBC values of the groups returned by Algorithm 10 except the group for $PA_{40,2}$ when $k = 5$. For this graph-group size combination the most central group of vertices has a GBC

Table 3.6: The mean and median gaps for YAT-HW bound pair on randomly generated different Erdös-Renyi and preferential attachment graphs for all of the groups with different group sizes k

	Mean Gap			Median Gap		
Graph	$k=3$	$k=4$	$k=5$	$k=3$	$k=4$	$k=5$
$ER_{40,0.1}$	0.002	0.007	0.015	0.001	0.006	0.013
$ER_{40,0.3}$	0.001	0.004	0.009	0.001	0.004	0.008
$ER_{40,0.5}$	0.002	0.005	0.011	0.001	0.004	0.010
$ER_{60,0.1}$	0.001	0.003	0.006	0.001	0.002	0.005
$ER_{60,0.3}$	0.001	0.002	0.004	0.001	0.002	0.003
$ER_{60,0.5}$	0.001	0.002	0.005	0.001	0.002	0.004
$ER_{80,0.1}$	0.000	0.001	0.003	0.000	0.001	0.003
$ER_{80,0.3}$	0.000	0.001	0.002	0.000	0.001	0.002
$ER_{80,0.5}$	0.000	0.001	0.003	0.000	0.001	0.003
$PA_{40,2}$	0.002	0.006	0.014	0.001	0.005	0.011
$PA_{40,5}$	0.002	0.005	0.010	0.001	0.004	0.009
$PA_{60,2}$	0.001	0.003	0.007	0.001	0.002	0.005
$PA_{60,5}$	0.001	0.002	0.005	0.001	0.002	0.004
$PA_{80,2}$	0.000	0.001	0.003	0.000	0.001	0.002
$PA_{80,5}$	0.000	0.001	0.003	0.000	0.001	0.002

value of 0.861. Although we have returned a group different than the optimal group, there is only 0.4% difference between the GBC values of the returned group and the optimal group. Based on Table 3.7, we can conclude that Algorithm 10 returns the optimal group for most of the time and the lower bound returned estimates the GBC value quite well. When the algorithm is unable to return the optimal group, the group returned by the algorithm is nearly optimal (i.e., the optimality gap is very small).

Time for finding bounds for all possible groups increases dramatically with the group size. Therefore, for large values of k, we randomly selected 10000 groups and found the gaps by calculating the bounds just for those groups. The mean and median gaps for large k values on different graphs are provided in Table 3.8. For each graph-group size combination, we can provide bounds with quite small gaps in nearly 1 second. Like the conclusions obtained from Table 3.6, we can say that the gaps increase with the group size for a given graph and the distributions of the gaps are right skewed.

Table 3.7: The estimated and exact GBC values, the gap and the optimality gap for the best group returned by Algorithm 10 for randomly generated different Erdös-Renyi and preferential attachment graphs for different group sizes k

	k=3			k=4			k=5		
Graph	Est. *(GAP)*	Exact	$optGAP$	Est. *(GAP)*	Exact	$optGAP$	Est. *(GAP)*	Exact	$optGAP$
$ER_{40,0.1}$	0.563 (0.003)	0.563	0.007	0.681 (0.017)	0.681	0.020	0.757 (0.033)	0.758	0.037
$ER_{40,0.3}$	0.253 (0.001)	0.253	0.000	0.323 (0.004)	0.323	0.000	0.389 (0.008)	0.389	0.007
$ER_{40,0.5}$	0.206 (0.001)	0.206	0.002	0.267 (0.005)	0.267	0.005	0.326 (0.009)	0.327	0.011
$ER_{60,0.1}$	0.327 (0.001)	0.327	0.001	0.411 (0.003)	0.411	0.000	0.489 (0.006)	0.489	0.000
$ER_{60,0.3}$	0.168 (0.001)	0.168	0.000	0.218 (0.002)	0.219	0.002	0.266 (0.003)	0.266	0.004
$ER_{60,0.5}$	0.137 (0.001)	0.137	0.000	0.180 (0.002)	0.180	0.001	0.221 (0.003)	0.221	0.004
$ER_{80,0.1}$	0.223 (0.000)	0.223	0.000	0.283 (0.001)	0.283	0.000	0.339 (0.005)	0.339	0.003
$ER_{80,0.3}$	0.117 (0.000)	0.117	0.000	0.154 (0.001)	0.154	0.000	0.189 (0.002)	0.189	0.002
$ER_{80,0.5}$	0.103 (0.000)	0.103	0.000	0.135 (0.001)	0.135	0.001	0.166 (0.002)	0.166	0.003
$PA_{40,2}$	0.744 (0.010)	0.747	0.000	0.798 (0.029)	0.808	0.033	0.847 (0.051)	0.857	0.056
$PA_{40,5}$	0.403 (0.007)	0.403	0.003	0.492 (0.018)	0.492	0.005	0.565 (0.032)	0.565	0.032
$PA_{60,2}$	0.614 (0.002)	0.614	0.000	0.716 (0.015)	0.716	0.000	0.760 (0.030)	0.772	0.033
$PA_{60,5}$	0.316 (0.005)	0.316	0.001	0.405 (0.011)	0.405	0.004	0.480 (0.019)	0.481	0.024
$PA_{80,2}$	0.785 (0.054)	0.785	0.000	0.838 (0.088)	0.842	0.058	0.875 (0.099)	0.879	0.107
$PA_{80,5}$	0.375 (0.004)	0.375	0.005	0.454 (0.015)	0.454	0.004	0.521 (0.026)	0.521	0.025

Table 3.8: The mean and median gaps for YAT-HW bound pair on randomly generated different Erdös-Renyi and preferential attachment graphs for randomly selected 10000 groups with different large group sizes k

	Mean Gap				Median Gap			
Graph	$k = 7$	$k = 8$	$k = 9$	$k = 10$	$k = 7$	$k = 8$	$k = 9$	$k = 10$
$ER_{40,0.1}$	0.042	0.061	0.083	0.108	0.039	0.058	0.081	0.106
$ER_{40,0.3}$	0.026	0.039	0.054	0.072	0.024	0.037	0.053	0.071
$ER_{40,0.5}$	0.029	0.042	0.058	0.077	0.029	0.043	0.058	0.077
$ER_{60,0.1}$	0.016	0.023	0.031	0.042	0.015	0.022	0.030	0.041
$ER_{60,0.3}$	0.011	0.017	0.024	0.032	0.011	0.016	0.023	0.031
$ER_{60,0.5}$	0.013	0.019	0.026	0.034	0.013	0.019	0.026	0.034
$ER_{80,0.1}$	0.008	0.011	0.016	0.020	0.007	0.011	0.015	0.020
$ER_{80,0.3}$	0.006	0.010	0.013	0.018	0.006	0.009	0.013	0.018
$ER_{80,0.5}$	0.007	0.011	0.015	0.019	0.007	0.011	0.015	0.019
$PA_{40,2}$	0.037	0.054	0.074	0.097	0.032	0.048	0.067	0.089
$PA_{40,5}$	0.028	0.041	0.056	0.075	0.025	0.038	0.053	0.071
$PA_{60,2}$	0.018	0.027	0.037	0.049	0.015	0.023	0.033	0.044
$PA_{60,5}$	0.013	0.018	0.025	0.034	0.011	0.016	0.023	0.030
$PA_{80,2}$	0.009	0.012	0.017	0.023	0.007	0.010	0.014	0.019
$PA_{80,5}$	0.007	0.010	0.014	0.018	0.006	0.009	0.012	0.016

Table 3.9: The number of all possible groups formed with the 20 nodes with the highest betweenness values; the number of remaining groups after elimination by YAT-HW bound pair; the total time spent for YAT-HW bound calculations; mean and median gaps for YAT-HW bound pair; the estimated and exact GBC values together with the optimality gaps for the best group returned by Algorithm 10; and the exact GBC value of the group with the highest GBC value on the Facebook network for different group sizes k

Group size	# of groups	Remaining groups	Total time (in s)	Gap		GBC of group by Algorithm 10			GBC of opt. group
				Mean	Median	Est. *(GAP)*	Exact	$optGAP$	
3	1140	3	0.037	0.002	0.001	0.790 *(0.027)*	0.790	0.024	0.790
4	4845	7	0.116	0.007	0.003	0.856 *(0.113)*	0.874	0.019	0.874
5	15504	97	0.377	0.017	0.010	0.873 *(0.114)*	0.891	0.127	0.900
6	38760	879	1.074	0.030	0.022	0.875 *(0.132)*	0.902	0.125	0.910
7	77520	4896	2.270	0.049	0.039	0.876 *(0.145)*	0.907	0.124	0.917

3.5.3 Real-life Networks

To measure the performance of Algorithm 10 on large scale networks, we utilize three real-life networks from [101]. The first network is named as Social Circles: Facebook. We will refer to this network as the Facebook network in the rest of the study. The network contains 4039 vertices and 88234 edges. Each vertex represents an individual and if two individuals are friends on Facebook there is an undirected edge between the corresponding vertices.

The number of vertices of the Facebook network is quite high to compute the bounds on the GBC of all possible groups. After the computation of $P(i, j)$ values in the preprocessing step taking approximately 80 minutes, we find the 20 vertices with the highest betweenness centrality values, $P(i)$, and use those vertices as the vertices from which the best group will be chosen. Table 3.9 reports the results for the Facebook network. Columns 2, 3, and 4 represent the number of possible groups, the number of remaining groups after the elimination step, and the total computational time to find bounds for all groups, respectively. The mean and median gaps are stored in columns 5 and 6, respectively. Moreover, for the best group -the one with the highest lower bound- returned by Algorithm 10, the estimated GBC values with gaps (given in parenthesis), the exact GBC values, and the optimality gaps are provided in columns 7, 8, and 9, respectively. In the last column, the GBC value of the group with the highest GBC is provided. Similar to the results obtained in the previous sections, we can make the following observations.

- The number of remaining groups as candidates for the best group are quite small when they are compared with the number of all groups. As the group size increases, finding the best group becomes harder as the number of remaining groups increases.
- The total computational time increases with the group size.
- Gaps are very small and tend to be right skewed as the median values are smaller than the mean values.
- For the best group returned by Algorithm 10, the estimated and exact GBC values are very close meaning that Algorithm 10 makes good estimations on the GBC values without calculating them exactly. Moreover, the optimality gaps are very small.
- Even though Algorithm 10 could not return the group with the highest GBC value in some cases, the GBC value of the group returned is not very far away from the optimal value.

The second real-life network is called as High Energy Physics - Theory Collaboration Network which will be referred as the HepTh network in the rest of the study. The original network contains 9877 nodes and 25998 edges. Each node represents an author who submitted a paper to High Energy Physics - Theory category between January 1993 and April 2003 (124 months). If two authors co-authored a paper there is an undirected edge between the corresponding nodes. Since the original network is disconnected, we utilized the largest connected component of the network. Also, we did not include the edges that connect a vertex to itself, i.e., loops. Finally, we obtained a network consisting of 8638 nodes and 24806 edges.

Like the Facebook network, given a group size $k \geq 3$, the number of all possible groups of size k is quite high for the HepTh network. After the preprocessing step taking approximately 14 hours, we randomly selected 1000 groups for each different group size and reported the results only for those groups in Table 3.10. Column 2, 3, and 4 provide the number of remaining groups among 1000 randomly selected groups after the elimination step, and the mean and median gaps of these groups, respectively. For the best group, i.e., the group having the highest lower bound, the estimated GBC values with gaps in parenthesis and the exact GBC values are provided in columns 5 and 6, respectively. Last two columns store the total time to calculate bounds on the GBC values and total time to compute the exact GBC values for all 1000 groups, respectively. Based on Table 3.10, we obtain results similar to the already obtained ones; namely, the remaining number of groups are small, gaps are quite small and tend to be right skewed, Algorithm 10 makes good estimations on the GBC values without exact calculations, there is positive correlation between the computational time and the number of groups. Moreover, Algorithm 10 is much more efficient than exact GBC calculations in terms of computational time and time saving of it becomes more significant as the group size increases. As an example, let us make a rough estimate to compute possible time saving that can be obtained by our method. When the group size is 100, the total time to compute the exact GBC values of 1000000 random groups is estimated to be around 54689.6 seconds. On the other hand, instead of computing the exact GBC values, if one employs Algorithm 10, the total time spent to compute the bounds for all of these 1000000 groups would be around 2419.1 seconds. If 1000 groups remain after the elimination step (which is proportional to 1 in 1000), the total time to find the best group would take around $2419.1 + 54.6896 = 2473.7896$ seconds which is much smaller than 54689.6 seconds. This is expected since computing the exact GBC value of a group has an $O(k^3)$ time complexity while the calculation of lower and upper bounds on the GBC value takes $O(k^2)$ time. The average time to compute the exact GBC and the bounds for a single group are demonstrated in Figure 3.7 which shows that as the group size increases the time savings by the bound computations increase. Figure 3.7 also displays the average time to compute only the lower bound for a single group. As the lower bounds estimate the exact GBC values quite well in our computational experiments, one may prefer computing only the lower bounds especially if the number of groups to be explored is quite high.

Table 3.10: The number of remaining groups among randomly selected 1000 groups after elimination by YAT-HW bound pair; mean and median gaps for YAT-HW bound pair; the estimated and exact GBC values for the best group returned by Algorithm 10; and the total time spent for YAT-HW bound calculations and exact GBC calculations on HepTh network for different group sizes k

Group size	Remaining groups	Gap		GBC of group by Algorithm 10		Total time (in s)	
		Mean	Median	Est. *(GAP)*	Exact	Algorithm 10	Exact
10	1	0.0000	0.0000	0.0418 *(0.0001)*	0.0418	0.0563	0.0955
20	1	0.0000	0.0000	0.0653 *(0.0001)*	0.0653	0.1010	0.4829
30	2	0.0001	0.0000	0.0575 *(0.0001)*	0.0575	0.1631	1.4966
40	1	0.0001	0.0001	0.0801 *(0.0005)*	0.0801	0.2675	3.4536
50	1	0.0002	0.0002	0.0944 *(0.0006)*	0.0945	0.4412	6.9800
60	1	0.0003	0.0003	0.0948 *(0.0008)*	0.0949	0.6708	12.0829
70	1	0.0004	0.0004	0.1012 *(0.0011)*	0.1013	0.9958	19.1056
80	1	0.0006	0.0005	0.1288 *(0.0017)*	0.1289	1.3466	28.3024
90	1	0.0008	0.0007	0.1292 *(0.0023)*	0.1295	1.8030	40.0195
100	1	0.0010	0.0009	0.1462 *(0.0028)*	0.1465	2.4191	54.6896

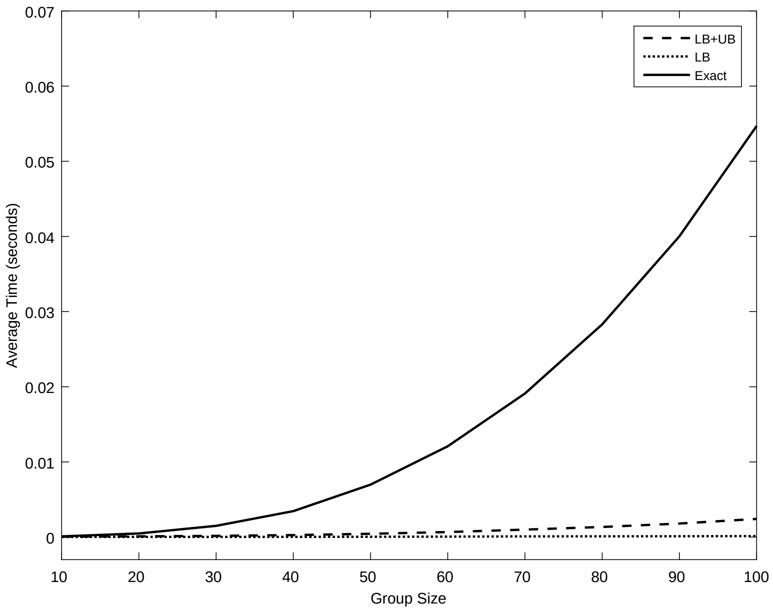


Figure 3.7: Average time to calculate lower bound, lower bound and upper bound, and exact value of GBC of a single group on HepTh network for different group sizes

The last real-life network is High Energy Physics - Phenomenology Collaboration Network which will be referred as the HepPh network hereafter. There are 12008 nodes and 118521 edges in the network. Similar to the HepTh network, the HepPh network is disconnected and it represents the co-authorship of the authors who submitted a paper to High Energy Physics - Phenomenology category between January 1993 and April 2003 (124 months). In the largest connected component of the network, there are 11204 nodes and 117619 edges (after removing the loops).

The preprocessing step of the HepPh network takes approximately 31 hours. Table 3.11 reports the results for randomly selected 1000 groups for different group sizes. Also, the time saving acquired by bounds instead of exact calculations is provided in Figure 3.8. For the HepPh network, the same result are obtained.

Table 3.11: The number of remaining groups among randomly selected 1000 groups after elimination by YAT-HW bound pair; mean and median gaps for YAT-HW bound pair; the estimated and exact GBC values for the best group returned by Algorithm 10; and the total time spent for YAT-HW bound calculations and exact GBC calculations on HepPh network for different group sizes k

Group size	Remaining groups	GAP		GBC of group by Algorithm 10		Total time (in seconds)	
		Mean	Median	Est. *(Gap)*	Exact	Algorithm 10	Exact
10	1	0.0000	0.0000	0.0289 *(0.0000)*	0.0289	0.1399	0.6253
20	1	0.0000	0.0000	0.0437 *(0.0000)*	0.0437	0.1215	0.7911
30	1	0.0000	0.0000	0.0411 *(0.0000)*	0.0411	0.1847	1.5800
40	1	0.0000	0.0000	0.0533 *(0.0001)*	0.0533	0.2994	3.7888
50	2	0.0001	0.0001	0.0582 *(0.0002)*	0.0582	0.4346	6.9452
60	1	0.0001	0.0001	0.0770 *(0.0004)*	0.0770	0.6848	11.7659
70	1	0.0002	0.0002	0.0736 *(0.0006)*	0.0737	0.9976	18.6951
80	1	0.0002	0.0002	0.0896 *(0.0006)*	0.0896	1.3499	27.7952
90	1	0.0003	0.0003	0.0946 *(0.0010)*	0.0946	1.8467	39.4611
100	2	0.0004	0.0004	0.0962 *(0.0008)*	0.0962	2.6795	53.9695

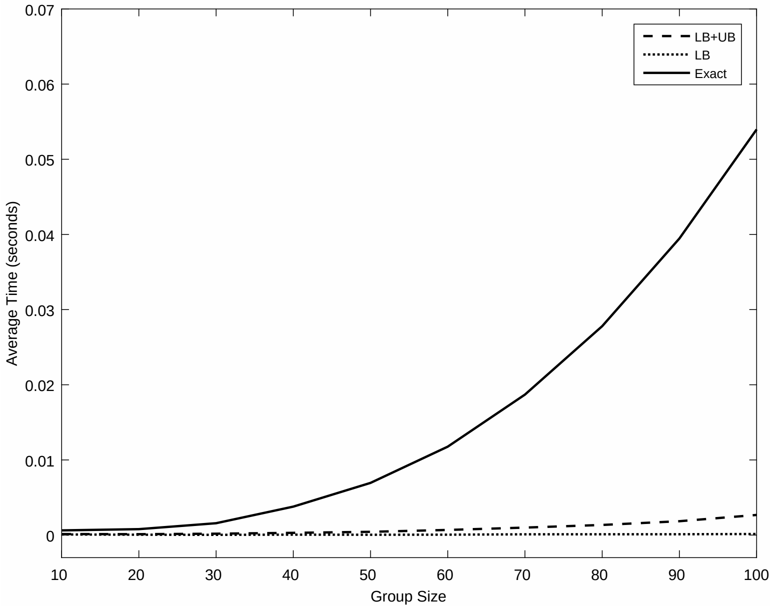


Figure 3.8: Average time to calculate lower bound, lower bound and upper bound, and exact value of GBC of a single group on HepPh network for different group sizes

3.6 Conclusion and Future Work

In this chapter, we propose an algorithm that computes successively bounds on the GBC of several groups of vertices of a given graph. The algorithm combines good properties of Puzis et al. [125] and Kolaczyk et al. [92]. It starts with a preprocessing step as is done in [125] which takes $O(n^3)$ time, where n is the number of vertices in the graph. After the preprocessing step, it finds stronger upper and lower bounds, in comparison with the bounds given in [92], on the GBC value of several groups successively for each one requiring $O(k^2)$ time. We show experimentally that by just using the bounds, only a small fraction of groups remain as candidates for the optimal group. We can then apply the method in [125] to the candidate groups to find the optimal group. Note that as we have experimentally shown, this can be much faster than applying the method in [125] to all the groups given as input. Our bounds are not only stronger than the bounds given in [92], they are also much faster to compute. We achieve this by eliminating the dependency of the co-betweenness values to the group in consideration. We have shown computationally that the gap values returned by the algorithm are small. This shows that the bounds nicely approximate the GBC values. The algorithm also returns an optimality gap which gives the user an upper bound on the difference between the GBC value of the group with the highest lower bound and the optimal GBC value. If the user is satisfied with the optimality gap, s/he may not continue computing the exact GBC values of the candidate groups.

For the networks with a heavy right-tailed distribution of GBCs, there exist groups of vertices controlling a significant portion of the information flow in the network. For such networks the group with the highest GBC can not be obtained from a few random samples. Therefore it is important to be able to compute in an efficient way successively the GBC of several groups.

In large scale real life networks, enumeration of all possible groups and the computation of the bounds for each group may not be practical. In such cases, one may employ various combinatorial optimization methods to find the optimal group of a given size k without enumerating all possible groups. One possible future research direction related to this study may be the following. Assume that one is looking for the optimal group (in terms of the highest GBC value) of size k in a large scale network. Once the preprocessing step is performed in Algorithm 10, one can find bounds on the GBC of a group of any size. For example, after the computation of the bounds on the GBC of a group, say T, of size $\ell < k$, it may be possible to bound the GBC of any group S of size k containing T. If the upper bound is small enough it may be possible to eliminate all groups of size k containing the set T from further consideration. Such branch-and-bound style algorithms may reduce the number of groups to be enumerated in the search for an optimal group.

In the search for the group with the highest group betweenness centrality value, some groups may be immediately discarded and the number of subsets given as input to the algorithm can therefore be reduced. For example, there exists an optimal group that does not contain any leaf vertex (a vertex of degree 1) if k is less than or equal to the number of non-leaf vertices. This is because if a leaf vertex v and the vertex u it is adjacent to are both in a group, then discarding v from the group does not reduce the GBC value of the group. If the vertex v belongs to a group, but not the vertex u, then one can discard v from the group, add u to the group and this operation does not decrease the GBC value of the group. Similarly, if a vertex w and all vertices adjacent to w are in a group, then deletion of w from the group does not change the GBC value of the group. Such preprocessing approaches to reduce the number of subsets that will be given as input to the proposed algorithm are left as future research directions.

Finally, although it is NP-hard to find the group with the highest GBC, it is not currently known whether the problem of finding the group with the highest YAT is NP-hard or not. This is an open question at present and is yet to be investigated.

The publication related with this chapter can be found in [53].

CHAPTER 4

DATA MINING FOR FINDING CENTROID TREES IN THE CLUSTERING OF TREE-STRUCTURED DATA OBJECTS

4.1 Introduction

In the previous chapter, we focused on a single graph to find its central nodes. In this chapter, we have more than one graph and we aim to mine them to discover the underlying information. To be more specific, we address a clustering problem in which each data object is a tree, which is a connected graph which does not include any cycle.

Tree clustering problems appear in many areas. Data objects in these problems may be binary trees or rooted trees. Edges in those trees can be unweighted or weighted. When the edges are unweighted, only topology (i.e., existence/nonexistence of edges) is considered. In the weighted case, trees are clustered based on one or more attributes in addition to the topology. Node correspondence in the trees to be clustered can be known or unknown. When the node correspondence is unknown, before finding the distance or similarity between two trees one should map the nodes of those trees which brings extra computational complexity. Nodes in those trees can be unweighted or weighted. Also, nodes may be labeled or unlabeled. For example, in [5], the branching structure of brain vessels of a patient is represented as a binary tree where both nodes and edges are unweighted and node correspondence is known. By clustering a dataset consisting of such binary trees, we may distinguish the patients with brain tumor from the normal patients. Similarly, in [107], authors represent the branching structure of retinal vessels of a patient as a rooted tree where edges are weighted, nodes are unweighted and node correspondence is known. Then, they aim to see the difference between the retinopathy patients and normal patients by clustering those rooted trees. Another example is available in [90]. Authors represent the mitosis pattern of a blood cell from a mice as a labeled binary tree where node correspondence is unknown and both nodes and edges are unweighted. By using a k-means based algorithm, they try to understand the effects of different biological experimental settings on mitosis patterns. Moreover, the term "tree clustering problem" also appears in the area of XML document clustering since an XML document can be represented as a rooted labeled tree [2].

In this chapter, we consider the clustering of m-ary trees where node correspondence is known, nodes are unweighted, and edges can be both unweighted or weighted. An m-ary tree is a special tree in which each node can have just one edge incident to it (except the root node) and at most m edges incident from it. By using two different measures to find distance/closeness between trees (namely; vertex/edge overlap and graph edit distance), we propose k-means based solution approaches for both unweighted and weighted edge cases.

The chapter is organized as follows: In Section 4.2, we review the related literature. Section 4.3 introduce the notation used throughout the chapter. In Section 4.4, we provide mathematical programming formulations that find the centroid (representative) tree of a given cluster of trees (with unweighted or weighted edges) by using vertex/edge overlap and graph edit distance with node correspondence. In Section 4.5, we give a sketch of algorithm, which is a k-means based algorithm, for the clustering of trees. In Section 4.6, after setting the parameters of tree-kmeans algorithms (for different similarity measures in both unweighted and weighted edge cases), we give the comparison of our algorithms with well known kmodes and kmeans. Moreover, the results of an application with a real-life data is provided in the same section. Finally, in Section 4.7, we conclude the study.

4.2 Literature Review

In [65], author focuses on the statistical aspects (appropriateness: to be able to reveal the underlying structure of the data and steadiness: to be able to obtain more reliable results with the increase in the number of data objects) of tree clustering problems. She considers three types of problem. In the first type, there are m-ary trees as in our study and Hamming distance is used as distance measure. Phylogenetic trees, i.e., binary trees where leaf nodes are labeled, are considered as the second type of problem and modified Hamming distance which uses hypergraphs and Robinson Foulds distance which counts the number of changes (by deleting or adding nodes) needed to obtain one tree from another are discussed as distance measures. Lastly, m-ary trees with weighted nodes are studied and a distance measure which takes the number of different nodes and the difference between the weights' of nodes into consideration is utilized. Then, author propose an extension of k-means algorithm to solve the problems. She states that the centroid tree of a given cluster of trees is the tree which minimizes the sum of distances between trees and the centroid, and it is computable for the first type of problem in such that a node exists in the centroid tree if and only if it appears in at least half of the trees in the cluster. She tests the performance of the proposed algorithm on the first type of problem, for which centroid tree is computable, with two datasets.

[108] and [107] consider the retinal vascular image of a patient which can be represented as a rooted tree. Root node of the tree is the first split-up point of the vessel entering to the retina and then for every split-up point, a new node is defined. Thanks to this rooted tree construction scheme, node correspondence is known. The edge between two nodes represents the vessel between the corresponding split-up points and the length, tortuosity and radius of the corresponding vessel can be used as edge's weights. By clustering of those rooted trees with weighted edges where node correspondence is known, authors aim to see the difference between the retinopathy patients and normal patients. In both of the studies, authors use a new matrix representation for trees. By this way, trees can be considered as points in high dimensional space. They use nonnegative matrix factorization (NMF) for dimensionality reduction. In order to preserve tree structures such as if an edge (vessel) has positive radius then it cannot have zero length, authors modified NMF so as to handle some constraints related with tree structures. After the implementation of structure-constrained NMF, trees are represented as points in the low dimensional space and kmeans algorithm is used for clustering of those points. In [108], inner product and

Euclidean distance are considered as distance measures and inner product resulted in better performance. In [107], NCut algorithm is applied for clustering in addition to kmeans and L1 norm is used as distance measure. NCut algorithm outperforms kmeans. Also, performance comparison of L1 norm with those of L2 norm -Euclidean distance-, quotient Euclidean distance and Torsello's metric reveals that L1 norm is better than others.

In [90], mitosis pattern of a blood cell from a mice is represented as a labeled binary tree. With different biological experimental settings (i.e., different stimulating factors), authors obtain different lineage trees from blood cells. By clustering those trees, they aim to see a difference between lineage trees emerging in different conditions. Although node correspondence is unknown in this study, it is the one of the most relevant studies to our study since it proposes a k-means based algorithm for clustering. In the assignment step of the algorithm, the distance between a tree and a representative tree of a cluster is measured by constrained tree edit distance by [160] or maximal similarity common subtree by [137] which can be utilized with 4 different functions. In the update step of the algorithm, to find the representative tree of a cluster, they start with a random tree and apply a single transformation (relabeling of a node or removing a leaf node or creation of a child for a leaf node) and keep this transformation if it improves the objective function. The trial of transformations are repeated until there is not predetermined number of improvements in a row. They evaluate the performance of the algorithm with different distance measures on both artificial and real datasets.

