DIMENSION REDUCTION FOR TREE-STRUCTURED DATA

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

BY

ERDİNÇ DURAK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

SEPTEMBER 2021

Approval of the thesis:

DIMENSION REDUCTION FOR TREE-STRUCTURED DATA

submitted by **ERDİNÇ DURAK** in partial fulfillment of the requirements for the degree of **Master of Science in Industrial Engineering Department, Middle East Technical University** by,

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

Prof. Dr. Esra Karasakal
Head of Department, **Industrial Engineering** _____

Assoc. Prof. Dr. Mustafa Kemal Tural
Supervisor, **Industrial Engineering, METU** _____

Prof. Dr. Cem İyigün
Co-supervisor, **Industrial Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Nur Evin Özdemirel
Industrial Engineering, METU _____

Assoc. Prof. Dr. Mustafa Kemal Tural
Industrial Engineering, METU _____

Prof. Dr. Cem İyigün
Industrial Engineering, METU _____

Prof. Dr. Sinan Gürel
Industrial Engineering, METU _____

Assoc. Prof. Dr. Ayşegül Altın Kayhan
Industrial Engineering, TOBB ETU _____

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: Erdinç Durak

Signature :

ABSTRACT

DIMENSION REDUCTION FOR TREE-STRUCTURED DATA

Durak, Erdinç
M.S., Department of Industrial Engineering
Supervisor: Assoc. Prof. Dr. Mustafa Kemal Tural
Co-Supervisor: Prof. Dr. Cem İyigün

September 2021, 97 pages

Statistical analysis of tree-structured data is one of the exciting research areas with expanding application areas. In classical data analysis, the data objects are points in the Euclidean space, whereas they are trees in the analysis of tree-structured data. The replacement of points in the Euclidean space with trees brings in additional complexity in data analysis and necessitates the use of dimension reduction techniques. In this study, we aim to develop dimension reduction techniques for tree-structured data where each tree is rooted and labeled. We consider two classical dimension reduction techniques; namely, the principal component analysis (PCA) and multidimensional scaling (MDS), and adapt them to tree-structured data. Unlike a previous study on the PCA for tree-structured data, the PCA techniques proposed in this thesis maximize a measure of variance. Computational experiments on randomly generated data and real life data show the superiority of the proposed PCA techniques over the existing one.

In the literature, there is no study that performs MDS on tree-structured data to project them in the tree space. In this direction, we propose the first MDS method for tree-

structured data, where the edges of the trees are considered as the dimensions. In the proposed MDS method, the aim is to keep the Hamming distances between pairs of trees proportionally similar. For this purpose, we propose a mixed-integer linear programming model which finds the edges to be kept in an optimal way and heuristic methods where the edges are greedily selected one by one. Computational experiments show that the proposed MDS methods are successful in keeping useful information as high clustering accuracy is achieved with only a fraction of the edges.

To be able to perform the computational experiments in a systematic way, a random tree generator algorithm is developed. This algorithm is able to generate clusters of trees with different skewness and density parameters. By changing these parameters systematically, we are able to understand the strength and weaknesses of the proposed methods.

Keywords: tree-structured data, dimension reduction, statistical analysis on graphs, principal component analysis, multidimensional scaling

ÖZ

AĞAÇ YAPILI VERİLER İÇİN BOYUT İNDİRGEME

Durak, Erdinç
Yüksek Lisans, Endüstri Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Mustafa Kemal Tural
Ortak Tez Yöneticisi: Prof. Dr. Cem İyigün

Eylül 2021 , 97 sayfa

Ağaç yapılı verilerin istatistiksel analizi, genişleyen uygulama alanlarıyla ilgi çekici araştırma alanlarından biridir. Klasik veri analizinde, veri nesneleri Öklid uzayındaki noktalarken, ağaç yapılı veri analizinde ağaçlardır. Öklid uzayındaki noktaların ağaçlarla değiştirilmesi, veri analizinde ek karmaşıklık getirir ve boyut indirgeme tekniklerinin kullanılmasını gerektirir. Bu çalışmada, her ağacın köklü ve etiketli olduğu ağaç yapılı veriler için boyut indirgeme teknikleri geliştirmeyi hedefliyoruz. İki klasik boyut küçültme tekniğini ele alıyoruz; temel bileşen analizi (PCA) ve çok boyutlu ölçekleme (MDS), ve bunları ağaç yapılı verilere uyarlıyoruz. Ağaç yapılı veriler için PCA üzerine yapılan önceki bir çalışmanın aksine, bu tezde önerilen PCA teknikleri bir varyans ölçüsünü maksimize ediyor. Rastgele oluşturulmuş veriler ve gerçek hayat verileri üzerinde yapılan hesaplama deneyleri, önerilen PCA tekniklerinin mevcut olana göre üstünlüğünü göstermektedir.