A related problem is the clustering of general graphs where node correspondence is known. In [136], authors consider a internet network with individual IP addresses as nodes and interactions between them as edges and want to analyze interactions patterns on the network. For the same set of nodes (individuals), they obtain different graphs whose edges are constructed based on different activities such as HTTP, DNS etc. for different time intervals. For instance, a graph may represent interactions between nodes based on the DNS activities for a given time interval and the total number of bytes transferred between nodes may be used as edges' weights. Then, by clustering those different graphs for different activities and different time intervals, they try to define activities/time intervals with similar interaction patterns. This information can be used while designing networks (by selecting hubs, setting capacities etc.). In [95], authors cluster brain connectomes of 114 individuals. To obtain the brain connectomes of the individuals, 70 cortical regions of a brain are defined. Each node on a connectome represents a center of a cortical region and connectivity between cortical regions are represented by edges. By using hierarchical clustering with Ward's method, those 114 connectomes are clustered in two clusters such that highly creative individuals are in one cluster while the remainings are in other cluster. As a distance measure, a new measure called as DELTACON is utilized.

As it is mentioned before, another related problem is XML document clustering problem in which data objects are rooted labeled trees. The solution methods for this problem can be categorized into two as distance based and summary based methods. Distance based methods calculate the distances between documents and uses those distances in order to find clusters. Summary based methods first obtain summaries from the documents and then clusters documents according to the similarities to summaries. The study of Chawathe [37] is an example of distance based solution approaches. Author utilizes the tree edit distance to compute the distance between two XML documents. Tree edit distance [143] is the most commonly used measure to find distance between two trees. It is basically the cost of transforming the first tree into other (or vice versa) by relabeling, deleting, or inserting nodes which have different predetermined costs. But it should be noted that computing the tree edit distance is computationally demanding. The study of Aggarwal et al. [1] is an example of summary based methods. Authors propose a k-means based algorithm whose update step finds frequent substructures (cluster summaries). In the assignment step of the algorithm, the distance between a tree and a cluster is measured as the fraction of covered nodes of the tree by the cluster summary. We eliminate XML document clustering studies since they are not in the scope of this study. In this study, we assume that the node correspondence is known but in an XML document clustering problem the node correspondence is unknown.

Tree clustering problem in which node correspondence is unknown can also be encountered in areas except XML document clustering. For example, in [139] and [102], RNA secondary structures are tried to be clustered after representing them as rooted labeled trees. In both of the studies, tree edit distance is utilized to find the distance between two trees. [139] uses PAM (partition around medoids) clustering while [102] applies hierarchical clustering. Moreover, in [16] and [71], authors use tree clustering concept for web page clustering. In the first one, authors represent JavaScripts of web pages as rooted labeled trees and cluster them by hierarchical clustering to determine malicious web pages. In the second one, authors use shared near neighbor clustering algorithm to cluster web pages based on their similarities after characterizing them as trees. Tree edit distance based measures are used in both studies. Furthermore, 2D figures can be categorized with tree notion. In [138] and [61], 2D figures are represented by their skeletons which are trees and distance between two figures is calculated with tree edit distance. By using the distance matrix representing the pairwise distances between figures, [138] uses pairwise clustering. [61] applies SVM (support vector machine) to classify figures. Besides, tree clustering problem with unknown node correspondence appears in neuroscience. In [75], neuronal cells are represented as 3D rooted labeled trees and distance between them are calculated with a measure based on constrained tree edit distance. Then, hierarchical clustering is utilized to cluster those cells. In a recent study [135], clustering of statute books of local governments, in which statute books are represented as trees and tree edit distance is utilized for similarity evaluations, implies geographical locality.

Although the literature which considers clustering of trees where node correspondence is known is very limited, there is more literature on the components of the problem. Two major components which should be addressed in this study are

- an appropriate measure to find distance/closeness between two trees,
- a method to find centroid (representative) tree for a given cluster of trees.

The problem of finding the distance between two graphs where node correspondence is known is encountered in the problem of anomaly detection in dynamic graphs. Remember the problem considered in Chapter 3. The object to be analyzed is a single (static) graph which is a snapshot of a real-life situation. Instead of a single snapshot, we may take multiple snapshots over time and may have a sequence of graphs where node correspondence is known. Such sequence of graphs is called as dynamic graph. Then, anomaly detection in dynamic graph is the problem of finding the time in which the properties of the graph is highly different than those in other times [127]. To find the anomaly, one needs to compare graphs of adjacent times by looking at the distance/closeness between them. In a recent study [95], authors utilize their distance measure, DELTACON, to find anomalies in a dynamic graph. They compare DELTACON with 6 best state-of-the-art similarity measures; namely vertex/edge overlap, graph edit distance, signature similarity and 3 variations of λ-distance. Vertex/edge overlap is proposed by [122]. It is based on an idea that if two graphs share many vertices and edges, they are similar. It can be defined as the fraction of the number of common nodes and edges to the total number of nodes and edges in two graphs. Graph edit distance is the most commonly used measure to find distance between two graphs whose concept is firstly proposed in 1983 [128]. It is an extension of tree edit distance and is basically the cost of transforming one graph into other by relabeling, deleting, or inserting nodes. Computing the graph edit distance when node correspondence is unknown is also computationally demanding. But when node correspondence is known there is no need for relabeling operation that is the main source of complexity and graph edit distance can be easily computed [49]. Signature similarity is proposed by [122]. It is based on an idea that if two graphs have similar signatures, they are similar. Signature of a graph is the random projection of graph's feature space into smaller dimensional feature space. λ-distance [151] is simply the Euclidean distance between the eigenvalues of matrices representing the graphs. 3 variations of this distance measure come from the different matrix representations of graphs; namely adjacency, laplacian and normalized laplacian matrices.

In this study, we use vertex/edge overlap as the first distance measure to find similarity between two trees because of its relative ease of computation and appropriateness to the problem considered here. Also, it is relatively new measure, proposed in 2010, and not studied to much in the literature. As the second distance measure we use graph edit distance (known node correspondence version) because the same reasons considered for the first measure (ease of computation and appropriateness). Note that where edges are unweighted it is equivalent to Hamming distance considered in [65]. Since graph edit distance is a generalization of tree edit distance and we have trees instead of general graphs in this study, we also search the literature for tree edit distance. In [19], author surveys algorithms to find tree edit distance (for unknown node correspondence) between two trees. He summarizes those algorithms in terms of cases in which they can be used and time and space complexities. To the best of our knowledge, tree edit distance for known node correspondence is not studied in the literature.

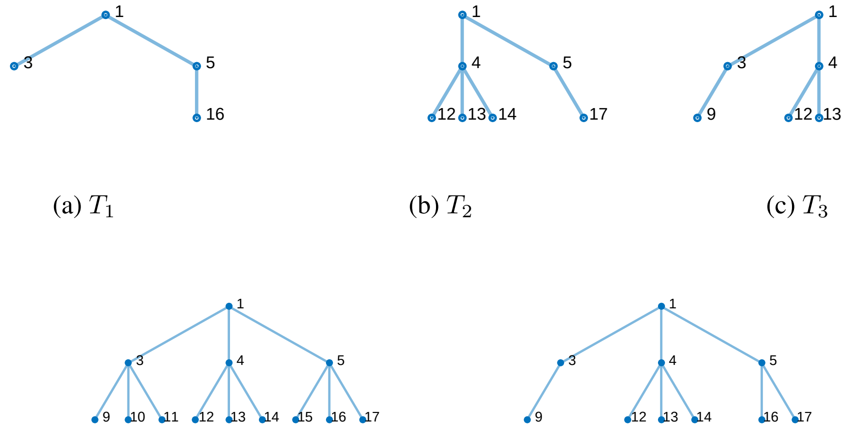
The second component of the clustering problem in this study, i.e., finding centroid tree, mainly studied by statisticians. Their main aim is to convert trees to numerical values so the traditional data mining techniques such as regression can be used to analyze the data. [147], [5] and [6] use the idea of Principal Component Analysis (PCA) which is introduced by Pearson [123]. For point data objects in a space, PCA finds a basis of the space which explains most of the variation in the data. First principal component explains the largest variation, second contains the second largest variation and so on. In [147], driving data is the blood vessels structure in brain which can be represented as binary tree where node correspondence is known as mentioned before. The binary tree for the brain structure of a patient is obtained from Magnetic Resonance Angiography (MRA) brain image. The root node of the tree is the first split-up point of the main vessel entering the brain. Edges represent the vessel parts and the nodes are the split-up points of the vessels. To find principal components for the binary tree space, which are also binary trees, authors consider optimization problems but they obtain results for toy examples. In [5], authors consider the same data and propose a PCA which enables the first real data analysis of binary brain trees. Results show that age and brain vessel structure are related. Two different approaches to PCA for binary trees are also provided in [6]. Although, these approaches are more complex and more expensive (computationally) than the approach in [5], they result in improved data analysis. Note that these studies are not directly related to the problem of finding a centroid tree of a given cluster of trees. But the first principal components found by these studies can be considered as a centroid tree. It should be also noted that PCA for binary tree data is challenging issue because of highly non-Euclidean nature. In [130], authors use Dyck path representation which is a tool for analysis of branching processes of binary trees for brain structures. By this representation, trees are converted to curves. Their approach is a bridge between tree space and standard Euclidean space.

In this study, to find the centroid tree of a given cluster we use mathematical programming formulations. After the formulation of the problem for each selected distance measure, we optimally/heuristically solve the problem. Details will be explained in Section 4.4.

4.3 Notation and Problem Description

A tree is a connected graph that has no cycles. A graph is a tree if and only if there is a unique path joining every pair of vertices. In this paper, the objects of interest are rooted trees with one distinguished vertex called the root. In a rooted tree, the parent of a vertex is the vertex that is visited first on the unique path joining that vertex to the root. Every vertex except the root has a unique parent. If u is the parent of v, then v is said to be a child of u. The level of a vertex is the length of the unique path joining that vertex to the root. Note that the level of the root is zero. The height of a rooted tree is the maximum of the levels of its vertices. When representing the edges of a rooted tree, we use the convention that the first vertex of an edge is the parent of the second vertex. In other words, if u is the parent of v, then we write the edge joining them as (u, v). If w is the parent of u, then (w, u) is called the parent of (u, v), and (u, v) is called a child of (w, u). Two edges having the same parent are called siblings. An edge (w, y) is an ascendant of edge (u, v), if (w, y) lies on the unique path joining (u, v) to the root. (u, v) is said to be a descendant of (w, y) if (w, y) is an ascendant of (u, v). The level of an edge (u, v) is equal to the level of the vertex v. For two edges (u, v) and (w, y) in a rooted tree, we say that (u, v) is at a higher level, if it is closer to the root.

An m-ary tree is a rooted tree in which each vertex has at most m children. A 2-ary tree is also called a binary tree. A complete m-ary tree is an m-ary tree in which each vertex except the vertices in the last level (i.e., vertices whose levels are equal to the height of the tree) has exactly m children. In this study, we have a population $P = \{T_1, T_2, \ldots, T_{nt}\}$ where nt is the population size. Each of these trees is a subtree of a given complete m-ary tree called the generator of the population. We assume that each tree in P has the same root as the generator. We denote by V_i and E_i the vertex and edge sets of $T_i, i \in \{1, 2, \ldots, nt\}$, respectively. Given a complete m-ary tree and its planar drawing with the root at the top, we call the leftmost child of a vertex the first



(a) T_1 (b) T_2 (c) T_3

(d) Generator G of P (e) Support tree ST of P

Figure 4.1: An example of a population $P = \{T_1, T_2, T_3\}$ of 3-ary trees, its generator G, and support tree ST

left child, the second leftmost child as the second left child, and so on. To have the node correspondence between the vertices in different m-ary trees in the population, we label the vertices of the generator recursively from top to bottom starting with the root as follows. The root is labeled with 1. A vertex which is the l^{th} left child of a vertex having label n is labeled with $mn + (l - 1)$. A subtree of a generator does not only take its vertices and edges but also the labels of its vertices from the generator.

Given a population $P = \{T_1, T_2, \ldots, T_{nt}\}$ of trees generated by the complete m-ary tree G, we construct another tree $ST = (SV, SE)$ by taking SV as $\bigcup_{i=1}^{i=nt} V_i$ and SE as $\bigcup_{i=1}^{i=nt} E_i$. The tree ST is called the support tree of the population P and is also a subtree of the generator G. We denote by nv and ne the number of vertices and the number of edges in the support tree, respectively. For an edge $j \in SE$, $ch(j)$, $desc(j)$, and $asc(j)$ represent the set of children, descendants, and ascendants of j in the support tree, respectively. Note that a given population of trees can be generated by different generators. If the generator is unknown, a possible way to construct it is to take the smallest complete m-ary tree containing the support tree of the population. In Figure 4.1, an example of a population, its generator, and support tree are provided.

With the help of the support tree, each tree in the population can be represented by a vector. For this purpose, we order the edges of the support tree level by level from top to bottom, and then from left to right within each level. Let e_{ij} be a binary parameter representing the existence of the j^{th} edge (in the ordered list) of the support tree in T_i. Then, E_i can be represented as a vector $\overrightarrow{E_i} = [e_{ij}]$, $j \in \{1, 2, \ldots, ne\}$. For instance, $\overrightarrow{E_3} = [1\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 0]$ is the vector representation of T_3 in Figure 4.1. This representation, which uses the edges of the support tree instead of those of the generator, is more handy for offline applications where the population is not subject to change.

In this study, the edges of the trees in the population may have some attributes (i.e., the edges of the trees can be weighted). For example, if each tree in the population represents the brain structure of a patient, then the edges may have attributes such as length and radius. Assuming that each edge has na attributes, we denote by w_{ijt} the value of the t^{th} attribute of the j^{th} edge of the support tree for T_i. Here, if an edge is non-existent in a tree, then the value of an attribute for that edge is taken as zero. (In other words, we assume that the values of all attributes, i.e., the weights, are positive for the edges that exist.) In the weighted case, each tree T_i is represented by a weight vector $\overrightarrow{W_i} = [w_{ijt}]$, $j \in \{1, 2, \ldots, ne\}$, $t \in \{1, 2, \ldots, na\}$. For example, the weight vector of T_3 in Figure 4.1 can be $\overrightarrow{W_3} = [0.1\ 0.5\ 0\ 0.4\ 0.8\ 0.2\ 0\ 0\ 0]$ when there is a single attribute and it can be $\overrightarrow{W_3} = [0.1\ 0.5\ 0\ 0.4\ 0.8\ 0.2\ 0\ 0\ 0\ |\ 0.9\ 0.8\ 0\ 0.9\ 0.7\ 0.5\ 0\ 0\ 0]$ when there are two attributes.

Given two (unweighted or weighted) trees T_1 and T_2, $d(T_1, T_2)$ and $s(T_1, T_2)$ represent the distance and the similarity between them, respectively. In this study, we aim to partition a given population $P = \{T_1, T_2, \ldots, T_{nt}\}$ of trees into a given number k of clusters $C_1, C_2, \ldots, C_k$ and find a representative (centroid) tree CT_c for each cluster $c \in \{1, 2, \ldots, k\}$ as

$$\text{minimize} \sum_{c \in \{1,2,\ldots,k\}} \sum_{i: T_i \in C_c} d(T_i, CT_c), \tag{4.1}$$

or

$$\text{maximize} \sum_{c \in \{1,2,\ldots,k\}} \sum_{i: T_i \in C_c} s(T_i, CT_c). \tag{4.2}$$

In the next section, we define the similarity and distance measures used in this study and discuss how the centroid tree of a given subpopulation of trees can be computed.

4.4 Finding Representative Tree of a Given Population of Trees

For a given subpopulation (cluster) $C \subseteq P = \{T_1, T_2, \ldots, T_{nt}\}$ of trees, finding a tree that represents all the trees in C well is an important problem that appears as a subproblem in centroid-based clustering algorithms.

The centroid tree, CT, of C is defined to be the tree that minimizes (maximizes) the sum of the distances (similarities) between the trees in the cluster C and the centroid tree CT. Mathematically, we have

$$CT = \operatorname*{argmin}_{CT \in \Gamma} \sum_{i:T_i \in C} d(T_i, CT) \qquad or \qquad CT = \operatorname*{argmax}_{CT \in \Gamma} \sum_{i:T_i \in C} s(T_i, CT), \tag{4.3}$$

where Γ is the set of all subtrees of the support tree.

Let $CT = (CV, CE)$ be a centroid tree with vertex set CV and edge set CE. Assume that ce_j is a binary variable representing the existence of the j^{th} edge of the support tree in CT. Then, $\overrightarrow{CE} = [ce_j]$, $j \in \{1, 2, \ldots, ne\}$, is the vector representation of CE. Let cw_{jt} be the value of the t^{th} attribute of the j^{th} edge of the support tree in CT. Similarly, we have $\overrightarrow{CW} = [cw_{jt}]$, $j \in \{1, 2, \ldots, ne\}$, $t \in \{1, 2, \ldots, na\}$ as the weight vector of CT.

The distance or similarity measure used is crucial in the centroid finding problem defined above. Next, we will consider the measures used in this study for the unweighted and weighted trees separately. Note that when the trees are unweighted, the clustering is done based only on the topologies of the trees in the population.

4.4.1 Unweighted Vertex/Edge Overlap (UWVEO)

In [122], the authors define a similarity measure named as the vertex/edge overlap (VEO) based on the idea that if two graphs share many vertices and edges, then they are similar. The VEO of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, denoted by $VEO(G_1, G_2)$, is defined as

$$VEO(G_1, G_2) = \frac{|V_1 \cap V_2| + |E_1 \cap E_2|}{|V_1| + |V_2| + |E_1| + |E_2|}. \tag{4.4}$$

Using the fact that the number of vertices in a tree is equal to the number of edges in the tree plus one, the VEO of two trees $T_A = (V_A, E_A)$ and $T_B = (V_B, E_B)$ generated by a generator G can be rewritten as

$$VEO(T_A, T_B) = \frac{2\,|E_A \cap E_B| + 1}{2\,|E_A| + 2\,|E_B| + 2}. \tag{4.5}$$

Note that, in obtaining the numerator of Equation 4.5, we also use the fact that the intersection of T_A and T_B is a tree.

Using the VEO as the distance measure, the centroid finding problem can be formulated as a nonlinear integer program (NIP) as follows.

$$\begin{aligned} \text{maximize} \quad & f_{VEO_{uw}} = \textstyle\sum_{i:T_i \in C} \frac{2\sum_{j=1}^{ne} e_{ij} ce_j \; + 1}{2\sum_{j=1}^{ne} e_{ij} \; + 2\sum_{j=1}^{ne} ce_j \; + 2} \\ \text{subject to} \quad & ce_j \leq ce_{j'},\, j' \in \{1, 2, \ldots, ne\}, j \in ch(j'), \\ & ce_j \in \{0, 1\}, j \in \{1, 2, \ldots, ne\}. \end{aligned} \tag{NIP-UWVEO}$$

The objective function maximizes the sum of the similarities between the centroid tree and the trees in the cluster. The first set of constraints preserves the parent-child relations of the edges. An edge j which is a child of the edge j' may appear in CT only if j' is in CT. The next set of constraints are the binary restrictions on the variables. For the sake of simplicity, let us represent the solution space (feasible region) of the above formulation as $\mathbb{CE}$ and the solution space of the relaxed problem, i.e., the problem in which the second set of constraints is replaced with $0 \leq ce_j \leq 1$, $j \in \{1, 2, \ldots, ne\}$, as $\mathbb{CE}_\mathrm{R}$. We first show that when the objective function of (NIP-UWVEO) is replaced with a linear objective function, then the resulting optimization problem can be solved as a linear program (LP) after relaxing the binary restrictions.

Proposition 1. $\mathbb{CE}_\mathrm{R}$ *is an integral polytope, i.e., it has integral extreme points.*

Proof. Assume that $\overrightarrow{CE^*}$ is an extreme point of $\mathbb{CE}_\mathrm{R}$ that is fractional, i.e., it has at least one fractional component. Consider the set $\mathcal{S}$ of all edges j with $0 < ce^*_j < 1$. As $\overrightarrow{CE^*}$ is fractional, $\mathcal{S}$ is non-empty. There exists an ϵ such that $ce^*_i + \epsilon \leq 1$ and $ce^*_i - \epsilon \geq 0$ for all $i \in \mathcal{S}$. Consider the two solutions $\overrightarrow{CE^*_1}$ and $\overrightarrow{CE^*_2}$ in $\mathbb{CE}_\mathrm{R}$ that are obtained from $\overrightarrow{CE^*}$ as follows. $\overrightarrow{CE^*_1}$ is obtained from $\overrightarrow{CE^*}$ by adding ϵ to the fractional components of $\overrightarrow{CE^*}$, and $\overrightarrow{CE^*_2}$ is obtained from $\overrightarrow{CE^*}$ by subtracting ϵ from the fractional components of $\overrightarrow{CE^*}$. Clearly, the solutions $\overrightarrow{CE^*_1}$ and $\overrightarrow{CE^*_2}$ satisfy the parent child restrictions in (NIP-UWVEO). We have that $\overrightarrow{CE^*} = 0.5\overrightarrow{CE^*_1} + 0.5\overrightarrow{CE^*_2}$

showing that $\overrightarrow{CE^*}$ is not an extreme point of $\mathbb{CE}_\mathrm{R}$. This shows by contradiction that $\mathbb{CE}_\mathrm{R}$ is integral. $\square$

Since there are decision variables in both the numerator and the denominator in (NIP-UWVEO), the objective function is nonlinear. Note, however, that when we fix the centroid tree size, i.e., the number of edges in the centroid tree, the objective function becomes linear. Consider now the problem of finding a centroid tree of a given size cs. In this case, we can replace $\sum_{j=1}^{ne} ce_j$ in the denominator of the objective function of (NIP-UWVEO) with cs and linearize the objective function. After adding the constraint, $\sum_{j=1}^{ne} ce_j = cs$, the formulation becomes the following integer linear program (ILP).

$$\begin{array}{lll}\text{maximize} & f_{VEO_{uw}}(cs) = \sum_{i:T_i \in C} \dfrac{2\sum_{j=1}^{ne} e_{ij}ce_j \ + 1}{2\sum_{j=1}^{ne} e_{ij} \ + 2cs \ + 2} & \\ \text{subject to} & ce_j \leq ce_{j'}, j' \in \{1, 2, \ldots, ne\}, j \in ch(j'), & \\ & \sum_{j=1}^{ne} ce_j = cs, & \text{(ILP-UWVEO)} \\ & ce_j \in \{0, 1\}, j \in \{1, 2, \ldots, ne\}. & \end{array}$$

Proposition 2. *The linear programming relaxation of (ILP-UWVEO) has at least one integral optimal solution. Moreover, from a fractional optimal solution of the linear programming relaxation of (ILP-UWVEO), an alternative integral optimal solution can be easily obtained.*

Proof. Assume that $\overrightarrow{CE^*}$ is a fractional optimal solution of the linear programming relaxation of (ILP-UWVEO). Consider the set $\mathcal{S}$ of all edges j with $0 < ce_j^* < 1$. Note that $\mathcal{S}$ cannot be a singleton. Let $coef_j$ represent the coefficient of the variable ce_j in the objective function for the edge j. For two edges $j, \bar{j} \in \mathcal{S}$ with $j \in desc(\bar{j})$, it is clear that $coef_j \leq coef_{\bar{j}}$ as $e_{ij} \leq e_{i\bar{j}}$. On the other hand, if $coef_j < coef_{\bar{j}}$, we can increase the objective function value by decreasing ce_ℓ^* by some small positive value ϵ, where $\ell \in desc(j)$ is an edge at the lowest level with fractional ce_ℓ^* and by increasing ce_ν^* by ϵ, where $\nu \in asc(\bar{j})$ is an edge at the highest level with fractional ce_ν^*. Thus, we have shown that for two edges $j, \bar{j} \in \mathcal{S}$ with $j \in desc(\bar{j})$, it holds that $coef_j = coef_{\bar{j}}$. If $j, \bar{j} \in \mathcal{S}$ are two edges with no ascendant or descendant relationship, then it also holds true that $coef_j = coef_{\bar{j}}$. To prove this, assume $coef_j < coef_{\bar{j}}$. Let $\ell \in desc(j)$ be the edge at the lowest level with fractional ce_ℓ^* and $\nu \in asc(\bar{j})$ be the edge at the highest level with fractional ce^*_ν. The objective function value can be increased by decreasing ce^*_ℓ by some small positive value ϵ and increasing ce^*_ν by ϵ resulting in a contradiction. We have thus shown that in an optimal solution the coefficients of the fractional variables in the objective function are all the same, i.e., $coef_j = coef_{\bar{j}}$ for all $j, \bar{j} \in \mathcal{S}$.

If $\mathcal{S} = \emptyset$, the optimal solution is integral. Otherwise, an alternative optimal solution that is integral can be obtained from $\overrightarrow{CE^*}$ as follows. It is clear that $\sum_{j \in S} ce^*_j$ is an integer, say s. Let $\bar{\mathcal{S}}$ be a subset of $\mathcal{S}$ such that $|\bar{\mathcal{S}}| = s$ and no edge in $\bar{\mathcal{S}}$ is a descendant of an edge in $S \setminus \bar{\mathcal{S}}$. We set ce^*_j to 1 for all edges in $\bar{\mathcal{S}}$ and to 0 for all edges in $\mathcal{S} \setminus \bar{\mathcal{S}}$ to obtain an alternative optimal solution that is integral. □

To obtain an optimal solution of (NIP-UWVEO), we can then solve the LP relaxation of (ILP-UWVEO) for all possible integer cs values (and obtain an integral alternative optimal solution when it is necessary) and select the solution with the highest objective function value.

Note that for different cs values, resulting ILPs are knapsack problems with some precedence relations. Because the coefficients of the variables in the objective function are non-increasing as we move down the tree, one can optimally solve the LP relaxation of (ILP-UWVEO) by the following greedy method. Add the edges (chosen among the unadded edges starting with the highest coefficient) one by one to the centroid tree until cs many edges are added. If among the unadded edges, there are multiple edges with the highest coefficient, add the one that is closest to the root node (breaking ties arbitrarily).

4.4.2 Unweighted Graph Edit Distance (UWGED)

A commonly used measure of dissimilarity between two graphs is the graph edit distance (GED) proposed in [128]. GED between two graphs is the minimum cost of transforming one of the graphs into the other by relabeling, deleting, or inserting nodes/edges where each operation has a predetermined cost. The computation of the GED is in general a difficult problem. However, when the node correspondence is known, there is no need for the relabeling operation which is the main source of complexity. In this case, the GED can be easily computed [49]. If only the topological changes are considered, i.e., the cost of deleting or inserting a node or an edge is 1, the GED between $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with a known node correspondence is calculated as follows.

$$GED(G_1, G_2) = |V_1| + |V_2| - 2\,|V_1 \cap V_2| + |E_1| + |E_2| - 2\,|E_1 \cap E_2| \tag{4.6}$$

Given two trees $T_A = (V_A, E_A)$ and $T_B = (V_B, E_B)$ generated using the generator G, the graph edit distance $GED(T_A, T_B)$ between T_A and T_B can be computed as follows where we use the facts that $T_A \cap T_B$ is a tree and the number of vertices in a tree is equal to the number of edges in the tree plus one.

$$GED(T_A, T_B) = 2\,|E_A| + 2\,|E_B| - 4\,|E_A \cap E_B| \tag{4.7}$$

The ILP formulation of the centroid finding problem with the unweighted GED as the distance measure is given below.

$$\begin{aligned} &\text{minimize} \quad f_{GED_{uw}} = \textstyle\sum_{i=1}^{nt} \left(\sum_{j=1}^{ne} e_{ij} + \sum_{j=1}^{ne} ce_j - 2 \sum_{j=1}^{ne} e_{ij} ce_j \right) \\ &\text{subject to} \quad ce \in \mathbb{CE}. \end{aligned} \tag{ILP-UWGED}$$

The constraints in (ILP-UWGED) are the same as those in (NIP-UWVEO). The objective function minimizes the sum of graph edit distances between the centroid tree and the trees in the dataset and is linear. As proved in Proposition 1, (ILP-UWGED) can be solved to optimality by solving its LP relaxation.

Note that the objective function of the above formulation can be rewritten as follows.

$$\begin{aligned} f_{GED_{uw}} = \sum_{i=1}^{nt} \sum_{j=1}^{ne} (e_{ij} + ce_j - 2e_{ij}ce_j) &= \sum_{i=1}^{nt} \sum_{j=1}^{ne} (e_{ij}(1 - ce_j) + ce_j(1 - e_{ij})) \\ &= \sum_{j=1}^{ne} \sum_{i=1}^{nt} |e_{ij} - ce_j| \, . \end{aligned} \tag{4.8}$$

It can be seen that for each $j \in \{1, 2, \ldots, ne\}$, the sum $\sum_{i=1}^{nt} |e_{ij} - ce_j|$ is minimized when ce_j is equal to the median of the $e_{ij}, i \in \{1, 2, \ldots, nt\}$. Note that if the median of the e_{ij}'s is 0.5, then both values $ce_j = 0$ or 1 also minimize $\sum_{i=1}^{nt} |e_{ij} - ce_j|$. If these solutions can be combined so that the parent-child relations hold true, i.e., the contraints $ce \in \mathbb{CE}$ are satisfied, then an optimal solution of (ILP-UWGED) is obtained.

Proposition 3. *An optimal solution of (ILP-UWGED) can be obtained by taking ce_j^* as 1 when $\sum_{i=1}^{nt} e_{ij^*} > \frac{nt}{2}$ and as 0 otherwise.*

Proof. Note that when $\sum_{i=1}^{nt} e_{ij^*} > \frac{nt}{2}$, the median of the e_{ij^*}'s is 1. In this case, taking ce_j^* as 1 minimizes $\sum_{i=1}^{nt} \left|e_{ij^*} - ce_j^*\right|$. On the other hand, when $\sum_{i=1}^{nt} e_{ij^*} \leq \frac{nt}{2}$, $e_{ij^*} = 0$ is a minimizer of $\sum_{i=1}^{nt} \left|e_{ij^*} - ce_j^*\right|$. As for two edges j, j' with $j \in ch(j')$, we have $\sum_{i=1}^{nt} e_{ij'} \geq \sum_{i=1}^{nt} e_{ij}$, this solution satisfies the parent-child constraints and therefore an optimal solution of (ILP-UWGED). □

Note that another optimal solution of (ILP-UWGED) can be obtained by taking ce_j^* as 1 when $\sum_{i=1}^{nt} e_{ij^*} \geq \frac{nt}{2}$ and as 0 otherwise. Although the objective function value of (ILP-UWGED) remains the same, when we have multiple clusters, having different centroid tree may affect the final clustering result. Thus, we consider this issue in parameter selection.

4.4.3 Weighted Vertex/edge Overlap (WVEO)

In this study, we have weights on the edges but not on the vertices of the trees. We define the weighted VEO between two trees with weights on the edges as the ratio of the total edge weight overlap to the total edge weights when a single attribute exists. In case of multiple attributes, these ratios are added up. Given two trees $T_A = (V_A, E_A)$ and $T_B = (V_B, E_B)$, let w_{Ajt} and w_{Bjt} represent the weights of edge $j \in E_A \cup E_B$ for the attribute $t \in \{1, 2, \ldots, na\}$ in T_A and T_B, respectively. Then, mathematically, the weighted VEO between T_A and T_B is given by

$$VEO_w(T_A, T_B) = \sum_{t=1}^{na} \frac{\sum_{j \in E_A \cup E_B} \min(w_{Ajt}, w_{Bjt})}{\sum_{j \in E_A} w_{Ajt} + \sum_{j \in E_B} w_{Bjt}}. \tag{4.9}$$

Note that $E_A \cup E_B$ in the numerator and E_A and E_B in the denominator in Equation 4.9 can all be replaced by the set of edges in the support tree because for non-existent edges we assign a weight value of zero. Letting y_{ijt}, $i \in \{1, 2, \ldots, nt\}$,

$j \in \{1, 2, \ldots, ne\}$, $t \in \{1, 2, \ldots, na\}$ represent $\min(w_{ijt}, cw_{jt})$, a nonlinear programming (NLP) formulation of the centroid finding problem when VEO$_w$ is used as the distance measure is as follows.

$$\begin{aligned} \text{maximize} \quad & f_{VEO_w} = \textstyle\sum_{i=1}^{nt} \sum_{t=1}^{na} \frac{\sum_{j=1}^{ne} y_{ijt}}{\sum_{j=1}^{ne} w_{ijt} + \sum_{j=1}^{ne} cw_{jt}} \qquad \text{(NLP-WVEO)} \\ \text{subject to} \quad & y_{ijt} \le w_{ijt}, i \in \{1, \ldots, nt\}, j \in \{1, \ldots, ne\}, t \in \{1, \ldots, na\}, \\ & y_{ijt} \le cw_{jt}, i \in \{1, \ldots, nt\}, j \in \{1, \ldots, ne\}, t \in \{1, \ldots, na\}. \end{aligned}$$

This formulation can be decomposed. For each attribute, the following NLP formulation can be solved.

$$\begin{aligned} \text{maximize} \quad & f^t_{VEO_w} = \textstyle\sum_{i=1}^{nt} \frac{\sum_{j=1}^{ne} y_{ijt}}{\sum_{j=1}^{ne} w_{ijt} + \sum_{j=1}^{ne} cw_{jt}} \\ \text{subject to} \quad & y_{ijt} \le w_{ijt}, i \in \{1, 2, \ldots, nt\}, j \in \{1, 2, \ldots, ne\}, \qquad \text{(NLP-WVEO}^t\text{)} \\ & y_{ijt} \le cw_{jt}, i \in \{1, 2, \ldots, nt\}, j \in \{1, 2, \ldots, ne\}. \end{aligned}$$

To solve (NLP-WVEO^t), we propose a heuristic, namely an iterative multi dimensional descent search algorithm. The algorithm starts with some initial weights for cw_{jt}'s, $j \in \{1, \ldots, ne\}$, and tries to improve the objective function value by changing the weights. The edges are ordered at the beginning of each algorithmic cycle consisting of ne iterations. In each iteration, it finds the best value of cw_{jt} for a particular edge $j \in \{1, \ldots, ne\}$ among the given weights $0, w_{1jt}, \ldots, w_{nt,jt}$ while keeping the weights of other edges constant. These iterations are repeated for the edges one by one considering the ordering of the edges in the current algorithmic cycle. Then, a new cycle begins. Cycles continue until it is guaranteed that the current solution cannot be improved any further. In this algorithm, the starting weights and the ordering of edges within a cycle may affect the final centroid tree. Thus, we tried different alternatives for these parameters.

4.4.4 Weighted Graph Edit Distance (WGED)

In [49], the authors used the graph edit distance to measure the distance between graphs having edge weights. We define the weighted GED between two trees $T_A = (V_A, E_A)$ and $T_B = (V_B, E_B)$, denoted by $GED_w(T_A, T_B)$, as follows.

$$GED_w(T_A, T_B) = \sum_{t=1}^{na} \sum_{j \in E_A \setminus E_B} w_{Ajt} + \sum_{j \in E_B \setminus E_A} w_{Bjt} + \sum_{j \in E_A \cap E_B} |w_{Ajt} - w_{Bjt}|, \tag{4.10}$$

where w_{Ajt} and w_{Bjt} are the weights of edge $j \in E_A \cup E_B$ for the attribute $t \in \{1, 2, \ldots, na\}$ in T_A and T_B, respectively. Note that the definition used in [49] includes a scaling parameter which is not used here.

By using the weighted GED as the distance measure, the centroid finding problem can be formulated as a mixed integer nonlinear program (MINLP) as follows.

$$\begin{array}{ll} \text{minimize} & f_{GED_w} = \sum_{i=1}^{nt} \left(\sum_{t=1}^{na} \left(\begin{array}{r} \sum_{j=1}^{ne} e_{ij}(1 - ce_j)w_{ijt} + \sum_{j=1}^{ne}(1 - e_{ij})ce_j cw_{jt} \\ + \sum_{j=1}^{ne} e_{ij}ce_j \left| w_{ijt} - cw_{jt} \right| \end{array} \right) \right) \\ \text{subject to} & ce \in \mathbb{CE}. \end{array} \qquad \text{(MINLP-WGED)}$$

The objective function in (MINLP-WGED) minimizes the sum of weighted graph edit distances between the centroid tree and the trees in the dataset. It is nonlinear due to the existence of the absolute values and the decision variables multiplied together.

Proposition 4. *The solution in which cw^*_{jt} is the median of $w_{1jt}, \ w_{2jt}, \ldots, \ w_{nt,jt}$ when nt is odd and is the smallest (largest) of the middle two observations when nt is even for each $j \in \{1, 2, \ldots, ne\}$ and $t \in \{1, 2, \ldots, na\}$ and ce^*_j is 1 if and only if any cw^*_{jt}, $t \in \{1, 2, \ldots, na\}$ is greater than 0 for each $j \in \{1, 2, \ldots, ne\}$, is optimal.*

Proof. Using our convention that the weight of a non–existent edge is 0, the objective function of (MINLP-WGED) can be rewritten as follows

$$f_{GED_w} = \sum_{i=1}^{nt} \sum_{t=1}^{na} \sum_{j=1}^{ne} \left| w_{ijt} - cw_{jt} \right| . \tag{4.11}$$

To see this, we consider four cases fixing i, j, and t and show in each case that

$$e_{ij}(1 - ce_j)w_{ijt} + (1 - e_{ij})ce_j cw_{jt} + e_{ij}ce_j \left| w_{ijt} - cw_{jt} \right| \tag{4.12}$$

is equal to

$$\left| w_{ijt} - cw_{jt} \right| . \tag{4.13}$$

In case 1, $e_{ij} = ce_j = 0$. In this case, $w_{ijt} = cw_{jt} = 0$ and therefore (4.12) and (4.13) are both 0. In case 2, $e_{ij} = ce_j = 1$. In this case, (4.12) reduces to (4.13). In case 3, $e_{ij} = 0$ and $ce_j = 1$. In this case, $w_{ijt} = 0$ and both (4.12) and (4.13) are equal to cw_{jt}. In case 4, $e_{ij} = 1$ and $ce_j = 0$. In this case, $cw_{jt} = 0$ and both (4.12) and (4.13) are equal to w_{ijt}.

The formulation (MINLP-WGED) with the objective in (4.11) can be decomposed. For each cw_{jt}, $j \in \{1, \ldots, ne\}$, $t \in \{1, \ldots, na\}$, $\sum_{i=1}^{nt} |w_{ijt} - cw_{jt}|$ is to be minimized. The solution, cw^*_{jt}, minimizing this objective, i.e., the sum of absolute differences, is the median value of $w_{1jt}, \ldots, w_{nt,jt}$ when nt is odd and is any value between the middle two observations when nt is even. Choosing each cw^*_{jt} as the smallest (or the largest) of the middle two observations and ce^*_j as 1 if and only if any cw^*_{jt}, $t \in \{1, \ldots, na\}$ is greater than 0 for each $j \in \{1, \ldots, ne\}$ make sure that the parent-child relationships are satisfied. □

In the presence of alternative optimal solutions, i.e., when there are an even number of trees in the centroid finding problem, selecting one of them does not affect the objective function value for the cluster in consideration, but this selection may eventually affect the overall objective function value of the final clustering. Different alternative optimal centroid trees may result in convergence to different partitioning results. Thus, we consider this issue in parameter selection.

4.5 Clustering of Tree-Structured Data

K-means algorithm [111], which is the most popular clustering algorithm because of its ease of implementation, simplicity, efficiency and empirical success, can be utilized to solve the tree clustering problem. After starting with initial centroid trees, the algorithm repeats the assignment and update steps until convergence. General framework of tree-kmeans algorithms is given in Algorithm 11.

Algorithm 11 tree-kmeans

```
Initialization: Obtain initial centroid trees.
repeat
    Assignment: Assign each tree $T_i \in P$ to the most similar cluster.
    Update: Update each centroid tree by considering trees assigned to it.
until Assignments do not change.
return Partition of $P$.
```

Selection of initial centroid trees, i.e., selection of edges from the support tree to be included in the initial centroid trees and setting their weights for the weighted cases, is an important issue in k-means, since the algorithm may converge different partitionings with different starting points. To be able to preserve topological structures in the dataset, in each replication we use frequency based random selection to decide edges of the initial centroid trees. This selection procedure can be explained with an example. Consider a small dataset with 4 3-ary trees given in Figure 4.2 whose support tree with edge labels representing the conditional frequencies of the edges is given in Figure 4.3. For example, edge (1,3) appears in half of the trees and edge (3,11) appears in all of the trees in which its parent edge appears. Starting from the edges in the first level of the support tree, we generate a random number and if it is smaller than the corresponding conditional relative frequency, we include the edge in the initial centroid tree. Then, we continue with the edges in the next level whose parent edges are already included in the centroid tree. For the weighted case, we tried two different alternatives for the weight of an edge in an initial centroid tree; namely, mean and median of nonzero weights for the corresponding edge in the dataset. Details of the alternative selection procedure are provided in Section 4.6.1.

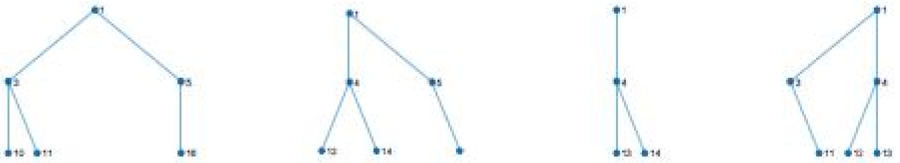


Figure 4.2: Small dataset with 4 3-ary trees

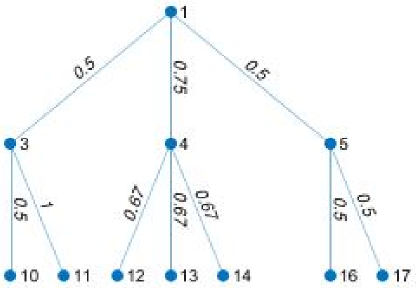


Figure 4.3: Conditional frequencies of edges appearing in the dataset in Figure 4.2

In the assignment steps, each tree in the data is assigned to the closest centroid tree (which is the tree minimizing the GED or maximizing the VEO between the tree and the centroid tree). In the update steps, the centroid trees are updated by considering the trees assigned to them. The details of finding the centroid tree of a given cluster are discussed in Section 4.4.

4.6 Computational Studies

4.6.1 Parameter Selection

In order to test the performances of the algorithms under different parametric settings and decide on which setting is the best, we generated 2 unweighted and 2 weighted random datasets. For the data generation, we use a procedure which is based on the random data generation scheme in [90]. The original procedure in [90] has 3 inputs to generate labeled binary trees, namely a matrix with transition probabilities ($pvals$), vector with probabilities to continue ($ContProbability$) and discontinuity factor (γ). For example, assume that there can be two different node labels, a and b, in a tree and the parameters are as follows.

$$pvals = \begin{bmatrix} 0.6 & 0.4 \\ 0.7 & 0.3 \end{bmatrix}, \quad ContProbability = \begin{bmatrix} 0.9 \\ 0.8 \end{bmatrix}, \quad \gamma = 0.7$$

Starting with a labeled node as a root node, this node will have at least one children, i.e., the generation procedure will continue with probability 0.9 if its label is a, or with probability 0.8 if its label is b. If the random variable generated for the continue decision is smaller than the corresponding $ContProbability$, next step is to decide on the labels of the children. The labels are sampled using multinomial distribution with parameters 0.6 and 0.4 if parent node's label is a, or with parameters 0.7 and 0.3 if parent node's label is b. This means that a labeled node will get a labeled child with probability 0.6 and b labeled child with probability 0.4, while b labeled node will get a labeled child with probability 0.7 and b labeled child with probability 0.3. For each newly added leaf node the procedure is repeated. While going down to the levels of generated tree, $ContProbability$ is reduced by multiplying it with γ.

Since the trees in the problem considered in this study are not labeled, we do not require a transition probabilities matrix. To generate an m-ary tree, a vector (p) with probabilities of generating 1^{st}, 2^{nd},..., m^{th} left child of a node and a discontinuity factor (γ) are sufficient. For example, assume that to generate a 3-ary tree the parameters are as follows. A node in the second level of a tree will have left-most child with probability $0.9 * 0.7 * 0.7$, middle child with probability $0.2 * 0.7 * 0.7$ and right-most child with probability $0.1 * 0.7 * 0.7$.

$$p = \begin{bmatrix} 0.9 \\ 0.2 \\ 0.1 \end{bmatrix}, \quad \gamma = 0.7$$

The details of random datasets are given in Tables 4.1, 4.2, 4.3 and 4.4; namely SmallDataUW, LargeDataUW, SmallDataW and LargeDataW. In each data set we have 50 problem instances. Each problem instance in SmallDataUW and LargeDataUW consists of two equally sized groups (clusters) of trees generated with different parameters. We try to differentiate clusters according to tree topologies in such a way that trees in the first cluster are left aligned while trees in the second cluster are right aligned. The cluster sizes in SmallDataUW is 20 while it is 30 in LargeDataUW. On the other hand, there are four equally sized clusters in the problem instances of SmallDataW and LargeDataW. We attempt to diversify clusters based on the edge weights in addition to tree topologies such that the first cluster consists of left aligned trees with edge weights that are randomly generated between 0 and 0.5, the second cluster is formed with left aligned trees with edge weights that are random between 0.5 and 1, the third cluster comprises of right aligned trees with random edge weights between 0 and 0.5, and there are right aligned trees with random edge weights between 0.5 and 1 in the last cluster. The cluster sizes are 10 and 15 for SmallDataW and LargeDataW, respectively. In those tables, we also report the properties of the problem instances, namely the average and maximum number of levels in individual trees, average and maximum number of edges in individual trees, average and maximum number of edges in the support trees of problem instances. For example, in the first five problem instances of SmallDataUW provided in Table 4.1, in the generation procedure, discounting factor is taken as 0.6, 20 3-ary trees are generated with p_1 and 20 3-ary trees are generated with p_2. In total, there are 40 3-ary trees in each instance.

Table 4.1: The properties of randomly generated problem instances in SmallDataUW

Instance	$\gamma^{(1)}$	$p_1^{(2)}$	$p_2^{(2)}$	# of Levels$^{(3)}$		# of Edges$^{(4)}$		$ne^{(5)}$	
				Avg.	Max.	Avg.	Max.	Avg.	Max.
1-5	0.6	(0.9,0.1,0.1)	(0.1,0.1,0.9)	2.10	6	2.53	9	22.6	29
6-10	0.6	(0.8,0.2,0.2)	(0.2,0.2,0.8)	2.11	5	3.02	14	27.6	34
11-15	0.6	(0.7,0.3,0.3)	(0.3,0.3,0.7)	2.24	5	3.37	14	34.0	37
16-20	0.6	(0.6,0.4,0.4)	(0.4,0.4,0.6)	2.28	6	3.82	16	34.0	47
21-25	0.6	(0.5,0.5,0.5)	(0.5,0.5,0.5)	2.54	5	4.18	15	39.8	45
26-30	0.8	(0.9,0.1,0.1)	(0.1,0.1,0.9)	3.17	8	4.07	17	47.0	62
31-35	0.8	(0.8,0.2,0.2)	(0.2,0.2,0.8)	3.06	8	4.65	18	58.6	64
36-40	0.8	(0.7,0.3,0.3)	(0.3,0.3,0.7)	3.43	8	5.96	26	88.6	113
41-45	0.8	(0.6,0.4,0.4)	(0.4,0.4,0.6)	3.75	9	7.08	24	111.6	129
46-50	0.8	(0.5,0.5,0.5)	(0.5,0.5,0.5)	4.12	10	8.97	36	142.6	162

(1) Discounting factor

(2) Probabilities of generating (left, middle, right) children of a node in different clusters for the corresponding instances

(3) Average and maximum number of levels of all trees in the corresponding instances

(4) Average and maximum number of edges in all trees in the corresponding instances

(5) Average and maximum number of edges of support trees of the corresponding instances

Table 4.2: The properties of randomly generated problem instances in LargeDataUW

Instance	$\gamma^{(1)}$	$p_1^{(2)}$	$p_2^{(2)}$	# of Levels$^{(3)}$		# of Edges$^{(4)}$		$ne^{(5)}$	
				Avg.	Max.	Avg.	Max.	Avg.	Max.
1-5	0.7	(0.9,0.1,0.1)	(0.1,0.1,0.9)	2.40	6	2.97	12	34.40	41
6-10	0.7	(0.8,0.2,0.2)	(0.2,0.2,0.8)	2.58	7	3.73	18	51.4	55
11-15	0.7	(0.7,0.3,0.3)	(0.3,0.3,0.7)	2.81	7	4.53	19	68.0	81
16-20	0.7	(0.6,0.4,0.4)	(0.4,0.4,0.6)	2.83	7	4.83	17	73.2	82
21-25	0.7	(0.5,0.5,0.5)	(0.5,0.5,0.5)	3.10	7	5.68	24	84.0	92
26-30	0.9	(0.9,0.1,0.1)	(0.1,0.1,0.9)	4.28	13	5.87	23	112.2	133
31-35	0.9	(0.8,0.2,0.2)	(0.2,0.2,0.8)	4.65	14	8.01	39	207.4	246
36-40	0.9	(0.7,0.3,0.3)	(0.3,0.3,0.7)	5.43	14	11.76	52	339.2	390
41-45	0.9	(0.6,0.4,0.4)	(0.4,0.4,0.6)	6.14	15	14.92	65	472.4	575
46-50	0.9	(0.5,0.5,0.5)	(0.5,0.5,0.5)	7.42	16	22.43	93	746.4	857

(1) Discounting factor

(2) Probabilities of generating (left, middle, right) children of a node in different clusters for the corresponding instances

(3) Average and maximum number of levels of all trees in the corresponding instances

(4) Average and maximum number of edges in all trees in the corresponding instances

(5) Average and maximum number of edges of support trees of the corresponding instances

Table 4.3: The properties of randomly generated problem instances in SmallDataW

Instance	$\gamma^{(1)}$	$p_1^{(2)}, p_2^{(2)}$	$p_3^{(2)}, p_4^{(2)}$	$w_1^{(3)}, w_3^{(3)}$	$w_2^{(3)}, w_4^{(3)}$	# of Levels$^{(4)}$		# of Edges$^{(5)}$		$ne^{(6)}$	
						Avg.	Max.	Avg.	Max.	Avg.	Max.
1-5	0.6	(0.9,0.1,0.1)	(0.1,0.1,0.9)	U(0,0.5)	U(0.5,1)	1.91	4	2.26	7	19.4	22
6-10	0.6	(0.8,0.2,0.2)	(0.2,0.2,0.8)	U(0,0.5)	U(0.5,1)	2.07	6	2.84	10	26.8	34
11-15	0.6	(0.7,0.3,0.3)	(0.3,0.3,0.7)	U(0,0.5)	U(0.5,1)	2.29	5	3.54	12	33.4	41
16-20	0.6	(0.6,0.4,0.4)	(0.4,0.4,0.6)	U(0,0.5)	U(0.5,1)	2.42	5	4.09	15	43.4	49
21-25	0.6	(0.5,0.5,0.5)	(0.5,0.5,0.5)	U(0,0.5)	U(0.5,1)	2.51	6	4.36	13	41.2	52
26-30	0.8	(0.9,0.1,0.1)	(0.1,0.1,0.9)	U(0,0.5)	U(0.5,1)	2.84	10	3.48	14	37.6	45
31-35	0.8	(0.8,0.2,0.2)	(0.2,0.2,0.8)	U(0,0.5)	U(0.5,1)	3.19	8	4.82	16	64.4	77
36-40	0.8	(0.7,0.3,0.3)	(0.3,0.3,0.7)	U(0,0.5)	U(0.5,1)	3.45	9	6.39	22	86.0	101
41-45	0.8	(0.6,0.4,0.4)	(0.4,0.4,0.6)	U(0,0.5)	U(0.5,1)	3.84	9	7.92	34	120.6	140
46-50	0.8	(0.5,0.5,0.5)	(0.5,0.5,0.5)	U(0,0.5)	U(0.5,1)	4.21	9	9.23	37	150.0	164

(1) Discounting factor

(2) Probabilities of generating (left, middle, right) children of a node in different clusters for the corresponding instances

(3) Probability distributions for generating edge weights in different clusters for the corresponding instances

(4) Average and maximum number of levels of all trees in the corresponding instances

(5) Average and maximum number of edges in all trees in the corresponding instances

(6) Average and maximum number of edges of support trees of the corresponding instances

Table 4.4: The properties of randomly generated problem instances in LargeDataW

Instance	$\gamma^{(1)}$	$p_1^{(2)}, p_2^{(2)}$	$p_3^{(2)}, p_4^{(2)}$	$w_1^{(3)}, w_3^{(3)}$	$w_2^{(3)}, w_4^{(3)}$	# of Levels$^{(4)}$		# of Edges$^{(6)}$		$ne^{(6)}$	
						Avg.	Max.	Avg.	Max.	Avg.	Max.
1-5	0.7	(0.9,0.1,0.1)	(0.1,0.1,0.9)	U(0,0.5)	U(0.5,1)	2.29	7	2.79	10	33.40	42
6-10	0.7	(0.8,0.2,0.2)	(0.2,0.2,0.8)	U(0,0.5)	U(0.5,1)	2.55	7	3.66	13	52.80	60
11-15	0.7	(0.7,0.3,0.3)	(0.3,0.3,0.7)	U(0,0.5)	U(0.5,1)	2.71	7	4.41	17	61.20	80
16-20	0.7	(0.6,0.4,0.4)	(0.4,0.4,0.6)	U(0,0.5)	U(0.5,1)	2.83	7	4.82	25	75.40	84
21-25	0.7	(0.5,0.5,0.5)	(0.5,0.5,0.5)	U(0,0.5)	U(0.5,1)	3.07	7	5.59	27	83.60	91
26-30	0.9	(0.9,0.1,0.1)	(0.1,0.1,0.9)	U(0,0.5)	U(0.5,1)	4.14	12	5.51	24	103.20	122
31-35	0.9	(0.8,0.2,0.2)	(0.2,0.2,0.8)	U(0,0.5)	U(0.5,1)	4.75	13	8.85	64	238.40	274
36-40	0.9	(0.7,0.3,0.3)	(0.3,0.3,0.7)	U(0,0.5)	U(0.5,1)	5.25	14	11.56	71	350.60	427
41-45	0.9	(0.6,0.4,0.4)	(0.4,0.4,0.6)	U(0,0.5)	U(0.5,1)	6.22	15	17.88	88	583.80	628
46-50	0.9	(0.5,0.5,0.5)	(0.5,0.5,0.5)	U(0,0.5)	U(0.5,1)	6.97	17	21.62	84	716.00	865

(1) Discounting factor

(2) Probabilities of generating (left, middle, right) children of a node in different clusters for the corresponding instances

(3) Probability distributions for generating edge weights in different clusters for the corresponding instances

(4) Average and maximum number of levels of all trees in the corresponding instances

(5) Average and maximum number of edges in all trees in the corresponding instances

(6) Average and maximum number of edges of support trees of the corresponding instances

The average number of levels in those trees is 2.10 while the number of levels in the maximum leveled tree is 6. Trees have 2.53 edges on the average while there are 9 edges in the largest tree. There are 22.6 edges, on the average, in the support trees of those problem instances and the maximum sized support tree has 29 edges.

4.6.1.1 Performance Measures

For different parametric settings, we solve each problem instance 10 times (i.e., make 10 replications) and compute the percent deviations of the objective function values from the best solution (among all replications of all different parametric settings) for that problem instance. Moreover, we calculate the rand index, which is explained in Section 2.6.1, for each replication by assuming that the desired partition of trees in the problem instance is the partition in the data generation procedure.

For a single problem instance, we consider five performance measures; namely average percent deviation of the objective function value across the replications, average rand index across the replications, percent deviation of the best replication, rand index of the best replication, and the total computation time for the replications. We can mathematically state these performance measures as follows. Let D_{ir} be the percent deviation of the objective function value obtained for problem instance i in replication r from the best objective function value obtained for the problem instance. Also, let R_{ir} represent the rand index of the solution for problem instance i in replication r. Then, the first and second performance measures are follows.

$$AvPD_i = \frac{\sum_{r=1}^{10} D_{ir}}{10}$$

$$AvRI_i = \frac{\sum_{r=1}^{10} R_{ir}}{10}$$

Let the best replication for problem instance i be the replication with the minimum percent deviation, i.e., $r_i = \operatorname{argmin}_r D_{ir}$. Then, the third and fourth performance measures are given below.

$$BPD_i = D_{i,r_i}$$

$$BRI_i = R_{i,r_i}$$

Finally, let S_{ir} be the solution time (in seconds) of problem instance i in replication r. Then, the last performance measures is

$$CT_i = \sum_{r=1}^{10} S_{ir}.$$

Note that the problem instances in Tables 4.1-4.4 are divided into 10 batches of size 5 such that the generation parameters are the same in a batch. For example, problem instances 1 through 5 constitute a batch while problem instances 6 through 10 form another batch. In the following subsections, we report the average performance over the problem instances with the same generation parameters. By this way, we aim to see the effect of different tree topologies and different edge weights on the clustering performance. Thus, the performance measures for each batch, $b \in \{1, \ldots, 10\}$, are

$$AvPD = \frac{\sum_i AvPD_i}{5}, \;\; AvRI = \frac{\sum_i AvRI_i}{5},$$

$$BPD = \frac{\sum_i BPD_i}{5}, \;\; BRI = \frac{\sum_i BRI_i}{5}, \;\; CT = \frac{\sum_i CT_i}{5},$$

where $i \in \{5(b-1)+1, \ldots, 5b\}$.

4.6.1.2 Results for tree-kmeans-UWVEO

For tree-kmeans-UWVEO algorithm, there are not any parameters to be decided. The details of the algorithm are given in Algorithm 12.