Literatürde ağaç yapılı veriler üzerinde MDS gerçekleştirip ağaç uzayına yansıtan bir çalışma bulunmamaktadır. Bu doğrultuda, ağaçların kenarlarının boyut olarak kabul edildiği ağaç yapılı veriler için ilk MDS yöntemini öneriyoruz. Önerilen MDS yön-

teminde amaç, ağaç çiftleri arasındaki Hamming uzaklıklarını orantısal olarak benzer tutmaktır. Bu amaçla, tutulacak kenarları optimal bir şekilde bulan bir karma tamsayılı doğrusal programlama modeli ve kenarların açgözlülükle tek tek seçildiği sezgisel yöntemler önermekteyiz. Hesaplamalı deneyler, önerilen MDS yöntemlerinin, kenarların yalnızca bir kısmı ile yüksek kümeleme doğruluğu elde edildiğinden yararlı bilgileri tutmada başarılı olduğunu göstermektedir.

Hesaplamalı deneyleri sistematik bir şekilde yapabilmek için bir rastgele ağaç üreteci algoritması geliştirilmiştir. Bu algoritma, farklı eğiklik ve yoğunluk parametrelerine sahip ağaç kümeleri oluşturabilmektedir. Bu parametreleri sistematik olarak değiştirerek önerilen yöntemlerin güçlü ve zayıf yönlerini anlayabiliriz.

Anahtar Kelimeler: ağaç yapılı veri, boyut indirgeme, çizgelerde istatistiksel analiz, temel bileşen analizi, çok boyutlu ölçekleme analizi

To the beauty of variation and the variation of beauty...

ACKNOWLEDGMENTS

First of all, I would like to thank my supervisors Assoc. Prof. Dr. Mustafa Kemal Tural and Prof. Dr. Cem İyigün for their patience, support, and encouragement during the thesis period. They have taught me a lot. It was a great privilege and honor to work under their guidance.

Besides my advisors, I would like to present my gratitude to the examining committee members of this thesis, Prof. Dr. Nur Evin Özdemirel, Prof. Dr. Sinan Gürel, and Assoc. Prof. Dr. Ayşegül Altın Kayhan, for their comments that improves the quality of this thesis. I am grateful to be a student of such esteemed academicians and proud of being a member of the METU-IE department. I am also very thankful to all professors of the METU-IE department for their teachings.

It is a pleasure to acknowledge all my friends for their supports. I am thankful to my brother Ufuk Koca for always being there. He has believed in me more than even myself. For their supports and jokes, I would like to thank my friends from Kemküm, namely Alif, Beril, Cemre, Deniz, Gizem, and so on. They taught me to laugh at difficult times. Without them, I could not be motivated to complete this thesis.

I am grateful to Metehan Koç for his joyful but mind-blowing talks. For their beautiful conversations, I am also grateful to Özen Yılmaz and Elif Kübra Çontar. I have learned a lot from their perspectives. I should thank my dear roommates Kaan and Taylan Uğur for their understanding and supports. I would also like to thank Gamze Karaca, who contributed well to my writing skills.

I would also thank my dear colleagues Bilgenur Erdoğan, Dilay Özkan, and Ceyhan Şahin for easing my work with their endless support. Moreover, I should thank Yağmur Caner. She was always there to ask for help.

Last but not least, I would like to thank my parents, Mustafa and Nimet Durak. They are the ones who sacrificed the most for this study.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vii