Tables 4.5 and 4.6 report the performance of tree-kmeans-UWVEO on SmallDataUW and LargeDataUW, respectively. Average percent deviation (AvPD) values are small. Also, it seems that AvPD values are not affected by the topological separability of the problem instances, i.e., there are not visible trends in the values such as "they increase when problem instances become topologically less separable." Average rand index (AvRI) values decrease when the problem instances become topologically less separable. For example, for the instances 26-30 of SmallDataUW in which trees of different clusters can be separable as left and right aligned, AvRI value is 0.91 while it is 0.50 for the instances 46-50 of SmallDataUW in which all trees are random. Actually, the latter setting is used for control purposes and its AvRI is not expected to be larger than 0.5. Best solution's percent deviation (BPD) values are all 0 by definition. As AvRI values, best solution's rand index (BRI) values are smaller for the problem instances which are topologically less separable. Also note that BRI values are larger than AvRI values which indicates that the solutions that are better in terms of objective value perform well in terms of an external evaluation measure such as rand index. Note that problem instances with larger indexes have larger support

Algorithm 12 tree-kmeans-UWVEO

```
Initialization: Obtain initial centroid trees.
Obtain support tree, $ST = (SV, SE)$ where $SV = \bigcup_{i=1}^{i=nt} V_i, SE = \bigcup_{i=1}^{i=nt} E_i$, and let $ne = |SE|$.
Let $ce_j^{(c)} = 0, j = 1, \ldots, ne, c = 1, \ldots, k$.
for j=1 to $ne$ do
    $condFreq_j = \frac{\sum_{i=1}^{i=nt} e_{ij}}{nt} / \frac{\sum_{i=1}^{i=nt} e_{ij'}}{nt}$ where $j \in ch(j')$.
    for c=1 to $k$ do
        if $rand() <= condFreq_j$ then
            $ce_j^{(c)} = 1$
        end if
    end for
end for
repeat
    Assignment: Assign each tree $T_i \in P$ to the most similar cluster.
    Let $C_c = \{\}, c = 1, \ldots, k$.
    for i=1 to $nt$ do
        Find the most similar cluster $c^*$, where $c^* = \text{argmax}_c \frac{2\sum_{j=1}^{ne} e_{ij} ce_j^{(c)} + 1}{2\sum_{j=1}^{ne} e_{ij} + 2\sum_{j=1}^{ne} ce_j^{(c)} + 2}$ and
        let $T_i \in C_{c^*}$.
    end for
    Update: Update each centroid tree by considering trees assigned to it.
    Let $max_{f_{VEO}}^{(c)} = 0, c = 1, \ldots, k$, and $ce_j^{(c)} = 0, j = 1, \ldots, ne, c = 1, \ldots, k$.
    for c=1 to $k$ do
        for s=1 to $ne$ do
            Let $ce_j^{(c)*} = 0, j = 1, \ldots, ne$ and $f_{VEO}^{(c)} = 0$.
            for j=1 to $ne$ do
                Calculate the coefficient of $ce_j^{(c)}$, $coef_j^{(c)} = \sum_{i:T_i \in C_c} \frac{2e_{ij}}{2\sum_{j=1}^{ne} e_{ij} + 2s + 2}$
            end for
            Sort coefficients in descending order and let $coef_{j_1}^{(c)}, ..., coef_{j_{ne}}^{(c)}$ be the ordering.
            for t=1 to $s$ do
                Set $ce_{j_t}^{(c)*} = 1$ and $f_{VEO}^{(c)} = f_{VEO}^{(c)} + coef_{j_t}^{(c)}$
            end for
            $f_{VEO}^{(c)} = f_{VEO}^{(c)} + \sum_{i:T_i \in C_c} \frac{1}{2\sum_{j=1}^{ne} e_{ij} + 2s + 2}$
            if $f_{VEO}^{(c)} \geq max_{f_{VEO}}^{(c)}$ then
                $ce_j^{(c)} = ce_j^{(c)*}, j = 1, \ldots, ne$
                $max_{f_{VEO}}^{(c)} = f_{VEO}^{(c)}$
            end if
        end for
    end for
until Assignments do not change.
return Partition $\{C_1, \ldots, C_k\}$ of $P$.
```

Table 4.5: The performance of tree-kmeans-UWVEO on SmallDataUW

Instance	AvPD	AvRI	BPD	BRI	CT
1-5	2.36	0.85	0.00	0.92	0.91
6-10	2.73	0.69	0.00	0.80	1.25
11-15	2.84	0.57	0.00	0.60	1.74
16-20	2.31	0.50	0.00	0.51	1.82
21-25	2.21	0.50	0.00	0.51	2.31
26-30	1.82	0.91	0.00	0.95	3.13
31-35	2.60	0.73	0.00	0.81	4.81
36-40	2.20	0.55	0.00	0.56	11.58
41-45	2.12	0.54	0.00	0.58	19.52
46-50	2.12	0.50	0.00	0.50	39.26

Table 4.6: The performance of tree-kmeans-UWVEO on LargeDataUW

Instance	AvPD	AvRI	BPD	BRI	CT
1-5	0.76	0.88	0.00	0.89	2.61
6-10	1.62	0.71	0.00	0.74	5.11
11-15	1.96	0.58	0.00	0.60	10.21
16-20	2.36	0.51	0.00	0.51	10.68
21-25	1.82	0.50	0.00	0.49	14.53
26-30	1.25	0.91	0.00	0.95	28.64
31-35	2.98	0.69	0.00	0.79	109.86
36-40	1.96	0.55	0.00	0.60	414.50
41-45	1.15	0.50	0.00	0.49	1203.53
46-50	1.93	0.50	0.00	0.50	3799.06

trees which bring higher computational burden. Thus, total solution time (CT) values increase when we go down the rows of the tables.

4.6.1.3 Results for tree-kmeans-UWGED

There is a single parameter to be decided in tree-kmeans-UWGED algorithm. It is related with the decision on the inclusion of the edges for which the median is 0.5 in the update step. We call design factor EU (initials of "Edge Update") from this point on for which we try the following two alternatives,

1. Do not include
2. Include

In the first alternative, we do not include any of the edges for which the median is 0.5. Similarly, in the second alternative, we include all such edges.

Tables 4.7 and 4.8 report the results for different levels of EU on SmallDataUW and LargeDataUW, respectively. To decide the parametric setting giving the best performance, we use Wilcoxon signed rank test for the pairwise comparison of different parametric settings. Wilcoxon signed rank test is a non-parametric test used to compare two alternatives, X and Y, by testing the following hypothesis.

$H_0 : E(X) = E(Y)$
$H_1 : E(X) \neq E(Y)$ or $H_1 : E(X) - E(Y) > 0$ or $H_1 : E(X) - E(Y) < 0$

Let x_i, $i = 1, \ldots, n$, and y_i, $i = 1, \ldots, n$, be the observation values of X and Y, respectively. To apply Wilcoxon signed rank test, the differences between observations should be calculated and ranked in descending order based on the absolute values, i.e., $|x_i - y_i|$ values should be ranked in descending order. Let r_i, $i = 1, \ldots, n$, be the rank of i^{th} observation. Then, the sign of $x_i - y_i$ should be assigned to r_i. Last, the test statistic is calculated by summing the signed ranks, i.e., $w_0 = r_i + \ldots, r_n$. If the hypothesis is two-sided, null hypothesis is rejected when $w_0 > w_{1-\alpha/2}$ or $w_0 < w_{\alpha/2}$. If the hypothesis is right-tailed, i.e. $H_1 : E(X) - E(Y) > 0$, we reject the null hypothesis when $w_0 > w_{1-\alpha}$. On the other hand, $w_0 < w_\alpha$ is the rejection criteria when the hypothesis is left-tailed. Note that α is the user defined significance level of the hypothesis test and we take it as 5%. The formula for critical value is as follows: $w_\alpha = z_\alpha\sqrt{\frac{n(n+1)(2n+1)}{6}}$.

Table 4.7: The performance of tree-kmeans-UWGED on SmallDataUW

Instance	EU	AvPD	AvRI	BPD	BRI	CT
1-5	1	8.21	0.88	0.00	0.93	0.33
	2	22.03	0.80	0.00	0.93	0.32
6-10	1	8.44	0.70	0.00	0.82	0.33
	2	8.81	0.69	0.00	0.82	0.32
11-15	1	6.12	0.58	0.00	0.59	0.34
	2	7.52	0.56	0.00	0.59	0.35
16-20	1	5.58	0.50	0.14	0.51	0.36
	2	6.72	0.50	0.18	0.51	0.36
21-25	1	5.69	0.50	0.00	0.50	0.39
	2	6.18	0.50	0.00	0.49	0.35
26-30	1	5.50	0.88	0.00	0.95	0.38
	2	13.98	0.82	0.00	0.95	0.37
31-35	1	6.82	0.66	0.00	0.80	0.42
	2	8.91	0.63	0.00	0.78	0.39
36-40	1	2.88	0.56	0.00	0.59	0.47
	2	5.20	0.52	0.31	0.59	0.43
41-45	1	3.09	0.52	0.16	0.54	0.51
	2	3.82	0.51	0.00	0.52	0.50
46-50	1	1.59	0.49	0.06	0.49	0.59
	2	2.08	0.49	0.13	0.49	0.58

Suppose that we are trying to select best parametric setting in terms of AvPD. Table 4.9 summarizes the application of Wilcoxon signed rank test to compare parametric setting alternatives, EU is 1 and EU is 2, for SmallDataUW. As it can be seen from the table, the test statistic is -55. If $H_1 : E(X) - E(Y) > 0$, the critical value is 32.38. Thus, we fail to reject $H_0 : E(X) = E(Y)$. On the other hand, when $H_1 : E(X) - E(Y) < 0$ the critical value is -32.38. In that case, we reject $H_0 : E(X) = E(Y)$ and accept $H_1 : E(X) - E(Y) < 0$. This means that the algorithm has better performance when EU is 1 since smaller AvPD values are desired. In other words, EU=1 setting dominates EU=2 setting. Table 4.10 reports the nondominated parametric settings for tree-kmeans-UWGED in terms of each performance measure on each dataset. As it can be seen from the table, the algorithm has better performance

Table 4.8: The performance of tree-kmeans-UWGED on LargeDataUW

Instance	EU	AvPD	AvRI	BPD	BRI	CT
1-5	1	12.34	0.80	0.00	0.92	0.51
	2	17.22	0.75	0.00	0.89	0.46
6-10	1	8.78	0.66	0.00	0.75	0.54
	2	10.23	0.63	0.00	0.75	0.53
11-15	1	4.46	0.56	0.00	0.61	0.58
	2	5.15	0.56	0.00	0.61	0.56
16-20	1	3.70	0.51	0.00	0.51	0.64
	2	4.59	0.51	0.00	0.51	0.60
21-25	1	3.34	0.50	0.00	0.50	0.67
	2	4.45	0.49	0.34	0.49	0.63
26-30	1	8.84	0.82	0.00	0.94	0.67
	2	9.34	0.81	0.00	0.93	0.68
31-35	1	5.85	0.61	0.00	0.78	0.80
	2	6.13	0.59	0.04	0.77	0.76
36-40	1	2.37	0.53	0.48	0.62	1.06
	2	2.86	0.52	0.50	0.56	1.04
41-45	1	2.04	0.50	0.85	0.50	1.29
	2	2.19	0.50	0.00	0.50	1.32
46-50	1	2.86	0.50	0.14	0.49	1.99
	2	2.77	0.49	0.46	0.49	2.02

when the first parametric setting (EU=1) is used. The details of the algorithm are given in Algorithm 13.

If we investigate the rows of 4.7 and 4.8 in which EU is 1, we can observe some trends as in previous section. AvPD and AvRI values seem to be affected by the topological separability of the problem instances. AvPD increases and AvRI decreases when problem instances become topologically less separable. BPD values are very small (0 in most of the cases). This indicates that the selected parametric setting gives the near best performance (the best performance in most of the cases). BRI values are larger for the problem instances which are topologically more separable. Also note that the solutions that are better in terms of objective value are generally better in terms of rand index since BRI values are larger than AvRI values in general. Finally,

Table 4.9: Wilcoxon signed rank test in terms of AvPD on SmallDataUW for tree-kmeans-UWGED

Observation (Instance batch)	$EU = 1$ (X)	$EU = 2$ (Y)	Difference (X-Y)	Rank	Signed Rank (R)
1-5	8.21	22.03	-13.82	1	-1
6-10	8.44	8.81	-0.38	10	-10
11-15	6.12	7.52	-1.41	5	-5
16-20	5.58	6.72	-1.15	6	-6
21-25	5.69	6.18	-0.49	9	-9
26-30	5.50	13.98	-8.48	2	-2
31-35	6.82	8.91	-2.09	4	-4
36-40	2.88	5.20	-2.32	3	-3
41-45	3.09	3.82	-0.73	7	-7
46-50	1.59	2.08	-0.49	8	-8

Table 4.10: Nondominated parametric settings for tree-kmeans-UWGED (1 means nondominated; 0 means dominated)

Dataset	Performance Measure	Setting	
		1	2
SmallDataUW	AvPD	1	0
	AvRI	1	0
	BPD	1	1
	BRI	1	1
	CT	1	0
LargeDataUW	AvPD	1	0
	AvRI	1	0
	BPD	1	1
	BRI	1	0
	CT	1	1
Total		10	4

CT values increase with the size of the support tree.

Algorithm 13 tree-kmeans-UWGED

```
*Initialization: Obtain initial centroid trees.*
Obtain support tree, $ST = (SV, SE)$ where $SV = \bigcup_{i=1}^{i=nt} V_i, SE = \bigcup_{i=1}^{i=nt} E_i$, and let $ne = |SE|$.
Let $ce_j^{(c)} = 0, j = 1, \ldots, ne, c = 1, \ldots, k$.
for j=1 to $ne$ do
    $condFreq_j = \frac{\sum_{i=1}^{i=nt} e_{ij}}{nt} / \frac{\sum_{i=1}^{i=nt} e_{ij'}}{nt}$ where $j \in ch(j')$.
    for c=1 to $k$ do
        if $rand() <= condFreq_j$ then
            $ce_j^{(c)} = 1$
        end if
    end for
end for
repeat
    *Assignment: Assign each tree $T_i \in P$ to the most similar cluster.*
    Let $C_c = \{\}, c = 1, \ldots, k$.
    for i=1 to $nt$ do
        Find the most similar cluster $c^*$, where $c^* = \text{argmin}_c \sum_{j=1}^{ne} \left| e_{ij} - ce_j^{(c)} \right|$ and let $T_i \in C_{c^*}$.
    end for
    *Update: Update each centroid tree by considering trees assigned to it.*
    Let $ce_j^{(c)} = 0, j = 1, \ldots, ne, c = 1, \ldots, k$.
    for c=1 to $k$ do
        for j=1 to $ne$ do
            if $\frac{\sum_{i:T_i \in C_c} e_{ij}}{|C_c|} > 0.5$ then
                $ce_j^{(c)} = 1, j = 1, \ldots, ne$.
            end if
        end for
    end for
until Assignments do not change.
return Partition $\{C_1, \ldots, C_k\}$ of $P$.
```

4.6.1.4 Results for tree-kmeans-WVEO

There are three decisions to be made in tree-kmeans-WVEO algorithm. First one is related with the initial centroid trees. While constructing the initial centroid trees, in addition to the decision on the inclusion of the edges for which we use frequency based random selection as explained before, we need to decide on the weights of the edges. This design factor will be called WSI (initials of "Weight Selection in Initialization") after now and we try the following two alternatives for the weight of an edge in an initial centroid tree.

1. Mean of nonzero weights in the problem instance

2. Median of nonzero weights in the problem instance

Next decision is the starting weights of the edges in the update step. We call this design factor SWU (initials of "Starting Weight in Update") and the following three alternatives are tested.

1. Weights of previous iteration
2. Mean of nonzero weights in the cluster
3. Median of nonzero weights in the cluster

The last one is the order of edges whose weights to be optimized while keeping other edges' weights fixed. We call this design factor SEU (initials of "Starting Edge in Update") and the following three alternatives are tested.

1. Fixed
2. Descending in weight
3. Descending in frequency

In the first alternative, we keep order of the edges fixed during the algorithm. The order is the order of the edges in the support tree. In the second alternative, we reorder the edges according to their weights in each optimization cycle. In the last alternative, we order edges according to their appearance frequencies in the cluster and the order may change during the algorithm since clusters change.

Tables 4.11 and 4.12 report the results for different factor level combinations on SmallDataW and LargeDataW, respectively. Table 4.13 provides the nondominated parametric settings. According to this table, setting 6 (WSI=1, SWU=2, SEU=3) and setting 13 (WSI=2, SWU=2, SEU=1) perform better than other settings. Selecting SWU=2 is common in both of the alternatives. Thus, we can fix this selection. Between WSI=1 and WSI=2, we choose WSI=1 since it is more concordant with SWU=2 selection. SEU=1 and SEU=3 actually imply similar things. If we use order of the edges in the support tree (SEU=1), we implicitly consider the frequencies of

Table 4.11: The performance of tree-kmeans-WVEO on SmallDataW

Ins.	WSI	SWU	SEU	AvPD	AvRI	BPD	BRI	CT
1-5	1	1	1	3.66	0.76	0.80	0.78	1.31
	1	1	2	4.48	0.76	0.65	0.79	0.86
	1	1	3	4.28	0.75	1.39	0.78	1.16
	1	2	1	5.24	0.75	1.61	0.78	2.13
	1	2	2	7.66	0.75	2.93	0.76	0.90
	1	2	3	4.27	0.76	0.20	0.80	2.10
	1	3	1	3.95	0.76	0.64	0.78	1.92
	1	3	2	6.90	0.74	2.35	0.77	1.05
	1	3	3	3.59	0.76	0.54	0.78	1.99
	2	1	1	3.97	0.76	0.10	0.80	1.23
	2	1	2	3.51	0.77	0.60	0.79	0.95
	2	1	3	4.84	0.75	1.25	0.77	1.17
	2	2	1	4.13	0.76	0.75	0.79	2.05
	2	2	2	7.05	0.75	3.18	0.77	0.99
	2	2	3	4.17	0.76	0.94	0.77	1.99
	2	3	1	3.71	0.76	0.71	0.78	1.96
	2	3	2	7.21	0.76	2.65	0.79	0.99
	2	3	3	4.44	0.76	0.64	0.78	2.02
6-10	1	1	1	2.94	0.69	0.41	0.71	1.48
	1	1	2	2.97	0.70	0.74	0.72	1.22
	1	1	3	2.56	0.70	0.34	0.71	1.49
	1	2	1	2.33	0.70	0.47	0.73	2.74
	1	2	2	8.33	0.68	3.99	0.68	1.44
	1	2	3	2.26	0.70	0.50	0.71	2.96
	1	3	1	2.57	0.70	0.83	0.73	2.75
	1	3	2	8.98	0.68	4.24	0.71	1.32
	1	3	3	1.95	0.70	0.39	0.71	2.93
	2	1	1	2.69	0.69	0.38	0.72	1.48
	2	1	2	2.99	0.69	0.43	0.71	1.20
	2	1	3	3.03	0.69	0.76	0.71	1.46
	2	2	1	2.47	0.70	0.37	0.71	3.04
	2	2	2	8.52	0.69	4.19	0.71	1.25
	2	2	3	2.54	0.70	0.36	0.72	2.85
	2	3	1	3.21	0.69	0.41	0.72	2.73
	2	3	2	8.52	0.68	3.41	0.70	1.25
	2	3	3	2.39	0.70	0.45	0.72	3.04
11-15	1	1	1	3.89	0.68	1.21	0.69	1.99
	1	1	2	3.90	0.68	1.48	0.68	1.62
	1	1	3	4.49	0.67	0.52	0.69	2.06
	1	2	1	4.23	0.68	0.76	0.69	3.74
	1	2	2	8.91	0.67	5.30	0.68	1.64
	1	2	3	3.28	0.68	0.55	0.70	4.05
	1	3	1	4.39	0.68	0.82	0.70	3.98
	1	3	2	9.42	0.66	5.28	0.68	1.81
	1	3	3	3.25	0.69	0.47	0.71	4.26
	2	1	1	3.91	0.68	0.36	0.70	1.99
	2	1	2	4.19	0.68	0.55	0.70	1.48
	2	1	3	4.01	0.68	0.34	0.70	2.05
	2	2	1	3.28	0.69	0.27	0.70	3.88
	2	2	2	8.89	0.67	4.55	0.70	1.55
	2	2	3	3.55	0.68	1.38	0.68	4.03
	2	3	1	3.86	0.68	0.46	0.70	3.93
	2	3	2	8.44	0.67	4.90	0.70	1.79
	2	3	3	4.08	0.68	1.75	0.69	4.07
16-20	1	1	1	4.11	0.65	0.77	0.66	2.62
	1	1	2	4.38	0.65	1.08	0.66	2.03
	1	1	3	4.67	0.64	1.36	0.66	2.69
	1	2	1	4.06	0.64	1.54	0.66	5.56
	1	2	2	11.15	0.64	6.92	0.64	2.23
	1	2	3	3.92	0.65	0.94	0.66	5.13
	1	3	1	4.15	0.65	0.58	0.66	5.24
	1	3	2	9.79	0.64	5.28	0.68	2.03
	1	3	3	4.53	0.65	0.42	0.67	5.74
	2	1	1	5.14	0.64	1.91	0.65	2.58
	2	1	2	4.45	0.65	0.80	0.65	2.03
	2	1	3	4.15	0.65	0.55	0.66	2.74
	2	2	1	3.87	0.65	1.28	0.66	5.71
	2	2	2	11.60	0.64	5.87	0.66	1.97
	2	2	3	4.29	0.65	0.96	0.67	5.86
	2	3	1	4.37	0.65	1.34	0.66	5.62
	2	3	2	9.90	0.65	5.14	0.65	2.42
	2	3	3	4.43	0.64	0.67	0.65	5.96
21-25	1	1	1	3.41	0.64	0.43	0.65	2.69
	1	1	2	3.85	0.64	0.68	0.64	1.80
	1	1	3	3.66	0.64	0.73	0.65	2.62
	1	2	1	3.56	0.64	0.46	0.66	5.21
	1	2	2	8.45	0.63	4.91	0.64	1.98
	1	2	3	3.14	0.64	0.37	0.65	5.35
	1	3	1	3.43	0.64	0.40	0.65	5.87
	1	3	2	8.56	0.63	3.65	0.65	2.17
	1	3	3	4.18	0.63	1.08	0.65	5.73
	2	1	1	3.85	0.64	1.15	0.64	2.82
	2	1	2	3.47	0.64	0.55	0.65	1.83
	2	1	3	3.74	0.64	0.62	0.65	2.72
	2	2	1	3.53	0.64	0.79	0.64	5.49
	2	2	2	8.17	0.63	4.02	0.66	1.99
	2	2	3	4.10	0.64	1.22	0.64	5.19
	2	3	1	3.44	0.64	0.80	0.64	5.45
	2	3	2	7.83	0.64	3.79	0.64	1.92
	2	3	3	3.68	0.64	1.19	0.66	5.86
26-30	1	1	1	5.22	0.75	1.71	0.78	1.95
	1	1	2	4.38	0.76	0.57	0.78	1.47
	1	1	3	4.30	0.76	0.85	0.79	2.04
	1	2	1	5.69	0.75	1.77	0.80	3.58
	1	2	2	9.70	0.74	4.67	0.77	1.43
	1	2	3	3.75	0.76	0.52	0.78	3.84
	1	3	1	4.62	0.75	1.20	0.78	3.63
	1	3	2	8.82	0.75	3.87	0.76	1.54
	1	3	3	3.94	0.75	0.33	0.78	4.11
	2	1	1	4.23	0.76	1.12	0.78	2.03
	2	1	2	4.79	0.75	0.86	0.78	1.54
	2	1	3	3.94	0.76	0.01	0.80	2.01
	2	2	1	3.68	0.76	1.23	0.80	4.05
	2	2	2	8.87	0.75	4.13	0.76	1.74
	2	2	3	4.37	0.76	0.13	0.79	4.33
	2	3	1	4.00	0.76	0.71	0.80	3.92
	2	3	2	7.89	0.76	3.54	0.78	1.66
	2	3	3	4.59	0.75	0.52	0.78	4.18
31-35	1	1	1	4.18	0.71	0.90	0.72	3.36
	1	1	2	3.99	0.70	1.29	0.70	2.36
	1	1	3	4.44	0.70	1.63	0.75	3.67
	1	2	1	4.27	0.70	1.54	0.70	6.50
	1	2	2	11.30	0.69	6.71	0.72	2.63
	1	2	3	4.18	0.70	0.81	0.72	6.79
	1	3	1	4.66	0.69	0.73	0.71	7.15
	1	3	2	9.92	0.70	4.48	0.71	2.44
	1	3	3	3.97	0.70	0.65	0.71	7.71
	2	1	1	4.20	0.69	0.69	0.70	3.20
	2	1	2	4.02	0.70	0.67	0.72	2.45
	2	1	3	3.92	0.70	0.45	0.72	3.19
	2	2	1	4.12	0.71	0.48	0.72	7.15
	2	2	2	10.59	0.69	5.29	0.70	2.63
	2	2	3	4.04	0.70	0.57	0.73	7.68
	2	3	1	3.83	0.71	0.59	0.72	7.03
	2	3	2	12.04	0.68	5.87	0.70	2.66
	2	3	3	4.78	0.70	0.70	0.71	7.28
36-40	1	1	1	4.82	0.66	1.84	0.68	5.52
	1	1	2	4.61	0.66	0.88	0.68	3.75
	1	1	3	5.02	0.66	1.35	0.68	5.37
	1	2	1	4.31	0.66	0.83	0.69	11.16
	1	2	2	14.19	0.64	8.89	0.66	3.64
	1	2	3	4.41	0.67	0.51	0.71	11.40
	1	3	1	4.49	0.67	0.63	0.69	10.43
	1	3	2	13.83	0.64	8.15	0.66	3.56
	1	3	3	4.37	0.67	0.39	0.71	10.99
	2	1	1	5.39	0.66	0.45	0.69	5.30
	2	1	2	5.19	0.67	1.74	0.67	3.80
	2	1	3	4.56	0.67	0.68	0.69	5.33
	2	2	1	4.41	0.67	0.45	0.70	10.97
	2	2	2	13.39	0.66	8.33	0.68	3.98
	2	2	3	4.67	0.67	1.07	0.67	11.03
	2	3	1	4.55	0.66	1.00	0.68	11.11
	2	3	2	13.57	0.65	9.20	0.68	3.48
	2	3	3	4.73	0.67	1.25	0.69	11.27
41-45	1	1	1	5.62	0.64	1.24	0.66	6.91
	1	1	2	4.48	0.65	1.66	0.67	4.84
	1	1	3	4.11	0.65	1.28	0.68	6.93
	1	2	1	4.25	0.65	0.98	0.67	14.40
	1	2	2	14.62	0.63	9.77	0.66	5.18
	1	2	3	4.14	0.65	0.91	0.67	16.03
	1	3	1	4.25	0.65	0.95	0.68	14.44
	1	3	2	12.51	0.64	7.85	0.68	5.50
	1	3	3	4.50	0.65	1.04	0.67	15.45
	2	1	1	4.40	0.65	1.11	0.66	7.44
	2	1	2	4.15	0.65	0.98	0.67	5.05
	2	1	3	3.95	0.65	0.91	0.67	7.26
	2	2	1	4.15	0.65	0.36	0.67	14.28
	2	2	2	13.34	0.64	7.02	0.66	4.86
	2	2	3	4.72	0.64	1.88	0.65	13.60
	2	3	1	4.38	0.65	0.76	0.67	14.27
	2	3	2	13.53	0.64	8.64	0.66	4.86
	2	3	3	4.31	0.65	1.37	0.66	15.89
46-50	1	1	1	3.94	0.64	1.30	0.64	9.46
	1	1	2	4.08	0.64	0.81	0.65	6.35
	1	1	3	4.54	0.65	1.75	0.64	9.20
	1	2	1	5.55	0.64	1.96	0.64	18.69
	1	2	2	13.62	0.63	9.06	0.64	6.23
	1	2	3	4.84	0.64	1.48	0.66	19.42
	1	3	1	6.09	0.63	3.00	0.65	19.73
	1	3	2	14.05	0.63	9.47	0.64	6.29
	1	3	3	5.01	0.64	1.07	0.65	19.13
	2	1	1	4.25	0.64	0.66	0.67	9.76
	2	1	2	5.25	0.64	1.63	0.67	6.04
	2	1	3	4.35	0.65	0.92	0.66	9.32
	2	2	1	5.15	0.64	2.09	0.66	20.20
	2	2	2	14.13	0.63	8.62	0.64	5.56
	2	2	3	5.46	0.63	1.59	0.67	19.35
	2	3	1	5.19	0.64	0.96	0.65	20.04
	2	3	2	15.08	0.63	10.34	0.64	6.01
	2	3	3	5.06	0.64	1.08	0.66	19.99

Table 4.12: The performance of tree-kmeans-WVEO on LargeDataW

Ins.	WSI	SWU	SEU	AvPD	AvRI	BPD	BRI	CT
1-5	1	1	1	4.05	0.73	0.78	0.76	3.42
	1	1	2	5.14	0.73	0.99	0.76	2.70
	1	1	3	4.38	0.73	0.23	0.77	3.15
	1	2	1	4.05	0.74	0.50	0.76	6.34
	1	2	2	10.36	0.72	4.77	0.74	2.61
	1	2	3	4.64	0.73	0.98	0.76	7.30
	1	3	1	4.77	0.74	0.34	0.76	6.91
	1	3	2	10.27	0.73	6.29	0.73	2.49
	1	3	3	4.33	0.73	0.93	0.76	7.34
	2	1	1	4.27	0.73	0.89	0.75	3.51
	2	1	2	4.39	0.74	0.32	0.75	2.48
	2	1	3	3.54	0.74	0.60	0.75	3.50
	2	2	1	3.54	0.74	0.00	0.77	6.49
	2	2	2	9.43	0.73	5.29	0.75	3.05
	2	2	3	3.62	0.74	0.41	0.75	7.50
	2	3	1	3.93	0.74	0.39	0.77	6.76
	2	3	2	9.98	0.73	3.90	0.73	2.16
	2	3	3	3.68	0.74	0.11	0.77	7.80
6-10	1	1	1	2.17	0.72	0.04	0.73	5.31
	1	1	2	2.34	0.71	0.40	0.72	3.68
	1	1	3	2.49	0.72	0.33	0.72	4.87
	1	2	1	2.28	0.71	0.42	0.72	10.33
	1	2	2	12.54	0.69	7.74	0.71	4.04
	1	2	3	2.70	0.71	0.13	0.72	10.40
	1	3	1	3.13	0.71	0.18	0.72	10.50
	1	3	2	12.68	0.70	6.36	0.73	3.73
	1	3	3	2.40	0.72	0.21	0.73	11.81
	2	1	1	2.80	0.72	0.20	0.72	4.97
	2	1	2	2.54	0.72	0.28	0.73	3.79
	2	1	3	2.88	0.72	0.19	0.73	5.34
	2	2	1	2.02	0.71	0.31	0.72	10.58
	2	2	2	12.21	0.70	6.77	0.71	4.74
	2	2	3	2.45	0.72	0.07	0.73	11.17
	2	3	1	2.46	0.72	0.42	0.72	11.82
	2	3	2	12.66	0.70	6.43	0.71	4.39
	2	3	3	2.34	0.72	0.04	0.73	11.34
11-15	1	1	1	3.45	0.67	0.47	0.68	6.57
	1	1	2	2.88	0.68	0.79	0.67	4.53
	1	1	3	3.82	0.67	0.42	0.68	6.24
	1	2	1	2.99	0.67	0.19	0.68	14.87
	1	2	2	13.59	0.66	9.76	0.66	5.44
	1	2	3	3.60	0.67	0.41	0.69	14.96
	1	3	1	3.57	0.67	0.30	0.68	15.18
	1	3	2	11.42	0.66	7.36	0.67	4.92
	1	3	3	3.30	0.67	0.71	0.68	16.05
	2	1	1	3.57	0.67	0.63	0.68	6.22
	2	1	2	4.76	0.67	0.34	0.68	4.68
	2	1	3	4.16	0.67	0.83	0.69	6.42
	2	2	1	3.84	0.67	0.59	0.68	14.41
	2	2	2	12.98	0.66	9.56	0.70	4.78
	2	2	3	2.61	0.67	0.43	0.70	17.55
	2	3	1	3.26	0.68	0.37	0.68	14.69
	2	3	2	12.19	0.66	7.72	0.66	4.94
	2	3	3	3.16	0.68	0.50	0.69	16.02
16-20	1	1	1	2.84	0.64	0.39	0.64	8.68
	1	1	2	3.96	0.64	0.70	0.65	5.21
	1	1	3	3.17	0.65	0.36	0.65	8.79
	1	2	1	3.43	0.64	0.40	0.64	19.21
	1	2	2	16.35	0.63	11.98	0.63	5.60
	1	2	3	3.95	0.64	0.40	0.65	19.67
	1	3	1	3.15	0.65	0.34	0.65	19.15
	1	3	2	16.17	0.62	9.95	0.63	5.84
	1	3	3	3.39	0.64	0.74	0.65	22.01
	2	1	1	3.22	0.64	0.44	0.64	8.01
	2	1	2	3.38	0.64	0.57	0.64	6.40
	2	1	3	3.24	0.64	0.39	0.64	8.50
	2	2	1	3.23	0.64	0.43	0.64	18.35
	2	2	2	14.98	0.62	11.26	0.63	6.17
	2	2	3	3.12	0.65	0.32	0.65	21.94
	2	3	1	3.61	0.64	0.52	0.64	18.44
	2	3	2	14.84	0.63	10.52	0.64	7.82
	2	3	3	3.99	0.64	0.75	0.65	20.05
21-25	1	1	1	3.50	0.64	0.98	0.66	9.23
	1	1	2	3.30	0.65	0.87	0.66	6.62
	1	1	3	2.86	0.64	0.43	0.65	10.08
	1	2	1	3.23	0.65	0.40	0.66	21.59
	1	2	2	13.96	0.63	10.33	0.65	7.12
	1	2	3	3.79	0.64	0.40	0.66	22.67
	1	3	1	2.63	0.65	0.31	0.66	21.63
	1	3	2	13.71	0.64	10.39	0.64	7.56
	1	3	3	3.61	0.64	0.43	0.65	24.32
	2	1	1	2.73	0.65	0.36	0.65	9.77
	2	1	2	3.15	0.65	0.46	0.65	6.47
	2	1	3	2.54	0.65	0.77	0.64	9.60
	2	2	1	3.10	0.65	0.53	0.67	22.21
	2	2	2	14.72	0.63	11.49	0.63	7.07
	2	2	3	3.11	0.65	0.56	0.65	22.68
	2	3	1	2.98	0.65	0.84	0.67	20.77
	2	3	2	13.42	0.63	8.51	0.65	7.13
	2	3	3	3.26	0.65	0.40	0.65	23.86
26-30	1	1	1	3.94	0.74	0.75	0.76	8.25
	1	1	2	4.26	0.75	0.61	0.77	6.13
	1	1	3	4.98	0.74	0.72	0.75	8.57
	1	2	1	3.51	0.75	0.86	0.76	18.34
	1	2	2	12.75	0.74	8.62	0.73	8.69
	1	2	3	4.10	0.74	0.69	0.76	19.17
	1	3	1	4.04	0.74	0.60	0.77	18.51
	1	3	2	13.21	0.74	9.13	0.75	6.24
	1	3	3	4.38	0.74	0.58	0.75	19.52
	2	1	1	4.06	0.74	1.15	0.75	9.15
	2	1	2	3.71	0.75	0.40	0.77	6.46
	2	1	3	4.12	0.74	0.77	0.75	7.78
	2	2	1	3.51	0.75	0.96	0.77	19.56
	2	2	2	12.34	0.74	8.25	0.74	6.71
	2	2	3	3.53	0.75	0.86	0.77	22.88
	2	3	1	4.47	0.74	0.86	0.75	19.84
	2	3	2	12.43	0.74	7.16	0.76	7.13
	2	3	3	4.01	0.74	0.93	0.76	20.29
31-35	1	1	1	3.75	0.72	0.77	0.74	25.58
	1	1	2	3.87	0.72	0.49	0.73	16.99
	1	1	3	3.70	0.72	0.38	0.73	27.42
	1	2	1	4.59	0.71	0.44	0.73	62.19
	1	2	2	23.60	0.69	15.49	0.72	16.62
	1	2	3	3.52	0.72	0.25	0.73	65.40
	1	3	1	3.67	0.72	0.57	0.74	62.45
	1	3	2	23.08	0.68	16.90	0.70	19.45
	1	3	3	4.62	0.71	1.04	0.73	67.75
	2	1	1	4.40	0.71	0.88	0.73	24.70
	2	1	2	4.28	0.72	0.44	0.72	17.35
	2	1	3	3.75	0.72	0.38	0.74	25.22
	2	2	1	3.86	0.72	0.51	0.73	59.38
	2	2	2	22.18	0.68	17.89	0.70	18.26
	2	2	3	3.35	0.72	0.67	0.73	68.63
	2	3	1	4.92	0.71	0.39	0.71	58.12
	2	3	2	23.16	0.67	16.20	0.70	18.26
	2	3	3	4.72	0.71	0.88	0.71	66.98
36-40	1	1	1	4.69	0.66	0.78	0.68	41.25
	1	1	2	4.69	0.66	0.59	0.67	28.22
	1	1	3	4.16	0.66	0.32	0.67	45.47
	1	2	1	4.85	0.66	0.34	0.67	104.46
	1	2	2	26.49	0.64	20.51	0.66	30.20
	1	2	3	5.16	0.66	0.49	0.66	110.95
	1	3	1	5.09	0.66	0.72	0.68	98.21
	1	3	2	27.38	0.63	21.37	0.65	27.21
	1	3	3	6.56	0.65	1.10	0.68	99.76
	2	1	1	5.00	0.66	0.64	0.67	40.98
	2	1	2	4.52	0.66	0.81	0.66	27.22
	2	1	3	5.37	0.66	0.73	0.66	41.33
	2	2	1	4.79	0.66	0.57	0.66	105.05
	2	2	2	27.08	0.64	20.77	0.66	29.29
	2	2	3	3.96	0.66	0.49	0.67	110.42
	2	3	1	5.54	0.66	0.60	0.68	102.39
	2	3	2	27.51	0.64	22.23	0.66	26.94
	2	3	3	5.44	0.66	0.61	0.67	110.17
41-45	1	1	1	3.89	0.63	0.90	0.65	98.09
	1	1	2	5.37	0.63	1.35	0.65	56.42
	1	1	3	4.12	0.64	1.04	0.63	96.93
	1	2	1	5.62	0.62	1.11	0.64	218.50
	1	2	2	24.77	0.62	17.08	0.65	52.88
	1	2	3	4.71	0.62	0.92	0.64	254.91
	1	3	1	4.77	0.63	0.69	0.64	228.74
	1	3	2	24.60	0.62	16.82	0.63	51.56
	1	3	3	4.74	0.63	1.47	0.64	241.24
	2	1	1	4.27	0.63	0.74	0.65	100.79
	2	1	2	5.85	0.62	1.86	0.63	59.06
	2	1	3	3.74	0.63	1.10	0.64	87.03
	2	2	1	4.63	0.63	0.51	0.65	249.88
	2	2	2	25.60	0.61	19.69	0.62	54.32
	2	2	3	4.58	0.63	1.43	0.64	240.17
	2	3	1	4.19	0.63	1.00	0.65	234.36
	2	3	2	25.61	0.61	18.27	0.64	52.04
	2	3	3	4.98	0.63	0.92	0.65	256.54
46-50	1	1	1	3.86	0.63	1.13	0.64	135.68
	1	1	2	4.69	0.63	1.22	0.64	87.31
	1	1	3	4.34	0.63	0.44	0.63	140.09
	1	2	1	4.99	0.63	1.22	0.64	331.43
	1	2	2	22.67	0.61	16.34	0.64	64.50
	1	2	3	4.90	0.63	1.63	0.64	352.23
	1	3	1	4.68	0.63	0.73	0.64	317.34
	1	3	2	23.91	0.61	18.79	0.63	65.15
	1	3	3	4.85	0.63	0.84	0.64	317.41
	2	1	1	4.17	0.63	0.66	0.63	144.05
	2	1	2	5.10	0.63	1.66	0.64	82.09
	2	1	3	4.31	0.63	0.59	0.65	142.24
	2	2	1	4.45	0.63	1.09	0.66	324.50
	2	2	2	23.00	0.61	17.27	0.64	67.05
	2	2	3	4.12	0.63	1.27	0.63	323.89
	2	3	1	6.23	0.63	2.25	0.63	326.70
	2	3	2	22.92	0.61	17.88	0.64	63.76
	2	3	3	4.98	0.62	1.23	0.66	347.81

Table 4.13: Nondominated parametric settings for tree-kmeans-WVEO (1 means nondominated; 0 means dominated)

Dataset	Performance Measure	Setting																	
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
SmallDataW	AvPD	1	1	0	0	0	1	0	0	1	0	0	1	1	0	0	1	0	0
	AvRI	1	0	1	0	0	1	0	0	1	0	1	1	1	0	0	1	0	1
	BPD	0	0	0	0	0	1	0	0	1	1	1	1	1	0	0	1	0	0
	BRI	0	0	1	1	0	1	1	0	1	1	1	1	1	0	1	1	0	0
	CT	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	1
LargeDataW	AvPD	1	0	1	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0
	AvRI	1	1	1	0	0	0	1	0	0	0	0	1	1	0	1	1	0	0
	BPD	0	0	1	0	0	1	1	0	0	0	1	0	1	0	0	0	0	0
	BRI	1	1	0	1	0	0	1	0	0	0	0	0	1	0	1	1	0	1
	CT	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	1
Total		5	3	5	2	0	7	4	0	6	2	4	6	8	0	6	6	0	4

the edges. The possibility of an edge which is at higher levels of support tree having larger frequency is high. Thus, SEU=1 and SEU=3 are not conflicting and we can arbitrarily select one of them. All in one, we decide to continue with setting 6. The details of the algorithm are given in Algorithm 14.

When we analyze the performance of the algorithm with the selected parametric setting, we can obtain the following results. AvPD and BPD values do not seem to be affected by the topological separability of the problem instances. They do not gradually increase/decrease with the topological changes. On the other hand, AvRI and BRI values are larger for the problem instances which are topologically more separable. Also, BRI values are larger than AvRI values. This is a sign of that the solutions which are better in terms of objective value are generally better in terms of rand index. Finally, solving the problem instances with larger support trees is computationally more demanding as CT values increase with the size of the support tree.

4.6.1.5 Results for tree-kmeans-WGED

There are three decisions in tree-kmeans-WGED algorithm. First one is the selection of weights in initial random centroid trees (WSI). We try the same alternatives explained in previous section. The second one is the decision on the inclusion of the edges, which appear exactly half of the trees in the cluster, in the centroid tree in update step (EU). We try the same levels explained previously. Last decision is on the

Algorithm 14 tree-kmeans-WVEO

```
*Initialization: Obtain initial centroid trees.*
Obtain support tree, $ST = (SV, SE)$ where $SV = \bigcup_{i=1}^{i=nt} V_i, SE = \bigcup_{i=1}^{i=nt} E_i$, and let $ne = |SE|$.
Let $ce_j^{(c)} = 0, j = 1, \ldots, ne, c = 1, \ldots, k$, and $cw_{jt}^{(c)} = 0, j = 1, \ldots, ne, t = 1, \ldots, na, c = 1, \ldots, k$
for j=1 to $ne$ do
    $condFreq_j = \frac{\sum_{i=1}^{i=nt} e_{ij}}{nt} / \frac{\sum_{i=1}^{i=nt} e_{ij'}}{nt}$ where $j \in ch(j')$.
    for c=1 to $k$ do
        if $rand() <= condFreq_j$ then
            $ce_j^{(c)} = 1$
            for t=1 to $na$ do
                $cw_{jt}^{(c)} = \frac{\sum_{i:e_{ij}=1} w_{ijt} e_{ij}}{\sum_{i:e_{ij}=1} e_{ij}}$
            end for
        end if
    end for
end for
repeat
    *Assignment: Assign each tree $T_i \in P$ to the most similar cluster.*
    Let $C_c = \{\}, c = 1, \ldots, k$.
    for i=1 to $nt$ do
        Find the most similar cluster $c^*$, where $c^* = \text{argmax}_c \sum_{t=1}^{na} \frac{\sum_{j=1}^{ne} \min(w_{ijt}, cw_{jt}^{(c)})}{\sum_{j=1}^{ne} w_{ijt} + \sum_{j=1}^{ne} cw_{jt}^{(c)}}$ and let $T_i \in C_{c^*}$.
    end for
    *Update: Update each centroid tree by considering trees assigned to it.*
    Let $ce_j^{(c)} = 0, j = 1, \ldots, ne, c = 1, \ldots, k$.
    for c=1 to $k$ do
        for t=1 to $na$ do
            for j=1 to $ne$ do
                Let $cw_{jt}^{(c)} = \frac{\sum_{i:T_i \in C_c \& e_{ij}=1} w_{ijt} e_{ij}}{\sum_{i:T_i \in C_c \& e_{ij}=1} e_{ij}}$, and $cw_{jt}^{(c)*} = cw_{jt}^{(c)}$.
            end for
            repeat
                Sort edges in descending order with respect to their frequencies in $C_c$ and let $j_1, ..., j_{ne}$ be the ordering.
                for s=1 to $ne$ do
                    Let $max_f = 0$
                    for $i : T_i \in C_c$ do
                        $cw_{j_s t}^{(c)} = w_{i j_s t}$
                        $f = \frac{\sum_{j=1}^{ne} \min(w_{ijt}, cw_{jt}^{(c)})}{\sum_{j=1}^{ne} w_{ijt} + \sum_{j=1}^{ne} cw_{jt}^{(c)}}$
                        if $f \geq max_f$ then
                            $cw_{j_s t}^{(c)*} = w_{i j_s t}$
                            $max_f = f$
                        end if
                    end for
                    $cw_{j_s t}^{(c)} = cw_{j_s t}^{(c)*}$
                end for
            until All $cw_{jt}^{(c)}$ values do not change.
        end for
    end for
until Assignments do not change.
return Partition $\{C_1, \ldots, C_k\}$ of $P$.
```

weights of the edges, for which we need to take median of even number of values, in the centroid tree. For those edges' weights there exist infinitely many medians. This issue is actually problematic for the edges which appear in exactly half of the trees in the cluster. In that case, values in the middle will be 0 and a positive value, let say b. It is clear that if the edge exists in a tree, its weight is greater than or equal to b or it does not exist at all. Taking a value between 0 and b as median will not represent the data well. We will call this factor WU (initials of "Weight Update") and we try the following two alternatives. Assume that a and b are the values in the middle.

1. a
2. b

Tables 4.14 and 4.15 report the results of tree-kmeans-WGED for different factor level combinations on SmallDataW and LargeDataW, respectively. Table 4.16 provides the nondominated parametric settings based on the results in Tables 4.14 and 4.15. Setting 1 (WSI=1, WU=1, EU=1), Setting 5 (WSI=2, WU=1, EU=1) and Setting 6 (WSI=2, WU=1, EU=2) have the same nondomination results and perform better than other settings. Since WU=1 selection is common in all alternatives we can fix this selection. Then, EU=1 and EU=2 imply the same numerical values. If we choose EU=1, the corresponding edge will not be included in the centroid tree. If we choose EU=2, the corresponding edge will be included in the centroid tree but its weight will be selected as 0. Thus, we can arbitrarily select one of EU=1 and EU=2. Since EU=1 appears in two of three alternatives and it is also selected in Section 4.6.1.3, we continue with EU=1 selection. Since taking median of weights is the optimal solution for a given cluster, we believe that taking the median of weights in initial centroid trees is more appropriate than taking the mean. All in all, we choose Setting 5. The details of the algorithm are given in Algorithm 15.

Analysis of the performance of the algorithm with the selected parametric setting reveals the following observations. As tree-kmeans-WGED algorithm, AvPD and BPD values do not seem to increase/decrease by the topological separability of the problem instances. It is expected because in the weighted problem instances the only source of complication is not the topology. Weights also have important effect in the objective function. However, AvRI and BRI values increase with the topological separability of

Table 4.14: The performance of tree-kmeans-WGED on SmallDataW

Ins.	WSI	WU	EU	AvPD	AvRI	BPD	BRI	CT
1-5	1	1	1	11.31	0.73	0.26	0.78	0.75
	1	1	2	11.75	0.73	0.55	0.76	0.80
	1	2	1	10.69	0.73	1.62	0.75	0.81
	1	2	2	11.42	0.71	0.99	0.75	0.72
	2	1	1	12.02	0.74	0.35	0.78	0.76
	2	1	2	10.17	0.75	0.64	0.78	0.79
	2	2	1	11.39	0.72	2.70	0.78	0.74
	2	2	2	12.10	0.70	1.69	0.77	0.77
6-10	1	1	1	10.37	0.62	3.32	0.67	0.70
	1	1	2	9.81	0.62	3.80	0.64	0.75
	1	2	1	8.82	0.63	0.84	0.65	0.68
	1	2	2	9.36	0.60	1.90	0.64	0.73
	2	1	1	10.16	0.62	3.32	0.69	0.74
	2	1	2	10.16	0.62	1.92	0.65	0.74
	2	2	1	9.66	0.62	1.61	0.65	0.72
	2	2	2	11.24	0.57	3.33	0.68	0.66
11-15	1	1	1	10.78	0.62	3.76	0.63	0.78
	1	1	2	10.63	0.62	0.85	0.63	0.74
	1	2	1	10.07	0.62	1.86	0.63	0.82
	1	2	2	9.82	0.59	2.15	0.64	0.77
	2	1	1	10.38	0.63	2.66	0.61	0.78
	2	1	2	12.33	0.63	2.40	0.67	0.76
	2	2	1	9.99	0.60	1.88	0.62	0.71
	2	2	2	10.31	0.57	2.84	0.64	0.70
16-20	1	1	1	5.77	0.61	2.33	0.61	0.90
	1	1	2	6.53	0.61	1.68	0.67	0.82
	1	2	1	5.53	0.59	0.84	0.62	0.81
	1	2	2	6.89	0.55	1.15	0.60	0.75
	2	1	1	5.95	0.61	1.89	0.63	0.83
	2	1	2	6.58	0.61	0.81	0.63	0.83
	2	2	1	6.76	0.60	2.45	0.61	0.81
	2	2	2	6.11	0.56	1.09	0.62	0.79
21-25	1	1	1	8.35	0.59	2.86	0.60	0.84
	1	1	2	7.86	0.59	1.35	0.59	0.76
	1	2	1	8.12	0.56	2.12	0.55	0.84
	1	2	2	7.47	0.54	0.90	0.58	0.75
	2	1	1	7.53	0.58	1.89	0.55	0.82
	2	1	2	8.96	0.58	4.37	0.58	0.77
	2	2	1	8.14	0.58	3.14	0.60	0.81
	2	2	2	8.12	0.53	3.08	0.59	0.74

Ins.	WSI	SWU	SEU	AvPD	AvRI	BPD	BRI	CT
26-30	1	1	1	11.06	0.67	2.78	0.69	0.78
	1	1	2	11.91	0.69	2.88	0.74	0.75
	1	2	1	8.86	0.66	0.70	0.71	0.83
	1	2	2	8.91	0.65	0.98	0.68	0.75
	2	1	1	13.91	0.67	3.50	0.75	0.88
	2	1	2	9.12	0.71	0.77	0.75	0.87
	2	2	1	12.50	0.66	4.09	0.69	0.81
	2	2	2	11.64	0.67	1.66	0.72	0.84
31-35	1	1	1	9.79	0.62	3.59	0.70	0.88
	1	1	2	9.48	0.64	3.78	0.63	0.81
	1	2	1	8.29	0.60	0.88	0.65	0.86
	1	2	2	9.51	0.54	2.62	0.54	0.82
	2	1	1	10.11	0.61	2.87	0.58	0.80
	2	1	2	10.43	0.62	4.33	0.67	0.82
	2	2	1	8.93	0.60	1.41	0.66	0.90
	2	2	2	7.91	0.59	0.63	0.59	0.82
36-40	1	1	1	10.16	0.56	2.81	0.50	0.95
	1	1	2	11.11	0.57	4.99	0.51	0.98
	1	2	1	9.35	0.53	1.51	0.47	0.99
	1	2	2	8.74	0.51	3.21	0.56	0.96
	2	1	1	10.39	0.58	4.24	0.52	0.94
	2	1	2	11.33	0.55	4.52	0.49	0.86
	2	2	1	10.54	0.52	3.10	0.46	0.92
	2	2	2	9.09	0.51	2.57	0.52	0.90
41-45	1	1	1	10.05	0.57	0.90	0.47	1.14
	1	1	2	10.80	0.56	3.39	0.52	1.14
	1	2	1	10.51	0.54	3.72	0.52	1.17
	1	2	2	9.73	0.49	2.08	0.41	1.08
	2	1	1	12.28	0.56	4.39	0.55	1.30
	2	1	2	11.93	0.56	6.10	0.45	1.14
	2	2	1	10.83	0.54	3.76	0.52	1.07
	2	2	2	8.39	0.49	1.11	0.44	1.11
46-50	1	1	1	9.56	0.54	2.02	0.47	1.34
	1	1	2	9.37	0.56	3.51	0.56	1.14
	1	2	1	8.32	0.53	1.35	0.46	1.31
	1	2	2	8.37	0.50	2.35	0.41	1.30
	2	1	1	9.27	0.56	2.33	0.49	1.35
	2	1	2	9.97	0.54	2.44	0.46	1.20
	2	2	1	9.31	0.54	4.61	0.47	1.28
	2	2	2	8.23	0.48	1.77	0.49	1.22

Table 4.15: The performance of tree-kmeans-WGED on LargeDataW

Ins.	WSI	WU	EU	AvPD	AvRI	BPD	BRI	CT
1-5	1	1	1	10.25	0.67	0.92	0.71	1.13
	1	1	2	11.29	0.67	1.48	0.75	1.12
	1	2	1	11.99	0.64	1.80	0.74	1.16
	1	2	2	10.81	0.66	1.03	0.75	1.23
	2	1	1	8.25	0.69	1.27	0.75	1.23
	2	1	2	9.17	0.70	1.21	0.75	1.23
	2	2	1	9.18	0.69	1.67	0.73	1.13
	2	2	2	12.30	0.66	0.80	0.75	1.04
6-10	1	1	1	5.91	0.67	0.22	0.70	1.22
	1	1	2	6.24	0.66	0.32	0.71	1.32
	1	2	1	6.83	0.65	1.19	0.70	1.20
	1	2	2	7.03	0.62	0.57	0.71	1.19
	2	1	1	6.77	0.66	1.15	0.70	1.19
	2	1	2	6.43	0.67	0.59	0.69	1.24
	2	2	1	6.36	0.67	0.94	0.67	1.22
	2	2	2	6.92	0.63	1.38	0.68	1.23
11-15	1	1	1	5.63	0.59	1.81	0.60	1.30
	1	1	2	5.56	0.61	0.42	0.61	1.13
	1	2	1	5.93	0.58	1.88	0.58	1.20
	1	2	2	6.95	0.53	3.01	0.57	1.18
	2	1	1	6.09	0.60	1.74	0.67	1.23
	2	1	2	6.25	0.59	2.14	0.65	1.20
	2	2	1	6.21	0.57	1.87	0.62	1.22
	2	2	2	5.71	0.54	1.51	0.62	1.25
16-20	1	1	1	8.66	0.55	3.44	0.55	1.34
	1	1	2	8.23	0.56	2.99	0.56	1.27
	1	2	1	8.41	0.55	2.36	0.59	1.27
	1	2	2	8.05	0.53	3.12	0.56	1.19
	2	1	1	8.14	0.57	2.86	0.58	1.30
	2	1	2	6.18	0.56	0.85	0.59	1.34
	2	2	1	6.73	0.57	2.00	0.58	1.27
	2	2	2	7.31	0.53	2.36	0.49	1.36
21-25	1	1	1	5.99	0.57	2.51	0.59	1.35
	1	1	2	5.51	0.58	2.65	0.56	1.53
	1	2	1	5.70	0.56	1.31	0.55	1.31
	1	2	2	5.05	0.52	1.14	0.52	1.38
	2	1	1	6.27	0.58	3.47	0.55	1.31
	2	1	2	6.29	0.56	1.63	0.54	1.36
	2	2	1	6.18	0.56	2.69	0.58	1.24
	2	2	2	5.66	0.53	1.83	0.55	1.33

Ins.	WSI	SWU	SEU	AvPD	AvRI	BPD	BRI	CT
26-30	1	1	1	7.37	0.69	1.59	0.74	1.59
	1	1	2	8.02	0.68	1.61	0.71	1.45
	1	2	1	9.22	0.62	1.80	0.72	1.54
	1	2	2	8.31	0.64	2.02	0.75	1.42
	2	1	1	7.51	0.67	2.03	0.73	1.48
	2	1	2	6.67	0.68	0.91	0.71	1.46
	2	2	1	7.32	0.67	1.18	0.69	1.57
	2	2	2	7.63	0.65	2.05	0.73	1.43
31-35	1	1	1	10.46	0.59	3.13	0.59	1.91
	1	1	2	10.89	0.58	4.13	0.53	2.07
	1	2	1	10.99	0.55	3.68	0.50	1.83
	1	2	2	8.96	0.53	0.70	0.55	1.83
	2	1	1	10.49	0.58	2.64	0.60	1.92
	2	1	2	9.76	0.55	2.59	0.52	1.93
	2	2	1	10.02	0.55	2.70	0.46	1.81
	2	2	2	10.05	0.49	3.10	0.51	1.81
36-40	1	1	1	11.94	0.49	2.83	0.43	2.30
	1	1	2	12.32	0.52	3.45	0.46	2.28
	1	2	1	10.98	0.45	2.26	0.41	2.24
	1	2	2	9.99	0.44	2.05	0.35	2.17
	2	1	1	11.67	0.51	1.27	0.42	2.22
	2	1	2	11.81	0.52	3.74	0.44	2.28
	2	2	1	11.91	0.47	2.74	0.41	2.04
	2	2	2	11.65	0.48	2.38	0.39	2.22
41-45	1	1	1	10.33	0.49	3.55	0.43	3.41
	1	1	2	11.13	0.49	5.84	0.42	3.26
	1	2	1	10.71	0.47	3.83	0.47	3.91
	1	2	2	10.38	0.42	2.49	0.29	3.60
	2	1	1	11.20	0.50	5.43	0.52	4.00
	2	1	2	11.38	0.48	6.22	0.47	3.42
	2	2	1	11.18	0.45	3.81	0.35	3.42
	2	2	2	9.98	0.40	2.89	0.32	3.53
46-50	1	1	1	11.84	0.46	3.92	0.36	4.81
	1	1	2	12.29	0.47	5.60	0.46	5.07
	1	2	1	11.71	0.44	2.85	0.39	4.36
	1	2	2	10.43	0.41	3.21	0.29	4.64
	2	1	1	12.82	0.47	4.12	0.36	4.67
	2	1	2	13.13	0.48	4.77	0.40	4.85
	2	2	1	12.87	0.45	5.10	0.38	4.53
	2	2	2	11.22	0.42	3.12	0.35	4.49

Table 4.16: Nondominated parametric settings for tree-kmeans-WGED (1 means nondominated; 0 means dominated)

Dataset	Performance Measure	Setting							
		1	2	3	4	5	6	7	8
SmallDataW	AvPD	0	0	1	1	0	0	0	1
	AvRI	1	1	0	0	1	1	0	0
	BPD	0	0	1	1	0	0	0	1
	BRI	1	1	0	0	1	1	0	0
	CT	1	0	1	0	1	1	0	0
LargeDataW	AvPD	1	1	0	1	1	1	1	1
	AvRI	1	1	0	0	1	1	0	0
	BPD	1	1	1	1	1	1	1	1
	BRI	1	1	1	0	1	1	0	0
	CT	1	1	0	0	1	1	0	1
Total		8	7	5	4	8	8	2	5

the problem instances in general. Unlike the previous algorithms' results, BRI values are not always larger than AvRI values. In other words, the solutions which are better in terms of objective value may not be better in terms of rand index. This is the result of the nonlinear objective function provided in Section 4.4.4. Finally, CT values increase when we go down the rows of the result tables. Thus, we can deduce that the solution time required to solve the problem instances with larger support trees are greater than time required for instances with smaller support trees.

4.6.2 Comparison of tree-kmeans-UWVEO, tree-kmeans-UWGED and kmodes

After the construction of the support tree, remember that we can represent the edge set of tree T_i as a vector, namely $\overrightarrow{E_i} = [e_{ij}]$, $j \in \{1, \ldots, ne\}$ where where e_{ij} is a binary parameter representing the existence of j^{th} edge of the support tree in T_i. With this representation, we can consider each tree as a point in $\mathbb{R}^{ne}$. As it is mentioned before, kmeans algorithm is the most commonly used algorithm to group given points into clusters. However, it is mostly suitable for numerical data. In our case $\overrightarrow{E_i}$ is a categorical data. Thus, to group such points into clusters, Huang proposes the

Algorithm 15 tree-kmeans-WGED

```
Initialization: Obtain initial centroid trees.
Obtain support tree, $ST = (SV, SE)$ where $SV = \bigcup_{i=1}^{i=nt} V_i, SE = \bigcup_{i=1}^{i=nt} E_i$, and let $ne = |SE|$.
Let $ce_j^{(c)} = 0, j = 1, \ldots, ne, c = 1, \ldots, k$, and $cw_{jt}^{(c)} = 0, j = 1, \ldots, ne, t = 1, \ldots, na, c = 1, \ldots, k$
for j=1 to $ne$ do
    $condFreq_j = \frac{\sum_{i=1}^{i=nt} e_{ij}}{nt} / \frac{\sum_{i=1}^{i=nt} e_{ij'}}{nt}$ where $j \in ch(j')$.
    for c=1 to $k$ do
        if $rand() <= condFreq_j$ then
            $ce_j^{(c)} = 1$
            for t=1 to $na$ do
                Sort $w_{ijt}$ values, where $i : e_{ij} = 1$, in ascending order and let $cw_{jt}^{(c)}$ be $(\left\lfloor \frac{\sum_{i:e_{ij}=1} e_{ij}+1}{2} \right\rfloor)^{st}$
                value of the ordered list
            end for
        end if
    end for
end for
repeat
    Assignment: Assign each tree $T_i \in P$ to the most similar cluster.
    Let $C_c = \{\}, c = 1, \ldots, k$.
    for i=1 to $nt$ do
        Find the most similar cluster $c^*$, where $c^* = \text{argmin}_c \sum_{t=1}^{na} \sum_{j=1}^{ne} \left| w_{ijt} - cw_{jt}^{(c)} \right|$ and
        let $T_i \in C_{c^*}$.
    end for
    Update: Update each centroid tree by considering trees assigned to it.
    for c=1 to $k$ do
        Let $ce_j^{(c)} = 0, j = 1, \ldots, ne$, and $cw_{jt}^{(c)} = 0, j = 1, \ldots, ne, t = 1, \ldots, na$.
        for j=1 to $ne$ do
            if $\frac{\sum_{i:T_i \in C_c} e_{ij}}{|C_c|} > 0.5$ then
                $ce_j^{(c)} = 1, j = 1, \ldots, ne$
                Sort $w_{ijt}$ values, where $i : T_i \in C_c$, in ascending order and let $cw_{jt}^{(c)}$ be $(\left\lfloor \frac{|C_c|+1}{2} \right\rfloor)^{st}$ value of
                the ordered list
            end if
        end for
    end for
until Assignments do not change.
return Partition $\{C_1, \ldots, C_k\}$ of $P$.
```

kmodes algorithm [80]. This algorithm is an extension of kmeans for categorical domain. It tries to minimize the sum of Hamming distances between data objects and cluster centroids they are assigned to. Assume that we aim to group categorical data

Algorithm 16 kmodes

```
Start with initial random cluster centroids, $cent_j = [c_{sj}]$.
Let $C_j = \emptyset$, for all $j \in \{1, \dots, k\}$.
for i=1 to n do
    Assign $X_i$ to the closest cluster $j^*$, where $j^* = \text{argmin}_j \sum_{s=1}^{d} 1_{x_{si} \neq c_{sj}}$
    and let $X_i \in C_{j^*}$
    Update $cent_{j^*}$ by letting $c_{sj}$ values be the mode of $x_{si}$ where $i : X_i \in C_{j^*}$
end for
repeat
    for i=1 to n do
        Assume that $X_i \in C_{j_i}$ and find $C_{j^*}$, where $j^* = \text{argmin}_j \sum_{s=1}^{d} 1_{x_{si} \neq c_{sj}}$
        if $j_i \neq j^*$ then
            Let $C_{j_i} = C_{j_i} \setminus X_i$ and update $cent_{j_i}$ by letting $c_{sj_i}$ values be the mode of
            $x_{si}$ where $i : X_i \in C_{j_i}$
            Let $C_{j^*} = C_{j^*} \bigcup X_i$ and update $cent_{j^*}$ by letting $c_{sj^*}$ values be the mode
            of $x_{si}$ where $i : X_i \in C_{j^*}$
        end if
    end for
until Assignments do not change in a full cycle.
return Partition $\{C_1, \dots, C_k\}$.
```

objects $X_i = [x_{si}]$, $i = 1, \dots, n$, $s = 1, \dots, d$, into k clusters whose centroids are $cent_j = [c_{sj}]$, $j = 1, \dots, k$, $s = 1, \dots, d$, respectively. Let $I_{x_{si} \neq c_{sj}}$ be the indicator function taking value 1 when $x_{si} \neq c_{sj}$. Then, Algorithms 16 shows the detail of kmodes.

First, in order to validate tree-kmeans-UWVEO, for each other algorithm and each instance, the best solution found is taken and the corresponding VEO objective function value is computed. If tree-kmeans-UWVEO does what it is intended to do, we expect that the best solution found by it has a higher VEO value than the VEO values of the best solutions found by the other algorithms. For this purpose, for each instance, we select the best solution (among 10 replications) found by each algorithm. We compute the VEO objective values of these solutions and choose the one with the maximum VEO value. We find the percent deviation of the VEO value of the best solution found by each algorithm from this overall maximum. We then report the average percent deviation over the instances in a batch. We follow a similar logic for the tree-kmeans-UWGED, taking the minimum of the GED values of the best solutions found by the algorithms.

Table 4.17 provides the comparison of the algorithms in terms of the VEO objective. The tree-kmeans-UWVEO always finds the best solutions in terms of the VEO objective as expected. Moreover, the solutions obtained by the tree-kmeans-UWGED usually have a higher VEO value than those obtained by the kmodes. Also, the average percent deviations presented in the table for tree-kmeans-UWGED and kmodes tend to increase as the problem size increases and as the topological separability decreases.

Table 4.17: Comparison of tree-kmeans-UWVEO, tree-kmeans-UWGED and kmodes with respect to BPD of UWVEO objective

SmallDataUW	**Instance**	**1-5**	**6-10**	**11-15**	**16-20**	**21-25**	**Average**
	tree-kmeans-UWVEO	0.00	0.00	0.00	0.00	0.00	0.00
	tree-kmeans-UWGED	0.22	0.56	0.86	1.84	6.59	2.01
	k-modes	0.22	0.56	4.26	1.81	5.38	2.44
	Instance	**26-30**	**31-35**	**36-40**	**41-45**	**46-50**	**Average**
	tree-kmeans-UWVEO	0.00	0.00	0.00	0.00	0.00	0.00
	tree-kmeans-UWGED	0.00	2.19	5.39	7.75	11.15	5.30
	k-modes	0.00	5.80	7.62	11.48	24.13	9.80
LargeDataUW	**Instance**	**1-5**	**6-10**	**11-15**	**16-20**	**21-25**	**Average**
	tree-kmeans-UWVEO	0.00	0.00	0.00	0.00	0.00	0.00
	tree-kmeans-UWGED	0.00	1.61	4.14	3.99	7.03	3.35
	k-modes	0.00	3.96	2.60	3.27	8.96	3.76
	Instance	**26-30**	**31-35**	**36-40**	**41-45**	**46-50**	**Average**
	tree-kmeans-UWVEO	0.00	0.00	0.00	0.00	0.00	0.00
	tree-kmeans-UWGED	0.81	3.68	9.26	17.84	23.73	11.06
	k-modes	6.66	17.99	19.82	21.56	25.90	18.38

Table 4.18: Comparison of tree-kmeans-UWGED, tree-kmeans-UWVEO and kmodes with respect to BPD of UWGED objective

	Instance	**1-5**	**6-10**	**11-15**	**16-20**	**21-25**	**Average**
SmallDataUW	tree-kmeans-UWGED	0.00	0.00	0.00	0.00	0.00	0.00
	tree-kmeans-UWVEO	0.00	4.17	2.31	2.22	8.76	3.49
	k-modes	0.00	0.00	4.85	0.34	1.39	1.32
	Instance	**26-30**	**31-35**	**36-40**	**41-45**	**46-50**	**Average**
	tree-kmeans-UWGED	0.00	0.00	0.00	0.00	0.00	0.00
	tree-kmeans-UWVEO	0.00	7.27	13.70	8.37	19.57	9.78
	k-modes	0.00	1.46	2.26	3.89	4.36	2.40
LargeDataUW	**Instance**	**1-5**	**6-10**	**11-15**	**16-20**	**21-25**	**Average**
	tree-kmeans-UWGED	0.00	0.00	0.00	0.00	0.00	0.00
	tree-kmeans-UWVEO	0.00	1.18	11.08	11.40	12.24	7.18
	k-modes	0.00	3.48	0.88	0.08	5.04	1.90
	Instance	**26-30**	**31-35**	**36-40**	**41-45**	**46-50**	**Average**
	tree-kmeans-UWGED	0.00	0.00	0.00	0.00	0.00	0.00
	tree-kmeans-UWVEO	0.17	2.50	8.30	27.26	29.28	13.50
	k-modes	5.70	6.48	4.54	2.38	4.22	4.67

The comparison of the algorithms in terms of the GED objective is given in Table 4.18. In this case, the tree-kmeans-UWGED always finds the best solutions in terms of the GED objective. The solutions obtained by the k-modes are usually better than those obtained by the tree-kmeans-UWVEO. This may be attributed to the fact that even though k-modes and tree-kmeans-UWGED differ in some algorithmic steps, they both consider the same objective function.

In addition to previously defined performance measures, we use new performance measures based on adjusted rand index, f-measure and mutual information. As it is stated before, the rand index is frequently used to measure the agreement between the found partition and the desired partition. However there are some known problems with this index [81]. Therefore as an improvement of the rand index, the adjusted rand index is proposed to overcome some of its limitations [81]. The higher the adjusted rand index, the better the performance. Let $\Gamma = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_k\}$ be the correct/desired partition, $C = \{C_1, C_2, \ldots, C_k\}$ be the found partition, $\varsigma_{j_f}^{j_c}$ be the number of data objects clustered into C_{j_f} while they actually belong to $\mathit{\Gamma}_{j_c}$, $\varsigma_{j_f}^{\bullet}$ be the number of data objects clustered into C_{j_f}, and $\varsigma_{\bullet}^{j_c}$ be the number of data objects belong to $\mathit{\Gamma}_{j_c}$. Then, the adjusted rand index is calculated as

$$ARI = \frac{\sum_{j_f=1}^{k}\sum_{j_c=1}^{k}\binom{\varsigma_{j_f}^{j_c}}{2} - \frac{\sum_{j_f=1}^{k}\binom{\varsigma_{j_f}^{\bullet}}{2}\sum_{j_c=1}^{k}\binom{\varsigma_{\bullet}^{j_c}}{2}}{\binom{n+m}{2}}}{\frac{1}{2}\left(\sum_{j_f=1}^{k}\binom{\varsigma_{j_f}^{\bullet}}{2} + \sum_{j_c=1}^{k}\binom{\varsigma_{\bullet}^{j_c}}{2}\right) - \frac{\sum_{j_f=1}^{k}\binom{\varsigma_{j_f}^{\bullet}}{2}\sum_{j_c=1}^{k}\binom{\varsigma_{\bullet}^{j_c}}{2}}{\binom{n+m}{2}}}. \tag{4.14}$$

F-measure and mutual information are already explained in Section 2.6.1. For a single problem instance, we define adjusted rand index of the best replication ($BARI_i$), f-measure of the best replication (BFM_i), mutual information of the best replication (BMI_i). Let A_{ir}, F_{ir} and M_{ir} be the adjusted rand index, f-measure and mutual information for problem instance i in replication r, respectively. Assume the index of best replication in terms of the objective function value for problem instance i is r_i. Then, new performance measures for a single problem instance are given below.

$$BARI_i = A_{i,r_i}$$

$$BFM_i = F_{i,r_i}$$

$$BMI_i = M_{i,r_i}$$

The new performance measures for each problem instance batch, $b \in \{1, \ldots, 10\}$, are

$$BARI = \frac{\sum_i BARI_i}{5}, \quad BFM = \frac{\sum_i BFM_i}{5}, \quad BMI = \frac{\sum_i BMI_i}{5},$$

where $i \in \{5(b-1)+1, \ldots, 5b\}$.

Table 4.19 reports the values of the external performance measures over the problem instances in each batch. It can be seen from the table that, the performances of tree-kmeans-UWVEO and tree-kmeans-UWGED are very similar and they both outperform k-modes algorithm, especially for the instances in LargeDataUW. Moreover, the performances of all algorithms get worse as the instances become less separable (i.e., as we move from batch 1-5 to 21-25 and from 26-30 to 46-50.).

Table 4.19: Comparison of tree-kmeans-UWVEO, tree-kmeans-UWGED and kmodes with respect to external performance measures

Instance	Algoritmh	SmallDataUW				LargeDataUW			
		BRI	BARI	BFM	BMI	BRI	BARI	BFM	BMI
1-5	tree-kmeans-UWVEO	0.92	0.85	0.92	0.39	0.89	0.78	0.89	0.36
	tree-kmeans-UWGED	0.93	0.86	0.93	0.40	0.92	0.84	0.92	0.38
	kmodes	0.92	0.85	0.92	0.39	0.86	0.73	0.86	0.34
6-10	tree-kmeans-UWVEO	0.80	0.60	0.80	0.28	0.74	0.47	0.74	0.20
	tree-kmeans-UWGED	0.82	0.64	0.82	0.29	0.75	0.51	0.75	0.21
	kmodes	0.82	0.64	0.82	0.29	0.71	0.42	0.74	0.18
11-15	tree-kmeans-UWVEO	0.60	0.19	0.60	0.08	0.60	0.19	0.60	0.08
	tree-kmeans-UWGED	0.59	0.19	0.59	0.08	0.61	0.22	0.62	0.10
	kmodes	0.55	0.10	0.57	0.05	0.57	0.14	0.59	0.06
16-20	tree-kmeans-UWVEO	0.51	0.03	0.54	0.02	0.51	0.02	0.51	0.01
	tree-kmeans-UWGED	0.51	0.02	0.51	0.02	0.51	0.02	0.51	0.02
	kmodes	0.50	0.00	0.52	0.01	0.51	0.03	0.52	0.02
21-25	tree-kmeans-UWVEO	0.51	0.01	0.50	0.01	0.49	-0.01	0.49	0.00
	tree-kmeans-UWGED	0.50	-0.01	0.49	0.01	0.50	-0.01	0.49	0.00
	kmodes	0.50	0.01	0.52	0.01	0.49	-0.01	0.57	0.00
Average (1-25)	**tree-kmeans-UWVEO**	**0.67**	**0.34**	**0.67**	**0.16**	**0.65**	**0.29**	**0.64**	**0.13**
	tree-kmeans-UWGED	**0.67**	**0.34**	**0.67**	**0.16**	**0.66**	**0.32**	**0.66**	**0.14**
	kmodes	**0.66**	**0.32**	**0.67**	**0.15**	**0.63**	**0.26**	**0.66**	**0.12**
26-30	tree-kmeans-UWVEO	0.95	0.90	0.95	0.43	0.95	0.90	0.95	0.43
	tree-kmeans-UWGED	0.95	0.90	0.95	0.43	0.94	0.89	0.94	0.42
	kmodes	0.95	0.90	0.95	0.43	0.87	0.74	0.90	0.36
31-35	tree-kmeans-UWVEO	0.81	0.62	0.80	0.27	0.79	0.59	0.79	0.26
	tree-kmeans-UWGED	0.80	0.59	0.79	0.26	0.78	0.57	0.78	0.25
	kmodes	0.70	0.41	0.74	0.18	0.58	0.17	0.68	0.09
36-40	tree-kmeans-UWVEO	0.56	0.13	0.58	0.06	0.60	0.19	0.60	0.08
	tree-kmeans-UWGED	0.59	0.19	0.59	0.08	0.62	0.24	0.65	0.11
	kmodes	0.55	0.10	0.62	0.05	0.49	0.00	0.65	0.01
41-45	tree-kmeans-UWVEO	0.58	0.17	0.59	0.07	0.49	-0.01	0.50	0.00
	tree-kmeans-UWGED	0.54	0.09	0.55	0.04	0.50	0.01	0.57	0.01
	kmodes	0.50	0.01	0.57	0.01	0.49	0.00	0.63	0.01
46-50	tree-kmeans-UWVEO	0.50	0.00	0.49	0.01	0.50	-0.01	0.51	0.00
	tree-kmeans-UWGED	0.49	-0.01	0.52	0.01	0.49	0.00	0.62	0.01
	kmodes	0.49	0.00	0.62	0.01	0.49	0.00	0.63	0.00
Average (26-50)	**tree-kmeans-UWVEO**	**0.68**	**0.36**	**0.68**	**0.17**	**0.67**	**0.33**	**0.67**	**0.16**
	tree-kmeans-UWGED	**0.68**	**0.35**	**0.68**	**0.16**	**0.67**	**0.34**	**0.71**	**0.16**
	kmodes	**0.64**	**0.28**	**0.70**	**0.14**	**0.58**	**0.18**	**0.70**	**0.09**

Table 4.20: Comparison of tree-kmeans-WVEO, tree-kmeans-WGED and k-means with respect to BPD of WVEO objective

SmallDataW	**Instance**	**1-5**	**6-10**	**11-15**	**16-20**	**21-25**	**Average**
	tree-kmeans-WVEO	0.00	0.00	0.00	0.00	0.00	0.00
	tree-kmeans-WGED	27.63	32.79	34.57	33.18	26.12	30.86
	kmeans	27.61	29.49	30.79	27.11	30.03	29.00
	Instance	**26-30**	**31-35**	**36-40**	**41-45**	**46-50**	**Average**
	tree-kmeans-WVEO	0.00	0.00	0.00	0.00	0.00	0.00
	tree-kmeans-WGED	38.78	35.77	45.66	40.45	45.73	41.28
	kmeans	29.43	35.05	36.93	39.37	35.77	35.31
LargeDataW	**Instance**	**1-5**	**6-10**	**11-15**	**16-20**	**21-25**	**Average**
	tree-kmeans-WVEO	0.00	0.00	0.00	0.00	0.00	0.00
	tree-kmeans–WGED	25.84	33.47	34.14	38.07	43.11	34.93
	kmeans	33.33	32.35	33.29	34.16	35.42	33.71
	Instance	**26-30**	**31-35**	**36-40**	**41-45**	**46-50**	**Average**
	tree-kmeans-WVEO	0.00	0.00	0.00	0.00	0.00	0.00
	tree-kmeans-WGED	36.49	43.05	54.71	57.85	61.13	50.65
	kmeans	33.54	44.45	50.54	50.18	51.08	45.96

4.6.3 Comparison of tree-kmeans-WVEO, tree-kmeans-WGED and kmeans

After the construction of the support tree, remember that we can represent the weights of tree T_i as a vector; namely $\overrightarrow{W_i} = [w_{ijt}]$, $j \in \{1, \ldots, ne\}$, $t \in \{1, \ldots, na\}$ where w_{ijt} is a numerical value representing the weight of j^{th} edge of support tree for attribute t in T_i. With this representation, we can consider each tree as a point in $\mathbb{R}^{ne*na}$. To cluster $\overrightarrow{W_i}$'s we can utilize kmeans algorithm.

To compare tree-kmeans-WVEO and tree-kmeans-WGED with kmeans on SmallDataW and LargeDataW, we follow the same procedure explained in previous part.

First, we compare the algorithms in terms of the VEO and GED objectives in Tables 4.20 and 4.21, respectively. The tree-kmeans-WVEO and the tree-kmeans-WGED always find the best solutions in terms of their own objectives, showing that the proposed solution methods serve their purposes. The k-means algorithm finds the worst solutions in terms of both objectives.

Table 4.21: Comparison of tree-kmeans-WGED, tree-kmeans-WVEO and kmeans with respect to BPD of WGED objective

	Instance	**1-5**	**6-10**	**11-15**	**16-20**	**21-25**	**Average**
SmallDataW	tree-kmeans-WGED	0.00	0.00	0.00	0.00	0.00	0.00
	tree-kmeans-WVEO	18.15	24.29	28.74	31.62	31.81	26.92
	kmeans	44.22	39.15	48.27	49.05	44.76	45.09
	Instance	**26-30**	**31-35**	**36-40**	**41-45**	**46-50**	**Average**
	tree-kmeans-WGED	0.00	0.00	0.00	0.00	0.00	0.00
	tree-kmeans-WVEO	23.41	16.78	39.26	54.10	48.74	36.46
	kmeans	45.80	53.81	56.13	55.38	53.63	52.95
LargeDataW	**Instance**	**1-5**	**6-10**	**11-15**	**16-20**	**21-25**	**Average**
	tree-kmeans-WGED	0.00	0.00	0.00	0.00	0.00	0.00
	tree-kmeans-WVEO	24.76	24.41	23.73	29.81	34.00	27.34
	kmeans	58.30	55.18	53.84	53.47	52.94	54.74
	Instance	**26-30**	**31-35**	**36-40**	**41-45**	**46-50**	**Average**
	tree-kmeans-WGED	0.00	0.00	0.00	0.00	0.00	0.00
	tree-kmeans-WVEO	20.16	31.03	50.18	45.49	64.65	42.30
	kmeans	56.21	61.92	61.12	70.76	70.71	64.14

Second, we compare the algorithms in Table 4.22 in terms of the values of the external performance measures. It can be seen from the table that, the tree-kmeans-WVEO outperforms the other algorithms. Moreover, its performance is stable with respect to increases in the sizes of the trees (as we move from batch 1-5 to 26-30, from 6-10 to 31-35, and so on) and the number of trees in the datasets (as we move from left to right in the table). The performances of tree-kmeans-WGED and kmeans are similar in most of the batches and both of these algorithms perform worse as the sizes of the trees and the number of trees in the dataset increase. Furthermore, the performances of all algorithms get worse as the instances become topologically less separable (i.e., as we go down from batch 1-5 to 21-25 and from 26-30 to 46-50 in the table).

Third, we compare the performances of the algorithms in the existence of outliers. Outliers may appear in datasets because of measurement errors, coding errors (for example, storing data in terms of millimeters instead of centimeters) or experimental errors etc. Note that the optimal centroid for a given cluster is obtained by taking

Table 4.22: Comparison of tree-kmeans-WVEO, tree-kmeans-WGED and kmeans with respect to external performance measures

Instance	**Algoritmh**	**SmallDataW**				**LargeDataW**			
		BRI	**BARI**	**BFM**	**BMI**	**BRI**	**BARI**	**BFM**	**BMI**
1-5	tree-kmeans-WVEO	0.89	0.70	0.77	0.39	0.89	0.72	0.79	0.39
	tree-kmeans-WGED	0.73	0.39	0.57	0.27	0.73	0.40	0.58	0.27
	k-means	0.73	0.39	0.57	0.28	0.69	0.33	0.54	0.24
6-10	tree-kmeans-WVEO	0.77	0.41	0.56	0.26	0.81	0.52	0.64	0.30
	tree-kmeans-WGED	0.66	0.27	0.49	0.20	0.67	0.30	0.52	0.21
	k-means	0.66	0.26	0.48	0.20	0.63	0.24	0.48	0.19
11-15	tree-kmeans-WVEO	0.80	0.47	0.60	0.30	0.79	0.45	0.59	0.29
	tree-kmeans-WGED	0.60	0.20	0.45	0.17	0.65	0.26	0.49	0.18
	k-means	0.67	0.27	0.48	0.20	0.66	0.25	0.48	0.18
16-20	tree-kmeans-WVEO	0.75	0.35	0.51	0.25	0.75	0.34	0.51	0.24
	tree-kmeans-WGED	0.57	0.15	0.42	0.13	0.57	0.16	0.43	0.13
	k-means	0.66	0.24	0.46	0.17	0.61	0.18	0.44	0.14
21-25	tree-kmeans-WVEO	0.75	0.35	0.51	0.25	0.75	0.35	0.51	0.25
	tree-kmeans-WGED	0.65	0.23	0.46	0.17	0.57	0.14	0.42	0.12
	k-means	0.64	0.22	0.45	0.16	0.59	0.16	0.43	0.13
Average (1-25)	**tree-kmeans-WVEO**	**0.79**	**0.45**	**0.59**	**0.29**	**0.80**	**0.48**	**0.61**	**0.29**
	tree-kmeans-WGED	**0.64**	**0.25**	**0.48**	**0.19**	**0.64**	**0.25**	**0.49**	**0.18**
	k-means	**0.67**	**0.27**	**0.49**	**0.20**	**0.64**	**0.23**	**0.47**	**0.18**
26-30	tree-kmeans-WVEO	0.97	0.92	0.94	0.46	0.91	0.77	0.83	0.41
	tree-kmeans-WGED	0.70	0.37	0.56	0.26	0.69	0.34	0.54	0.24
	k-means	0.74	0.43	0.59	0.29	0.70	0.36	0.55	0.24
31-35	tree-kmeans-WVEO	0.85	0.62	0.71	0.35	0.84	0.58	0.69	0.33
	tree-kmeans-WGED	0.65	0.29	0.51	0.21	0.54	0.16	0.44	0.14
	k-means	0.64	0.26	0.49	0.22	0.59	0.19	0.45	0.16
36-40	tree-kmeans-WVEO	0.79	0.44	0.58	0.29	0.80	0.47	0.61	0.29
	tree-kmeans-WGED	0.55	0.15	0.43	0.13	0.48	0.11	0.42	0.11
	k-means	0.58	0.15	0.42	0.14	0.48	0.11	0.42	0.11
41-45	tree-kmeans-WVEO	0.76	0.38	0.54	0.26	0.74	0.35	0.52	0.23
	tree-kmeans-WGED	0.55	0.15	0.43	0.13	0.39	0.05	0.39	0.06
	k-means	0.52	0.10	0.39	0.11	0.43	0.04	0.38	0.06
46-50	tree-kmeans-WVEO	0.75	0.35	0.51	0.24	0.74	0.34	0.52	0.23
	tree-kmeans-WGED	0.50	0.09	0.39	0.10	0.37	0.03	0.38	0.05
	k-means	0.58	0.15	0.42	0.13	0.44	0.04	0.38	0.07
Average (26-50)	**tree-kmeans-WVEO**	**0.82**	**0.54**	**0.66**	**0.32**	**0.81**	**0.50**	**0.63**	**0.30**
	tree-kmeans-WGED	**0.59**	**0.21**	**0.46**	**0.17**	**0.49**	**0.14**	**0.44**	**0.12**
	k-means	**0.61**	**0.22**	**0.46**	**0.18**	**0.53**	**0.15**	**0.44**	**0.13**

Table 4.23: Outlier case

Instance	**Algoritmh**	**SmallDataW**				**LargeDataW**			
		BRI	**BARI**	**BFM**	**BMI**	**BRI**	**BARI**	**BFM**	**BMI**
1-5	tree-kmeans-WVEO	0.90	0.74	0.80	0.40	0.93	0.82	0.87	0.43
	tree-kmeans-WGED	0.72	0.39	0.57	0.27	0.71	0.38	0.57	0.25
	k-means	0.60	0.25	0.49	0.18	0.59	0.25	0.50	0.18
6-10	tree-kmeans-WVEO	0.80	0.46	0.59	0.29	0.85	0.60	0.70	0.33
	tree-kmeans-WGED	0.61	0.23	0.47	0.18	0.57	0.18	0.45	0.15
	k-means	0.53	0.14	0.43	0.12	0.57	0.19	0.46	0.14
11-15	tree-kmeans-WVEO	0.80	0.47	0.60	0.28	0.80	0.47	0.61	0.29
	tree-kmeans-WGED	0.55	0.14	0.42	0.13	0.58	0.19	0.45	0.15
	k-means	0.53	0.13	0.42	0.12	0.49	0.09	0.41	0.09
16-20	tree-kmeans-WVEO	0.76	0.37	0.53	0.25	0.76	0.38	0.54	0.26
	tree-kmeans-WGED	0.60	0.18	0.44	0.15	0.53	0.14	0.42	0.11
	k-means	0.53	0.12	0.41	0.11	0.50	0.10	0.41	0.09
21-25	tree-kmeans-WVEO	0.75	0.36	0.52	0.25	0.75	0.33	0.50	0.25
	tree-kmeans-WGED	0.59	0.18	0.44	0.14	0.55	0.13	0.41	0.11
	k-means	0.55	0.13	0.42	0.12	0.52	0.12	0.41	0.10
Average (1-25)	**tree-kmeans-WVEO**	**0.80**	**0.48**	**0.61**	**0.29**	**0.82**	**0.52**	**0.64**	**0.31**
	tree-kmeans-WGED	**0.62**	**0.22**	**0.47**	**0.17**	**0.59**	**0.20**	**0.46**	**0.15**
	k-means	**0.55**	**0.16**	**0.43**	**0.13**	**0.53**	**0.15**	**0.44**	**0.12**
26-30	tree-kmeans-WVEO	0.89	0.73	0.80	0.40	0.88	0.70	0.78	0.37
	tree-kmeans-WGED	0.65	0.29	0.51	0.22	0.67	0.31	0.53	0.23
	k-means	0.52	0.16	0.44	0.14	0.56	0.22	0.49	0.16
31-35	tree-kmeans-WVEO	0.83	0.56	0.67	0.33	0.81	0.51	0.63	0.30
	tree-kmeans-WGED	0.65	0.30	0.52	0.21	0.59	0.21	0.47	0.18
	k-means	0.53	0.15	0.43	0.13	0.45	0.08	0.41	0.09
36-40	tree-kmeans-WVEO	0.77	0.42	0.57	0.27	0.78	0.43	0.58	0.27
	tree-kmeans-WGED	0.48	0.09	0.40	0.10	0.37	0.04	0.39	0.06
	k-means	0.48	0.07	0.38	0.09	0.40	0.05	0.40	0.07
41-45	tree-kmeans-WVEO	0.74	0.33	0.51	0.23	0.74	0.35	0.52	0.24
	tree-kmeans-WGED	0.46	0.07	0.39	0.09	0.36	0.03	0.38	0.05
	k-means	0.48	0.09	0.40	0.10	0.41	0.04	0.38	0.06
46-50	tree-kmeans-WVEO	0.75	0.35	0.52	0.24	0.74	0.35	0.52	0.23
	tree-kmeans-WGED	0.44	0.06	0.38	0.08	0.36	0.02	0.38	0.05
	k-means	0.46	0.07	0.38	0.09	0.43	0.04	0.38	0.06
Average (26-50)	**tree-kmeans-WVEO**	**0.80**	**0.48**	**0.61**	**0.29**	**0.79**	**0.47**	**0.61**	**0.28**
	tree-kmeans-WGED	**0.54**	**0.16**	**0.44**	**0.14**	**0.47**	**0.12**	**0.43**	**0.11**
	k-means	**0.49**	**0.11**	**0.41**	**0.11**	**0.45**	**0.09**	**0.41**	**0.09**

median of the weights in tree-kmeans-WGED while it is the mean of the weights in kmeans. As it is known, mean is influenced by outliers while median is more robust. To see the performance of the algorithms in cases where there are outliers, in each problem instance, a tree is selected at random from each of the second and fourth clusters. The edge weights of these trees are multiplied by 10 to turn the trees into outliers. The results of the computational experiments on the instances with outliers are summarized in Table 4.23. When we compare these results with the results in Table 4.22, it is clear that the performance of the kmeans decreases significantly with the existence of outliers while both of our proposed methods are robust to outliers.

4.6.4 Performance on real life datasets

In [14], authors try to find the possible correlations between brain artery structures, and age and sex of patients. For that purpose, they use new representation of brain artery structures, which is called as persistent homology representation, and find that brain structures change with sex and age. There are 109 patients; 98 of them are healthy and 11 of them are with brain tumor. They shared the data of 98 normal patients as supplemental material. For each patient, sex (47 female, 46 male, 2 male to female), age (ranging between 19 and 79) and handedness (89 right handed, 6 left handed, 1 ambidextrous, 2 left/ambidextrous) information are available. Figure 4.4 summarizes the number of patients with different characteristics. We remove the male-to-female, ambidextrous, and left/ambidextrous individuals from the dataset and carry out the computational experiments on the data of 93 individuals.

The main problem here is to obtain branching structure of patients from the available

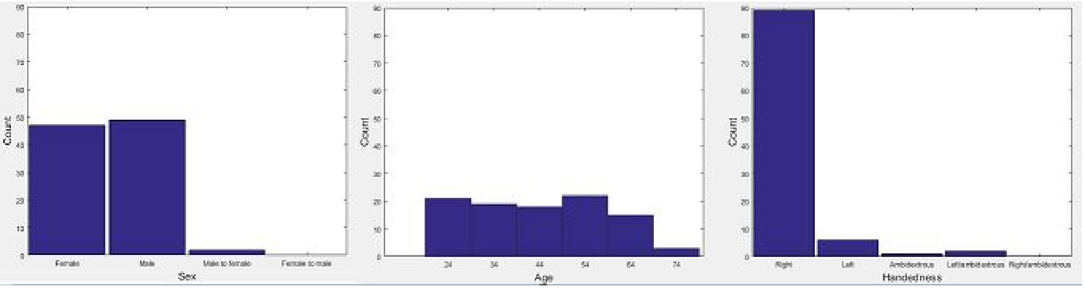


Figure 4.4: Information of patients in brain artery data

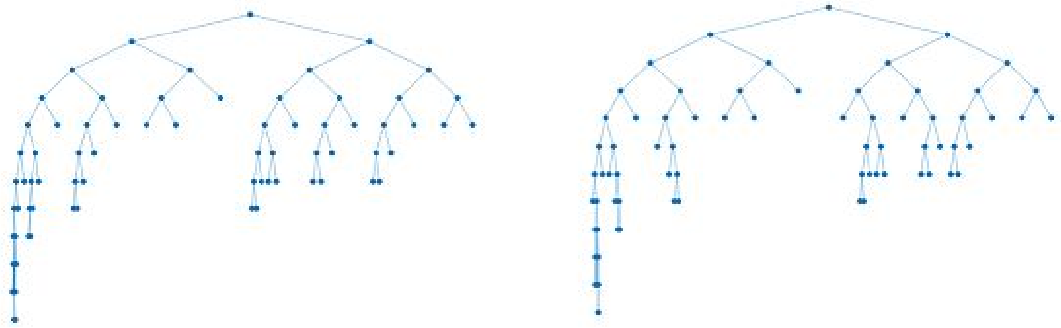


Figure 4.5: Binary trees for patient 2 with child and radius correspondence, respectviely

data. For each patient, the output of tube-tracking algorithm used in [5], which takes MRA images as input, is provided. The output consists of two main parts. In the first part, there is information about approximately 120000 points on brain vessels. For each point, x/y/z coordinates, id of vessel part on which the corresponding point lies, id of parent vessel part, attachment point of current vessel part to the parent vessel part, radius of vessel at the corresponding point etc. are stored. In the second part, there is information about only branching points of the brain vessels. The number of these points are around 50. By using the information provided in the second part, starting from the first split-up point of the vessel entering to the brain as root node, we define a node for each split-up point and an edge between nodes represent the vessel between the corresponding split-up points. As edge weights, we store two information; namely median radius (ranging between 0.5 and 3.9717 mm) of the corresponding vessel and the length (ranging between 1 and 2372 tube tracking ticks) of the vessel. Figure 4.5 shows the binary tree representations of brain vessels of patient 2 which is constructed by using nearly 50 branching points. In the representation, there are 99 nodes and 98 edges. Note that the leaf nodes of the trees define the vessel ends. In the left part of the figure, node correspondence is obtained by locating the nodes with higher number of child to the left as discussed in [5]. The other node correspondence method discussed in [5] is that put the thicker edge to the left. The binary tree obtained with this method is provided in the right part of Figure 4.5. As it can be seen from the figure, we can obtain two different tree structures with different correspondence methods. In total, we may represent each brain artery tree in four ways: using one of the two different topologies (child or radius correspondence), and with or without edge weights. We normalize the weights (length and radius) of each edge of a brain artery tree to a value within 0.5-1 range. The reason behind the selection of this range instead of 0-1 range is to differentiate between non-existent edges, i.e., edges with 0 weight, and edges with too small weights.

Properties of the dataset can be summarized as follows. The average number of levels in brain artery trees is 15.27 while the maximum leveled tree has 25 levels. The average number of edges is 95.65 whereas the largest tree has 188 edges. Moreover, the number of edges in the support tree is 942 and 3738 when we use child correspondence and radius correspondence, respectively.

In [5], the authors find that brain topology and age are correlated. Moreover, the study in [30] shows that age is correlated with total vessel length and average vessel radius. In addition to age effects, some studies such as [130] and [133] report some sex effect. In this study, we cluster the brain artery trees in order to see whether or not the observed age and sex effects pronounce themselves in the resulting clusters. We group the whole data into 4 clusters where the tree topologies, and the length and radius of the vessels (in the weighted case) are the inputs of the algorithms. Note that the age and sex of the individuals are not given as input to the algorithms.

The clustering results are provided in Table 4.24. For each algorithm, we make 10 replications and report the results for the best found solution. The first column of Table 4.24 shows the type of the node correspondence method used. The second column indicates whether the trees are weighted or unweighted. The third column specifies the algorithm used and the remaining columns show the clustering results. For instance, when we obtain the topologies with child correspondence, do not consider any weight, and use tree-kmeans-UWGED for clustering, there are 8 females and 4 males in the first cluster of the best replication. In this cluster, the median ages of females and males are 33.5 and 44.0, respectively. The age effect is visible among females in this solution since the median ages of the females in different clusters (33.5, 37.0, 48.0, and 57.0) are well-separated. In general, the age effect is more pronounced among females than males, which we think is an interesting result. Moreover, when two clusters have similar median ages for a sex, there is usually a larger difference

Table 4.24: Results for clustering whole dataset with different data representations

Correspondence	Weight	Algorithm	Cluster	Median Age		Count	
				Female	Male	Female	Male
Child	None	tree-kmeans-UWVEO	C1	34.0	49.0	8	11
			C2	39.5	37.0	14	7
			C3	41.0	44.0	11	12
			C4	52.5	44.0	14	16
		tree-kmeans-UWGED	C1	33.5	44.0	8	4
			C2	37.0	43.0	15	15
			C3	48.0	49.0	17	16
			C4	57.0	36.0	7	11
		k-modes	C1	36.0	41.5	24	18
			C2	39.0	NaN	1	0
			C3	47.0	43.5	14	14
			C4	52.5	45.5	8	14
	Length & Radius	tree-kmeans-WVEO	C1	33.5	43.5	12	12
			C2	41.0	52.0	13	13
			C3	48.0	37.0	13	9
			C4	57.0	28.5	9	12
		tree-kmeans-WGED	C1	39.5	37.0	18	9
			C2	40.0	50.0	14	13
			C3	44.5	47.5	4	10
			C4	48.0	33.5	11	14
		k-means	C1	36.0	41.0	10	13
			C2	39.0	47.0	14	8
			C3	44.5	42.0	12	10
			C4	48.0	47.0	11	15
Radius	None	tree-kmeans-UWVEO	C1	27.0	46.0	7	12
			C2	42.5	38.5	12	14
			C3	43.0	49.5	15	12
			C4	48.0	46.5	13	8
		tree-kmeans-UWGED	C1	NaN	58.0	0	1
			C2	34.5	45.5	16	20
			C3	44.0	42.0	21	16
			C4	52.5	48.0	10	9
		k-modes	C1	NaN	68.0	0	1
			C2	40.0	39.0	15	16
			C3	43.0	44.0	22	19
			C4	52.5	49.0	10	10
	Length & Radius	tree-kmeans-WVEO	C1	34.5	52.5	16	14
			C2	44.0	36.0	1	5
			C3	44.5	41.0	12	11
			C4	50.5	49.5	18	16
		tree-kmeans-WGED	C1	36.5	56.0	12	9
			C2	37.5	36.0	8	10
			C3	40.0	41.0	14	17
			C4	51.0	47.5	13	10
		k-means	C1	37.0	34.0	2	1
			C2	40.0	48.0	20	20
			C3	46.0	40.5	22	22
			C4	55.0	43.0	3	3

between the median ages of the other sex. This is an indication of the age and sex interaction. For example, in the second and third clusters of the solution obtained with tree-kmeans-UWVEO for the unweighted case in which the topologies are obtained with radius correspondence, the median ages of females are almost equal (42.5 and

Table 4.25: p-values obtained by Kruskal-Wallis test for the equality of age medians

Corres.	Weight	Algorithm	Female	Male
Child	None	tree-kmeans-UWVEO	0.20	0.75
		tree-kmeans-UWGED	0.21	0.74
		k-modes	0.29	0.98
	Length & Radius	tree-kmeans-WVEO	0.10	0.03
		tree-kmeans-WGED	0.85	0.25
		k-means	0.32	0.99
Radius	None	tree-kmeans-UWVEO	0.18	0.64
		tree-kmeans-UWGED	0.06	0.44
		k-modes	0.40	0.45
	Length & Radius	tree-kmeans-WVEO	0.15	0.35
		tree-kmeans-WGED	0.38	0.25
		k-means	0.41	0.67

43.0). However, the median ages of males are 38.5 and 49.5, respectively.

To test the null hypothesis that the median ages of the clusters for a given sex are the same, we apply Kruskall-Wallis test. The alternative hypothesis is that at least one of the clusters' median age for the given sex is different. Table 4.25 reports the p-values. Note that sufficiently large sample sizes are necessary to produce results that are statistically significant at the 5% significance level. In our case, for any partition, the average number of females and males in a cluster are 11.75 and 11.5, respectively. For this reason, we consider p-values ≤ 0.3 as indication of some evidence supporting the alternative hypothesis. In Table 4.25, p-values smaller than 0.1 are dark-shaded indicating a high level of evidence against the null hypothesis while those that are between 0.1 and 0.3 are light-shaded indicating weaker evidence. For example, Kruskal-Wallis test for the equality of median ages of female individuals in different clusters when we use radius correspondence and unweighted edges in tree-kmeans-UWGED algorithm results in a p-value of 0.06 indicating a high level of evidence supporting the alternative hypothesis. The investigation of the table reveals that the age effect is more apparent on females as most of the shaded cells are in the column corresponding to the female individuals. Another observation is that child correspondence seems better than radius correspondence in terms of separating the clusters based on their median ages. Moreover, both tree-kmeans-UWVEO and tree-kmeans-UWGED are better than k-modes and both tree-kmeans-WVEO and

Table 4.26: p-values obtained by Wilcoxon rank sum test for the equality of pairwise age medians

Corres.	Weight	Algorithm	Clusters	Female	Male
Child	None	tree-kmeans-UWVEO	C1/C2	0.41	0.55
			C1/C3	0.89	0.52
			C1/C4	0.06	0.34
			C2/C3	0.43	0.82
			C2/C4	0.31	0.71
			C3/C4	0.11	0.63
		tree-kmeans-UWGED	C1/C2	0.70	0.80
			C1/C3	0.17	0.78
			C1/C4	0.29	0.77
			C2/C3	0.06	0.77
			C2/C4	0.24	0.39
			C3/C4	0.82	0.31
		k-modes	C1/C2	0.94	NaN
			C1/C3	0.48	0.98
			C1/C4	0.07	0.83
			C2/C3	0.80	NaN
			C2/C4	0.44	NaN
			C3/C4	0.21	0.91
	Length & Radius	tree-kmeans-WVEO	C1/C2	0.30	0.48
			C1/C3	0.15	0.72
			C1/C4	0.01	0.11
			C2/C3	0.72	0.09
			C2/C4	0.13	0.00
			C3/C4	0.44	0.12
		tree-kmeans-WGED	C1/C2	0.83	0.48
			C1/C3	0.77	0.64
			C1/C4	0.62	0.27
			C2/C3	0.78	0.98
			C2/C4	0.38	0.06
			C3/C4	0.64	0.20
Radius	None	tree-kmeans-UWVEO	C1/C2	0.05	0.37
			C1/C3	0.06	0.64
			C1/C4	0.10	0.70
			C2/C3	1.00	0.27
			C2/C4	0.72	0.41
			C3/C4	0.61	1.00
		tree-kmeans-UWGED	C1/C2	NaN	0.46
			C1/C3	NaN	0.47
			C1/C4	NaN	0.20
			C2/C3	0.03	0.43
			C2/C4	0.08	0.28
			C3/C4	0.85	0.84
	Length & Radius	tree-kmeans-WVEO	C1/C2	0.71	0.29
			C1/C3	0.09	0.34
			C1/C4	0.03	0.62
			C2/C3	1.00	0.60
			C2/C4	0.95	0.11
			C3/C4	0.88	0.24
		tree-kmeans-WGED	C1/C2	0.94	0.15
			C1/C3	0.38	0.09
			C1/C4	0.21	0.07
			C2/C3	0.58	0.98
			C2/C4	0.15	0.71
			C3/C4	0.31	0.96

tree-kmeans-WGED are better than k-means in terms of separability of the resulting median ages of the clusters.

As a post hoc analysis, we make pairwise comparisons of the median ages of the clusters for a given sex by using Wilcoxon rank sum test whose null hypothesis is that the medians of two clusters are the same. This test is only applied to the partitions having a p-value of ≤ 0.3 as a result of Kruskall-Wallis test for at least one sex. The resulting p-values are displayed in Table 4.26. For example, when we use child correspondence and weighted edges in tree-kmeans-WVEO, the median ages of the females in Cluster 1 and Cluster 4 are different at the 10% significance level since the corresponding p-value is 0.01. The median ages of males in those clusters are also different to a certain degree with a corresponding p-value of 0.11. In our view, the p-values displayed in Table 4.26 provide supporting evidence of the finding that brain artery structures are correlated with age and sex.

4.7 Conclusion and Future Work

In this chapter, we consider a non-traditional clustering problem in which data objects are m-ary trees. There are three assumptions of the problem: node correspondence is known, nodes are unweighted and edges can be both unweighted or weighted. We propose kmeans based algorithms for the solution of the problem; namely tree-kmeans-UWVEO, tree-kmeans-UWGED, tree-kmeans-WVEO and tree-kmeans-WGED. In the initialization steps of the algorithms, we have a procedure, which preserves the properties of the data, to obtain initial centroid trees. In the assignment steps, we have utilized unweighted VEO, unweighted GED, weighted VEO and weighted GED measures, respectively. To find the centroid tree of the given cluster of trees in the assignment steps, for each measure, we first mathematically formulated the problem and then proposed heuristic approach. Except the heuristic for the weighted VEO, all approaches are proven to find the optimal centroid of the given cluster. We tested our algorithms with randomly generated datasets and compared the results with well known traditional algorithms: kmodes and kmeans. The results showed the efficiency of proposed methods. Moreover, we applied our algorithms to a real-life dataset and observed that the relationship between the brain artery structures versus the age and sex can be discovered by means of clustering.

As a future work, one may consider to include preliminary knowledge (such as instance level constraints used in Chapter 2) to the clustering process to improve the clustering performance. To the extend we see, the performances of tree clustering algorithms are not well. For example, in [107], clustering accuracy is around 0.7 even though there exists visually separable clusters. By providing a few instance-level constraints or a few data objects with label, one can improve the clustering performances. Thus, in order to be able to handle preliminary knowledge, our algorithms can be modified or completely new algorithms may be developed .

Another extension can be the relaxation of the known node correspondence assumption. In that case, appropriate distance/similarity measures and heuristics for the centroid finding problems with these measures should be defined. But it should be noted that when the node correspondence is unknown, the distance/similarity measures are computationally complex.

Moreover, weights associated with vertices can be considered in addition to the edge weights.

The study in this chapter has been submitted to Annals of Operations Research for publication.

CHAPTER 5

CONCLUSION

Improving technology and measurement capabilities, and the need for deeper analyses result in collecting more and more data which makes data mining, the science of extracting new and useful information from data, more important day by day. We address three data mining problems in this thesis.

In the first problem, we consider semi-supervised clustering of regional data objects where the aim is to minimize the sum of the violation costs of the unsatisfied instance level must-link and cannot-link constraints plus the weighted sum of squared maximum Euclidean distances between the data objects and the centroids of the clusters they are assigned to. Semi-supervised learning is at the intersection of supervised and unsupervised learning. It has attracted the attention of the researchers in last decades since it improves the performance of unsupervised learning when there is a small amount of information which is not enough to use supervised learning [144]. Regional data objects mainly arise because the uncertainty in the location of the data objects in feature space can be represented by regions. These uncertainities may occur becasue of several reasons such as imprecision in measurements, sampling error, reporting errors, and so on. Also, when we have interval valued datasets, each data object is defined as a region, i.e., a hyper-rectangle, see e.g., [18].

In this thesis it was assumed that the data objects were closed convex polytopes and/or closed disks in $\mathbf{R}^n$. We proposed two solution approaches for computing the centroid of a given cluster consisting of regional data objects, namely a mathematical programming formulation (i.e., SOCP formulation) and an adaptation of the subgradient method. These two solution methods, whose detailed time complexity analysis were also provided, were computationally compared on several instances and the subgradient method turned out to be very efficient in terms of solution time. We then considered seven solution approaches for finding a partition of regional data objects into a given number of clusters which is expected to be in accordance with a given set of instance-level constraints. These approaches are an MISOCP formulation, UCOP-k-means, UPC-k-means, UCVQE, USeeded-k-means, UConstrained-k-means, and UAHCP. The first one is a mathematical programming formulation, the next five are extensions of k-means based algorithms proposed for semi-supervised clustering with point data objects, and the last one is an extension of an agglomerative hierarchical clustering algorithm. The proposed subgradient method is utilized in these algorithms for centroid computation. Solution approaches were compared using three different initialization procedures (for k-means based algorithms), four different instance level constraint generation techniques, different numbers of instance level constraints, and six different performance measures on an artificial two-dimensional dataset and four real-life higher dimensional datasets. The MISOCP formulation turned out to be applicable only for small size instances. For larger size instances, UConstrained-kmeans and UAHCP performed better than the other algorithms, in most of the cases. UConstrained-k-means turned out to be the best solution method when we take the solution time into account. The study is published as an article in Neurocomputing, see [52]. Also, a book chapter related with the literature review on this problem can be found in [51].

As a future work for this first problem, one can consider regional centroids instead of point ones. In that case, after introducing new distance measures to find the distance between two regions, our algorithms can be modified accordingly. Also, fuzzy version of the problem, where the assumption of assigning each regional data object to a single cluster is relaxed, can be studied. Moreover, in this study, we compared our algorithms in terms of several performance measures. Based on their effectiveness on semi-supervised clustering, multi-objective solution approaches optimizing several performance measures simultaneously can be considered as another future research direction.

The second problem considered in this thesis is the problem of finding a group of central nodes in a graph. In this problem, we aim to mine a single graph based on the edge behaviour. The identification of central nodes in a graph may arise in several domains such as social networks, computer networks, vehicular networks, and so on. In this study, we utilized the group betweenness centrality (GBC) measure which assumes that information flows through shortest paths. The GBC of a group of nodes measures the influence the group has on communication between every pair of nodes in the network under the shortest paths assumption. This assumption is valid in several applications. For example, assume that an investor wants to locate a few roadhouses. To increase his profit, he needs to increase the number of people who are driving by the roadhouses since each one is a potential customer. As the drivers have a tendecy to use the shorthest paths between their origins and destinations, the investor's problem is a problem of finding the group of nodes with the highest GBC on the road network.

Given a group size, the problem of finding the group of vertices with the highest GBC is a combinatorial problem. We proposed a method that first computes upper and lower bounds on the GBC of several groups after a preprocessing step taking time proportional to the cube of the number of vertices in the network. The computation of the bounds on the GBC for each group requires a running time proportional to the square of the group size. After computation of bounds, we eliminate groups with upper bounds that are lower than the maximum lower bound obtained to find candidates for the optimal group. Our method brings an improvement over the method in [92] and uses the method in [125] as preprocessing step. The method uses a given number of subsets of vertices as input and returns a list of candidate groups (for the group with the highest GBC value) and a group with the highest lower bound together with the optimality gap. The method we improve upon has to be restarted for each group, which makes the method less efficient for the computation of the GBC of several groups. In addition, the bounds used in our method are stronger and/or faster to compute in general.

We also utilized an algorithm from the literature (which finds the exact GBC value for a given group) in order to compare the output of the proposed algorithm and the optimal group. We conducted computational experiments with several randomly generated graphs, one small and three large scale real life networks. Our computational experiments show that in the search for a group of a certain size with the highest GBC value, our method reduces the number of candidate groups substantially and in some cases gives the optimal group without exactly computing the GBC values which is computationally more demanding. Moreover, the group returned by the proposed algorithm (the one with the highest lower bound) is the same with the optimal group (if it is available) for almost all instances and the exact GBC values are closer to the lower bounds. The study is published in Networks, see [53].

As a future work for the second problem, some elegant methods can be developed to decrease the number of groups given as input to the algorithm since enumeration of all possible groups and the computation of the bounds for each one may not be practical in large networks. For example, in the search for the optimal group of size k, after the computation of the bounds on the GBC of a group, say T, of size $< k$, it may be possible to bound the GBC of any group S of size k containing T. If the upper bound is small enough it may be possible to eliminate all groups of size k containing the set T from further consideration. Moreover, in the search for the optimal group of size k, if k is less than or equal to the number of non-leaf vertices, there exists an optimal group that does not contain any leaf vertex. Because if a leaf vertex v and the vertex u it is adjacent to are both in a group, then discarding v from the group does not reduce the GBC value of the group. Similarly, if the vertex v belongs to a group, but the vertex u does not, then one can discard v from the group, add u to the group and this operation does not decrease the GBC value of the group. Such methods to reduce the number of groups that will be given as input to the proposed algorithm can be developed. Another future research direction is to find the group with the highest lower bound. Although it is NP-hard to find the group with the highest GBC, it is not currently known whether the problem of finding the group with the highest YAT is NP-hard or not.

The third problem considered in this thesis is the clustering of tree-structured data objects. Such data objects are results of improving technology/measurement capabilities and the need for deeper analyses which lead to collecting more complex datasets. Tree-structured data objects appear in many applications such as brain artery analysis, retinal vessel clustering, protein sequence classification, and so on. The clustering literature is mostly devoted to point data objects. Clustering tree-structured data objects is a challenging problem which is not considered much in the literature. Thus, in this study, we assumed that data objects are m-ary rooted trees where node correspondence is known, nodes are unweighted, and edges can be either unweighted or weighted.

For the solution of the problem, we proposed a k-means based algorithm which starts with initial centroid trees and repeats tree-to-cluster assignment and centroid update steps until convergence. In the assignment step, to measure the distance between two trees, we utilized vertex/edge overlap and graph edit distance. In the update step, to find a centroid tree for a given set of trees, we provided mathematical programming formulations. When edges are unweighted, we proposed a Nonlinear Integer Programming formulation and an Integer Linear Programming formulation for vertex/edge overlap and graph edit distance, respectively. We solved these formulations to optimality. When edges are weighted, we proposed a Nonlinear Programming formulation and a Mixed Integer Nonlinear Programming formulation for vertex/edge overlap and graph edit distance, respectively. For vertex/edge overlap, we developed a heuristic not guaranteeing optimality. For graph edit distance, we solved the formulation to optimality. We experimented with randomly generated datasets to select parameters of the algorithms. After setting the parameters, we compared our approaches with k-modes (for unweighted edges) and k-means (for weighted edges) on randomly generated datasets. Moreover, we conducted computational experiments with a real life brain artery dataset and found out the relationship between the brain artery structures versus the age and sex of the individuals. Our approaches performed better than k-modes and k-means in terms of discovering this relationship. The study is submitted to Annals of Operations Research for publication.

As a future work, we may introduce some preliminary knowledge in the form of labeled data or instance-level constraints (as in the first problem) to the clustering process. As tree-structured data clustering is a newly emerging issue, the performances of the algorithms in the literature are not very good and they may be improved by semi-supervised learning as for their point and regional data object counterparts. To be able to handle preliminary knowledge, our algorithms can be modified or completely new algorithms may be developed. Another extension can be the relaxation of the assumptions of the study, namely known node correspondence and unweighted nodes. In this case, after defining appropriate distance/similarity measures, one may try to modify our algorithms or define new solution approaches for the problem. However, for the unknown node correspondence case, it should be noted that the distance/similarity measures are computationally complex. Finally, one may work on adapting the proposed methods to clustering of general graphs.

REFERENCES

[1] C. C. Aggarwal, N. Ta, J. Wang, J. Feng, and M. Zaki. Xproj: a framework for projected structural clustering of xml documents. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 46–55. ACM, 2007.

[2] C. C. Aggarwal and H. Wang. A survey of clustering algorithms for graph data. In *Managing and mining graph data*, pages 275–301. Springer, 2010.

[3] F. Alizadeh and D. Goldfarb. Second-order cone programming. *Mathematical programming*, 95(1):3–51, 2003.

[4] A. Aly and A. Marucheck. Generalized weber problem with rectangular regions. *The Journal of the Operational Research Society*, 13:983–989, 1982.

[5] B. Aydin, G. Pataki, H. Wang, E. Bullitt, and J. Marron. A principal component analysis for trees. *The Annals of Applied Statistics*, pages 1597–1615, 2009.

[6] B. Aydın, G. Pataki, H. Wang, A. Ladha, E. Bullitt, and J. Marron. New approaches to principal component analysis for trees. *Statistics in Biosciences*, 4(1):132–156, 2012.

[7] A. Banerjee and J. Ghosh. Scalable clustering algorithms with balancing constraints. *Data Mining and Knowledge Discovery*, 13(3):365–395, 2006.

[8] A. Bar-Hillel, T. Hertz, N. Shental, and D. Weinshall. Learning distance functions using equivalence relations. In *Proceedings of 20th International Conference on Machine Learning*, volume 3, pages 11–18, 2003.

[9] S. Basu. *Semi-supervised Clustering: Probabilistic Models, Algorithms and Experiments*. PhD thesis, Austin, TX, USA, 2005. AAI3187658.

[10] S. Basu, A. Banerjee, and R. Mooney. Semi-supervised clustering by seeding. In *Proceedings of 19th International Conference on Machine Learning*, pages 19–26. Citeseer, 2002.

[11] S. Basu, A. Banerjee, and R. J. Mooney. Active semi-supervision for pairwise constrained clustering. In *Proceedings of the SIAM International Conference on Data Mining*, volume 4, pages 333–344. SIAM, 2004.

[12] S. Basu, M. Bilenko, and R. J. Mooney. Comparing and unifying search-based and similarity-based approaches to semi-supervised clustering. In *Proceedings of the ICML-2003 workshop on the continuum from labeled to unlabeled data in machine learning and data mining systems*, pages 42–49. Citeseer, 2003.

[13] S. Basu, M. Bilenko, and R. J. Mooney. A probabilistic framework for semi-supervised clustering. In *Proceedings of 10th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 59–68. ACM, 2004.

[14] P. Bendich, J. S. Marron, E. Miller, A. Pieloch, and S. Skwerer. Persistent homology analysis of brain artery trees. *The annals of applied statistics*, 10(1):198, 2016.

[15] C. Bennet and A. Mirakhor. Optimal facility location with respect to several regions. *Journal of Regional Science*, 14:131–136, 1974.

[16] L. Biao, Z. Kejun, F. Huamin, and L. Yang. A new approach of clustering malicious javascript. In *Software Engineering and Service Science (ICSESS), 2014 5th IEEE International Conference on*, pages 157–160. IEEE, 2014.

[17] M. Bilenko, S. Basu, and R. J. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *Proceedings of 21st international conference on Machine learning*, pages 11–18. ACM, 2004.

[18] L. Billard and E. Diday. *Principal Component Analysis*, page 166. John Wiley & Sons Inc., Hoboken, NJ, 2007.

[19] P. Bille. A survey on tree edit distance and related problems. *Theoretical computer science*, 337(1-3):217–239, 2005.

[20] H.-H. Bock. *Analysis of symbolic data: Exploratory methods for extracting statistical information from complex data*. Studies in Classification, Data Analysis, and Knowledge Organization., 2000.

[21] S. Borgwardt, A. Brieden, and P. Gritzmann. Geometric clustering for the consolidation of farmland and woodland. *The Mathematical Intelligencer*, 36(2):37–44, 2014.

[22] E. Boros, A. Scozzari, F. Tardella, and P. Veneziani. Polynomially computable bounds for the probability of the union of events. *Mathematics of Operations Research*, 39(4):1311–1329, 2014.

[23] S. Boyd and A. Mutapcic. Subgradient methods. *Lecture notes of EE364b, Stanford University, Winter Quarter*, 2006, 2007.

[24] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, USA, 2004.

[25] P. S. Bradley, K. P. Bennett, and A. Demiriz. Constrained k-means clustering. Technical report, Microsoft Corperation, 2000.

[26] U. Brandes. A faster algorithm for betweenness centrality. *The Journal of Mathematical Sociology*, 25(2):163–177, 2001.

[27] U. Brandes. On variants of shortest-path betweenness centrality and their generic computation. *Social Networks*, 30(2):136 – 145, 2008.

[28] J. Brimberg and G. Wesolowsky. Locating facilites by minimax relative to closest points of demand areas. *Computers and Operations Research*, 29:625–636, 2002.

[29] J. Brimberg and G. Wesolowsky. Minisum location with closest euclidean distances. *Annals of Operations Research*, 11:151–165, 2002.

[30] E. Bullitt, D. Zeng, B. Mortamet, A. Ghosh, S. R. Aylward, W. Lin, B. L. Marks, and K. Smith. The effects of healthy aging on intracerebral blood vessels visualized by magnetic resonance angiography. *Neurobiology of aging*, 31(2):290–300, 2010.

[31] H. Calik, M. Labbé, and H. Yaman. *Location Science*, chapter p-Center Problems, pages 79–92. Springer International Publishing, Cham, 2015.

[32] G. S. Canright and K. Engø-Monsen. 3.3 - some relevant aspects of network analysis and graph theory. In J. Bergstra and MarkBurgess, editors, *Handbook*

of Network and System Administration, pages 361 – 424. Elsevier, Amsterdam, 2008.

[33] E. Carrizosa, E. Conde, M. Munoz-Marquez, and J. Puerto. The generalized weber problem with expected distances. *RAIRO Operations Research*, 29(1):35–57, 1995.

[34] M. E. Celebi, H. A. Kingravi, and P. A. Vela. A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Systems with Applications*, 40(1):200 – 210, 2013.

[35] H. Chang and D.-Y. Yeung. Locally linear metric adaptation for semi-supervised clustering. In *Proceedings of 21st international conference on Machine learning*, pages 153–160. ACM, 2004.

[36] M. Chau, R. Cheng, B. Kao, and J. Ng. *Advances in Knowledge Discovery and Data Mining: 10th Pacific-Asia Conference, PAKDD 2006, Singapore, April 9-12, 2006. Proceedings*, chapter Uncertain Data Mining: An Example in Clustering Location Data, pages 199–204. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[37] S. S. Chawathe. Comparing hierarchical data in external memory. In *VLDB*, volume 99, pages 90–101, 1999.

[38] C.-A. Chou, W. A. Chaovalitwongse, T. Y. Berger-Wolf, B. DasGupta, and M. V. Ashley. Capacitated clustering problem in computational biology. *Computers and Operations Research*, 39(3):609–619, 2012.

[39] K. L. Chung and P. Erdos. On the application of the borel-cantelli lemma. *Transactions of the American Mathematical Society*, 72(1):179–186, 1952.

[40] L. Cooper. A random locational equilibrium problem. *Journal of Regional Sciences*, 14:47–54, 1974.

[41] I. CVX Research. CVX:Matlab software for disciplined convex programming, version 2.0., Apr. 2011.

[42] I. Davidson and S. Basu. A survey of clustering with instance level constraints. *ACM Transactions on Knowledge Discovery from Data*, 1:1–41, 2007.

[43] I. Davidson and S. Ravi. Agglomerative hierarchical clustering with constraints: Theoretical and empirical results. In A. M. Jorge, L. Torgo, P. Brazdil, R. Camacho, and J. Gama, editors, *Knowledge Discovery in Databases: PKDD 2005*, pages 59–70. Springer-Verlag, Berlin Heidelberg, 2005.

[44] I. Davidson and S. Ravi. Clustering with constraints: Feasibility issues and the k-means algorithm. In *Proceedings of 2005 SIAM International Conference on Data Mining*, pages 138–149. SIAM, 2005.

[45] D. Dawson and D. Sankoff. An inequality for probabilities. *Proceedings of the American Mathematical Society*, 18(3):504–507, 1967.

[46] D. De Caen. A lower bound on the probability of a union. *Discrete mathematics*, 169(1-3):217–220, 1997.

[47] A. Demiriz, K. P. Bennett, and M. J. Embrechts. Semi-supervised clustering using genetic algorithms. *Artificial neural networks in engineering*, pages 809–814, 1999.

[48] L. Dicken and J. Levine. Applying clustering techniques to reduce complexity in automated planning domains. In *Proceedings of 11th International Conference on Intelligent Data Engineering and Automated Learning – IDEAL 2010*, volume 6283, pages 186–193, 2010.

[49] P. Dickinson and M. Kraetzl. Novel approaches in modelling dynamics of networked surveillance environment. In *Proc. of the 6th Intl. Conf. of Information Fusion*, volume 1, pages 302–309, 2003.

[50] D. Dinler, M. Tural, and C. Iyigun. Location problems with demand regions. In B. Kara, I. Sabuncuoglu, and B. Bidanda, editors, *Global Logistic Management*. CRC Press, 2014.

[51] D. Dinler and M. K. Tural. *A Survey of Constrained Clustering*, pages 207–235. Springer International Publishing, Cham, 2016.

[52] D. Dinler and M. K. Tural. Robust semi-supervised clustering with polyhedral and circular uncertainty. *Neurocomputing*, 265:4–27, 2017.

[53] D. Dinler and M. K. Tural. Faster computation of successive bounds on the group betweenness centrality. *Networks*, 71(4):358–380, 2018.

[54] D. Dinler, M. K. Tural, and C. Iyigun. Heuristics for a continuous multi-facility location problem with demand regions. *Computers and Operations Research*, 62:237–256, 2015.

[55] K. Dohmen and P. Tittmann. Bonferroni-type inequalities and binomially bounded functions. *Discrete Mathematics*, 310(6):1265–1268, 2010.

[56] S. Dolev, Y. Elovici, R. Puzis, and P. Zilberman. Incremental deployment of network monitors based on group betweenness centrality. *Information Processing Letters*, 109(20):1172 – 1176, 2009.

[57] Z. Drezner. The p-centre problem-heuristic and optimal algorithms. *Journal of the Operational Research Society*, pages 741–748, 1984.

[58] Z. Drezner and G. Wesolowsky. Location models with groups of demand points. *INFOR*, 38:359–372, 2000.

[59] R. O. Duda and P. E. Hart. *Pattern classification and scene analysis*, volume 3. Wiley, New York, 1973.

[60] M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 95, pages 14863–14868. National Academy Sciences, 1998.

[61] A. Erdem and S. Tari. A similarity-based approach for shape classification using aslan skeletons. *Pattern Recognition Letters*, 31(13):2024–2032, 2010.

[62] M. G. Everett and S. P. Borgatti. The centrality of groups and classes. *The Journal of Mathematical Sociology*, 23(3):181–201, 1999.

[63] M. Fink and J. Spoerhase. Maximum betweenness centrality: Approximability and tractable cases. In *WALCOM*, pages 9–20. Springer, 2011.

[64] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[65] A. Flesia. Unsupervised classification of tree structured objects. In *BIOMAT 2008*, pages 280–299. 2009.

[66] S. Gallot. A bound for the maximum of a number of random variables. *Journal of Applied Probability*, 3(2):556–558, 1966.

[67] N. Ganganath, C.-T. Cheng, and K. T. Chi. Data clustering with cluster size constraints using a modified k-means algorithm. In *Proceedings of 2014 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pages 158–161. IEEE, 2014.

[68] S. Geetha, G. Poonthalir, and P. Vanathi. Improved k-means algorithm for capacitated clustering problem. *INFOCOMP Journal of Computer Science*, 8(4):52–59, 2009.

[69] S. Ghiasi, A. Srivastava, X. Yang, and M. Sarrafzadeh. Optimal energy aware clustering in sensor networks. *Sensors*, 2(7):258–269, 2002.

[70] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[71] T. Gowda and C. A. Mattmann. Clustering web pages based on structure and style similarity (application paper). In *2016 IEEE 17th International Conference on Information Reuse and Integration (IRI)*, pages 175–180, July 2016.

[72] F. Gullo, G. Ponti, and A. Tagarelli. *Scalable Uncertainty Management: Second International Conference, SUM 2008, Naples, Italy, October 1-3, 2008. Proceedings*, chapter Clustering Uncertain Data Via K-Medoids, pages 229–242. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[73] I. Gurobi Optimization. Gurobi optimizer reference manual, 2013.

[74] P. Hansen and N. Mladenovic. J-means: a new local search heuristic for minimum sum of squares clustering. *Pattern recognition*, 34(2):405–413, 2001.

[75] H. Heumann and G. Wittum. The tree-edit-distance, a measure for quantifying neuronal morphology. *Neuroinformatics*, 7(3):179–190, 2009.

[76] F. M. Hoppe. Improving probability bounds by optimization over subsets. *Discrete Mathematics*, 306(5):526–530, 2006.

[77] R. Howard. Classifying a population into homogeneous groups. In J. R. Lawrence, editor, *Operational Research in the Social Sciences*. Tavistock Publication, London, 1966.

[78] J. Hu, M. Singh, and A. Mojsilovic. Categorization using semi-supervised clustering. In *19th International Conference on Pattern Recognition, ICPR 2008*, pages 1–4. IEEE, 2008.

[79] Y. Huang and T. M. Mitchell. Text clustering with extended user feedback. In *Proceedings of 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 413–420. ACM, 2006.

[80] Z. Huang. A fast clustering algorithm to cluster very large categorical data sets in data mining. *DMKD*, 3(8):34–39, 1997.

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

[82] D. Hunter. An upper bound for the probability of a union. *Journal of Applied Probability*, pages 597–603, 1976.

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

[84] B. Jiang, J. Pei, Y. Tao, and X. Lin. Clustering uncertain data based on probability distribution similarity. *Knowledge and Data Engineering, IEEE Transactions on*, 25(4):751–763, 2013.

[85] J. Jiang and Y. Xu. Minisum location problem with farthest euclidean distances. *Mathematical Methods of Operations Research*, 64:285–308, 2006.

[86] J. Jiang and X. Yuan. A barzilai-borwein-based heuristic algorithm for locating multiple facilities with regional demand. *Computational Optimization and Applications*, 51:1275–1295, 2012.

[87] S. D. Kamvar, D. Klein, and C. D. Manning. Spectral learning. In *Proceedings of 18th International Joint Conference of Artificial Intelligence*, pages 561–566. Stanford InfoLab, 2003.

[88] L. Kaufman and P. J. Rousseeuw. *Partitioning Around Medoids (Program PAM)*, pages 68–125. John Wiley and Sons, Inc., 2008.

[89] A. Kchiche and F. Kamoun. Access-points deployment for vehicular networks based on group centrality. In *2009 3rd International Conference on New Technologies, Mobility and Security*, pages 1–6. IEEE, 2009.

[90] V. Khakhutskyy, M. Schwarzfischer, N. Hubig, C. Plant, C. Marr, M. A. Rieger, T. Schroeder, and F. J. Theis. Centroid clustering of cellular lineage trees. In *International Conference on Information Technology in Bio-and Medical Informatics*, pages 15–29. Springer, 2014.

[91] D. Klein, S. D. Kamvar, and C. D. Manning. From instance-level constraints to space-level constraints: Making the most of prior knowledge in data clustering. In *Proceedings of 19th International Conference on Machine Learning*, pages 307–314. Stanford, 2002.

[92] E. D. Kolaczyk, D. B. Chua, and M. Barthélemy. Group betweenness and co-betweenness: Inter-related notions of coalition centrality. *Social Networks*, 31(3):190 – 203, 2009.

[93] E. G. Kounias. Bounds for the probability of a union, with applications. *The Annals of Mathematical Statistics*, 39(6):2154–2158, 1968.

[94] S. Kounias and J. Marin. Best linear bonferroni bounds. *SIAM Journal on Applied Mathematics*, 30(2):307–323, 1976.

[95] D. Koutra, J. T. Vogelstein, and C. Faloutsos. Deltacon: A principled massive-graph similarity function. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, pages 162–170. SIAM, 2013.

[96] H. Kuai, F. Alajaji, and G. Takahara. A lower bound on the probability of a finite union of events. *Discrete Mathematics*, 215(1):147–158, 2000.

[97] S. M. Kwerel. Bounds on the probability of the union and intersection of m events. *Advances in Applied Probability*, 7(2):431–448, 1975.

[98] M. H. C. Law, A. Topchy, and A. K. Jain. Clustering with soft and group constraints. In A. Fred, T. M. Caeli, R. P. W. Duin, A. C. Campilho, and

D. Ridder, editors, *Structural, Syntactic, and Statistical Pattern Recognition*, pages 662–670. Springer-Verlag, Berlin Heidelberg, 2004.

[99] M. H. C. Law, A. P. Topchy, and A. K. Jain. Model based clustering with probabilistic constraints. In *Proceedings of 2005 SIAM International Conference on Data Mining*, pages 641–645. Citeseer, 2005.

[100] S. D. Lee, B. Kao, and R. Cheng. Reducing uk-means to k-means. In *Proceedings of the Seventh IEEE International Conference on Data Mining Workshops*, ICDMW '07, pages 483–488, Washington, DC, USA, 2007. IEEE Computer Society.

[101] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[102] F. A. Lessa, T. Raiol, M. M. Brigido, D. S. Martins Neto, M. E. M. Walter, and P. F. Stadler. Clustering rfam 10.1: Clans, families, and classes. *Genes*, 3(3):378–390, 2012.

[103] M. Lichman. UCI machine learning repository, 2013.

[104] A. Likas, N. Vlassis, and J. J. Verbeek. The global k-means clustering algorithm. *Pattern recognition*, 36(2):451–461, 2003.

[105] M. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. *Linear Algebra and its Applications*, 284:193–228, 1998.

[106] R. Love. A computational procedure for optimally locating a facility with respect to several rectangular regions. *Journal of Regional Sciences*, 12:233–242, 1972.

[107] N. Lu and H. Miao. Clustering tree-structured data on manifold. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):1956–1968, 2016.

[108] N. Lu and Y. Wu. Clustering of tree-structured data. In *Information and Automation, 2015 IEEE International Conference on*, pages 1210–1215. IEEE, 2015.

[109] P. Lyman and H. R. Varian. How much information 2000?, 2000.

[110] P. Lyman and H. R. Varian. How much information 2003?, 2003.

[111] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[112] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, England, 2008.

[113] J. S. Marron and A. M. Alonso. Overview of object oriented data analysis. *Biometrical Journal*, 56(5):732–753, 2014.

[114] I. McCulloh, H. Armstrong, and A. N. Johnson. *Social network analysis with applications*. Hoboken, New Jersey John Wiley & Sons, Inc, 2013.

[115] N. Megiddo and K. J. Supowit. On the complexity of some common geometric location problems. *SIAM journal on computing*, 13(1):182–196, 1984.

[116] S. Miyamoto and A. Terami. Constrained agglomerative hierarchical clustering algorithms with penalties. In *Proceedings of 2011 IEEE International Conference on Fuzzy Systems*, pages 422–427. IEEE, 2011.

[117] J. M. Mulvey and M. P. Beck. Solving capacitated clustering problems. *European Journal of Operational Research*, 18(3):339–348, 1984.

[118] F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354–359, 1983.

[119] W. K. Ngai, B. Kao, C. K. Chui, R. Cheng, M. Chau, and K. Y. Yip. Efficient clustering of uncertain data. In *Proceedings of the Sixth International Conference on Data Mining*, ICDM '06, pages 436–445, Washington, DC, USA, 2006. IEEE Computer Society.

[120] C. Ni, C. R. Sugimoto, and J. Jiang. Degree, closeness, and betweenness: Application of group centrality measurements to explore macro-disciplinary evolution diachronically. In *Proceedings of ISSI*, pages 1–13, 2011.

[121] J. Ouyang and I. K. Sethi. A novel distance measure for interval data. In *PRIS*, pages 49–58, 2007.

[122] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina. Web graph similarity for anomaly detection. *Journal of Internet Services and Applications*, 1(1):19–30, 2010.

[123] K. Pearson. On lines and planes of closest fit to system of points in space. philiosophical magazine, 2, 559-572, 1901.

[124] B. T. Polyak. Minimization of unsmooth functionals. *USSR Computational Mathematics and Mathematical Physics*, 9(3):14–29, 1969.

[125] R. Puzis, Y. Elovici, and S. Dolev. Fast algorithm for successive computation of group betweenness centrality. *Physical Review E*, 76:056709, Nov 2007.

[126] R. Puzis, Y. Elovici, and S. Dolev. Finding the most prominent group in complex networks. *AI Communications*, 20(4):287–296, 2007.

[127] S. Ranshous, S. Shen, D. Koutra, S. Harenberg, C. Faloutsos, and N. F. Samatova. Anomaly detection in dynamic networks: a survey. *Wiley Interdisciplinary Reviews: Computational Statistics*, 7(3):223–247, 2015.

[128] A. Sanfeliu and K.-S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE transactions on systems, man, and cybernetics*, (3):353–362, 1983.

[129] E. Segal, M. Shapira, A. Regev, D. Pe'er, D. Botstein, D. Koller, and N. Friedman. Module networks: identifying regulatory modules and their condition-specific regulators from gene expression data. *Nature genetics*, 34(2):166–176, 2003.

[130] D. Shen, H. Shen, S. Bhamidi, Y. Muñoz Maldonado, Y. Kim, and J. S. Marron. Functional data analysis of tree data objects. *Journal of Computational and Graphical Statistics*, 23(2):418–438, 2014.

[131] K. Shin and A. Abraham. Two phase semi-supervised clustering using background knowledge. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 707–712. Springer, 2006.

[132] N. Z. Shor. Application of the gradient method for the solution of network transportation problems. In *Scientific Seminar on Theory and Application of Cybernetics and Operations Research*, volume 1, pages 9–17. Academy of Sciences, 1962.

[133] S. Skwerer, E. Bullitt, S. Huckemann, E. Miller, I. Oguz, M. Owen, V. Patrangenaru, S. Provan, and J. Marron. Tree-oriented analysis of brain artery structure. *Journal of Mathematical Imaging and Vision*, 50(1-2):126–143, 2014.

[134] A. Strehl, J. Ghosh, and R. Mooney. Impact of similarity measures on web-page clustering. In *Workshop on Artificial Intelligence for Web Search (AAAI 2000)*, pages 58–64, 2000.

[135] Y. Takenaka and T. Wakao. Similarity measure among structures of local government statute books based on tree edit distance. In *Knowledge and Systems Engineering (KSE), 2015 Seventh International Conference on*, pages 49–54. IEEE, 2015.

[136] H. S. Thota, V. V. Saradhi, and T. Venkatesh. Network traffic analysis using principal component graphs. 2013.

[137] A. Torsello, D. Hidovic-Rowe, and M. Pelillo. Polynomial-time metrics for attributed trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1087–1099, 2005.

[138] A. Torsello, A. Robles-Kelly, and E. R. Hancock. Discovering shape classes using tree edit-distance and pairwise clustering. *International Journal of Computer Vision*, 72(3):259–285, 2007.

[139] H. H. Tsang and K. C. Wiese. Sarna-ensemble-predict: the effect of different dissimilarity metrics on a novel ensemble-based rna secondary structure prediction algorithm. In *Computational Intelligence in Bioinformatics and Computational Biology, 2009. CIBCB'09. IEEE Symposium on*, pages 8–15. IEEE, 2009.

[140] A. K. H. Tung, J. Han, L. V. S. Lakshmanan, and R. T. Ng. Constraint-based

clustering in large databases. In J. V. Bussche and V. Vianu, editors, *Database Theory*, pages 405–419. Springer-Verlag, Berlin Heidelberg, 2001.

[141] L. Vandenberghe. Subgradient method. *Lecture notes of EE236c, University of California, Los Angeles, Spring Quarter*, 2016.

[142] A. Veremyev, O. A. Prokopyev, and E. L. Pasiliao. Finding groups with maximum betweenness centrality. *Optimization Methods and Software*, 32(2):369–399, 2017.

[143] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974.

[144] K. Wagstaff and C. Cardie. Clustering with instance-level constraints. In *Proceedings of 17th International Conference on Machine Learning*, pages 1103–1110. Standford, 2000.

[145] K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl. Constrained k-means clustering with background knowledge. In *Proceedings of 18th International Conference on Machine Learning*, volume 1, pages 577–584. Williams College, 2001.

[146] K. L. Wagstaff. *Intelligent clustering with instance-level constraints*. PhD thesis, 2002.

[147] H. Wang and J. Marron. Object oriented data analysis: Sets of trees. *The Annals of Statistics*, 35(5):1849–1873, 2007.

[148] X. Wang, C. Wang, and J. Shen. Semi-supervised k-means clustering by optimizing initial cluster centers. In *International Conference on Web Information Systems and Mining*, pages 178–187. Springer, 2011.

[149] Y. Wang, Y. Xiang, J. Zhang, and S. Yu. Internet traffic clustering with constraints. In *8th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 619–624. IEEE, 2012.

[150] S. Wasserman and K. Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

[151] R. C. Wilson and P. Zhu. A study of graph spectra for comparing graphs and trees. *Pattern Recognition*, 41(9):2833–2841, 2008.

[152] K. Worsley. An improved bonferroni inequality and applications. *Biometrika*, 69(2):297–302, 1982.

[153] I. Xenarios, E. Fernandez, L. Salwinski, X. J. Duan, M. J. Thompson, E. M. Marcotte, and D. Eisenberg. Dip: the database of interacting proteins: 2001 update. *Nucleic acids research*, 29(1):239–241, 2001.

[154] E. P. Xing, M. I. Jordan, S. Russell, and A. Y. Ng. Distance metric learning with application to clustering with side-information. In *Proceedings of the Advances in neural information processing systems 15*, pages 505–512. MIT Press, 2002.

[155] R. Xu and D. Wunsch. *Clustering*, volume 10. John Wiley & Sons, Hoboken, New Jersey, 2008.

[156] J. Yang, F. Alajaji, and G. Takahara. New bounds on the probability of a finite union of events. In *Information Theory (ISIT), 2014 IEEE International Symposium on*, pages 1271–1275. IEEE, 2014.

[157] J. Yang, F. Alajaji, and G. Takahara. On bounding the union probability using partial weighted information. *Statistics & Probability Letters*, 116:38–44, 2016.

[158] S. X. Yu and J. Shi. Segmentation given partial grouping constraints. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):173–183, 2004.

[159] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, pages 452–473, 1977.

[160] K. Zhang. A constrained edit distance between unordered labeled trees. *Algorithmica*, 15(3):205–222, 1996.

[161] Z. Zhang, J. T. Kwok, and D.-Y. Yeung. Parametric distance metric learning with label information. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1450–1452, 2003.

[162] S. Zhu, D. Wang, and T. Li. Data clustering with size constraints. *Knowledge-Based Systems*, 23(8):883–889, 2010.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name	: Dinler, Derya
Nationality	: Turkish (TC)
Date and Place of Birth	: 11 November 1989, Silivri
Phone	: +90 555 852 43 32
E-mail	: dinler@metu.edu.tr

EDUCATION

Degree	**Institution**	**Year of Graduation**
PhD	METU Industrial Engineering	2019
MS	METU Operations Research	2013
BS	METU Industrial Engineering	2011
High School	Bolu Science High School, Bolu	2006

WORK EXPERIENCE

Year	**Place**	**Enrollment**
2019-Present	Hacettepe University	Instructor
2011-2019	METU	Research Assistant
2013-2018	Aselsan	Project Member
2010-2011	METU	Student Assistant
2010-2011	Renault	Project Member
June 2011	Renault	Intern
June 2009	Banvit	Intern

FOREIGN LANGUAGES

English – Proficient user, Turkish – Native speaker

PUBLICATIONS

1. Dinler, D., & Tural M. K. (2018). Faster Computation of Successive Bounds on the Group Betweenness Centrality, Networks, 71 (4), 358-380. https://doi.org/10.1002/net.21817

2. Dinler, D., & Tural M. K. (2017). Robust Semi-Supervised Clustering with Polyhedral and Circular Uncertainty, Neurocomputing, 265, 4-27. http://dx.doi.org/10.1016/j.neucom.2017.04.073

3. Dinler, D., & Tural, M. K. (2016). A Minisum Location Problem with Regional Demand Considering Farthest Euclidean Distances. Optimization Methods and Software, 31 (3), 446-470. http://dx.doi.org/10.1080/10556788.2015.1121486

4. Dinler, D., & Tural, M. K. (2016). A Survey of Constrained Clustering. In E. Celebi, K. Aydin (Eds.), Unsupervised Learning Algorithms (pp. 207-235). Switzerland: Springer.

5. Dinler, D., Tural, M. K., & Iyigun, C. (2015). Heuristics for a Continuous Multi-facility Location Problem with Demand Regions. Computers & Operations Research, 62, 237-256. http://dx.doi.org/10.1016/j.cor.2014.09.001

6. Damgacioglu, H., Dinler, D., Ozdemirel, N. E., & Iyigun, C. (2015). A Genetic Algorithm for the Uncapacitated Single Allocation Planar Hub Location Problem. Computers & Operations Research, 62, 224-236. http://dx.doi.org/10.1016/j.cor.2014.09.003

7. Dinler, D., Tural, M. K., & Iyigun, C. (2014). Location Problems with Demand Regions. In B. Y. Kara, I. Sabuncuoglu, B. Bidanda (Eds.), Global Logistics Management (pp. 237-252). Boca Raton: Taylor & Francis (CRC Press).