ACKNOWLEDGMENTS x

TABLE OF CONTENTS xi

LIST OF TABLES xiv

LIST OF FIGURES xv

LIST OF ALGORITHMS xviii

LIST OF ABBREVIATIONS xix

CHAPTERS

1 INTRODUCTION 1

2 LITERATURE REVIEW 5

3 NOTATION AND DEFINITION OF THE PROBLEMS 11

4 PRINCIPAL COMPONENT ANALYSIS FOR TREE-STRUCTURED DATA 19

4.1 Principal Component Analysis 19

4.2 Motivation and Previous Work 20

4.3 Variance Maximization PCA for Tree-Structured Data 25

4.4 Similarity Minimization PCA for Tree-Structured Data 30

5 COMPUTATIONAL EXPERIMENTS FOR PCA METHODS FOR TREE-STRUCTURED DATA 37

5.1 The Performances of PCA Methods on Synthetic Data Sets 38

5.1.1 A Random Tree Generator 39

5.1.2 Computational Experiments on Randomly Generated Tree Sets 41

5.1.2.1 PCA Methods on Density-Different Clusters on Small Trees 45

5.1.2.2 PCA Methods on Density-Different Clusters on Big Trees 47

5.1.2.3 PCA Methods on Skewness-Different Clusters on Small Trees 49

5.1.2.4 PCA Methods on Skewness-Different Clusters on Big Trees 51

5.2 The Performances of PCA Methods on a Real Data Set 53

5.2.1 Computational Experiments on Real Data 55

6 MULTIDIMENSIONAL SCALING FOR TREE-STRUCTURED DATA . . 61

6.1 Multidimensional Scaling 61

6.2 MDS for Tree-Structured Data - Problem Definition 62

6.3 Multidimensional Scaling for Tree-Structured Data - Optimization Model 63

6.4 Multidimensional Scaling for Tree-Structured Data - Greedy Forward Algorithm 70

6.5 Multidimensional Scaling for Tree-Structured Data - Greedy Backward Algorithm 76

7 COMPUTATIONAL EXPERIMENTS FOR MDS METHODS FOR TREE-STRUCTURED DATA 81

7.1 The Performances of MDS Methods on Synthetic Data Sets 81

7.1.1 MDS Methods on Density-Different Clusters on Small Trees . 82

7.1.2 MDS Methods on Density-Different Clusters on Big Trees . . 84

7.1.3 MDS Methods on Skewness-Different Clusters on Small Trees 86

7.1.4 MDS Methods on Skewness-Different Clusters on Big Trees . 86

7.2 The Performances of MDS Methods on a Real Data Set 89

7.3 Comparison of the Performances of PCA and MDS Methods for Tree-Structured Data 91

8 CONCLUSION 93

REFERENCES 95

LIST OF TABLES

TABLES

Table 5.1 The adjusted Rand index scores of the PCA methods for tree-structured data for the instance where $h = 5$, skewness: *Too Left* for all trees, and the pair of density levels: (0.99-0.95) 43

Table 6.1 The results of MDST model on the set of trees given in Figure 3.1 . 66

Table 6.2 The results of MDSTGA method on the set of trees given in Figure 3.1 72

Table 6.3 The results of MDSTGB method on the set of trees given in 3.1 . . . 78

LIST OF FIGURES

FIGURES

Figure 3.1 A coherent set $S = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\}$ of trees 13

Figure 3.2 Support tree of the coherent set S given in Figure 3.1 14

Figure 3.3 The flowchart of the thesis 17

Figure 4.1 The first two principal components selected by DMIN method (dotted and dashed treelines, respectively) for the set of trees given in Figure 3.1 23

Figure 4.2 Projections of trees in Figure 3.1 onto the first two principal components selected by DMIN method 24

Figure 4.3 The first two principal components selected by VMAX method (dotted and dashed treelines, respectively) for the set of trees given in Figure 3.1 28

Figure 4.4 Projections of trees in Figure 3.1 onto the first two principal components selected by VMAX method 29

Figure 5.1 Clustering performances of the PCA methods for tree-structured data on density-different data sets with $h = 5$ 45

Figure 5.2 Clustering performances of the PCA methods for tree-structured data on density-different data sets with $h = 10$ 48

Figure 5.3 Clustering performances of the PCA methods for tree-structured data on skewness-different data sets with $h = 5$ 50

Figure 5.4 Clustering performances of the PCA methods for tree-structured data on skewness-different data sets with $h = 10$ 52

Figure 5.5 The support tree of the real data set 54

Figure 5.6 Principal components of the PCA methods for tree-structured on the support tree of the real data set 56

Figure 5.7 Clustering performances of the PCA methods for tree-structured data on the real data set 58

Figure 6.1 The scaling tree with 5 edges constructed by MDST method (dashed edges) for the set of trees given in Figure 3.1 67

Figure 6.2 The scaling tree with 8 edges constructed by MDST method (dashed edges) for the set of trees given in Figure 3.1 67

Figure 6.3 Projections of trees given in Figure 3.1 onto the scaling tree shown in Figure 6.1 68

Figure 6.4 Projections of trees given in Figure 3.1 on the scaling tree shown in Figure 6.2 69

Figure 6.5 The scaling tree with 8 edges constructed by MDSTGA method (dashed edges) for the set of trees given in Figure 3.1 74

Figure 6.6 Projections of trees in Figure 3.1 on the scaling tree shown in Figure 6.5 75

Figure 7.1 Clustering performances of the MDS methods for tree-structured data on density-different data sets with $h = 5$ 83

Figure 7.2 Clustering performances of the MDS methods for tree-structured data on density-different data sets with $h = 10$ 85

Figure 7.3 Clustering performances of the MDS methods for tree-structured data on skewness-different data sets with $h = 5$ 87

Figure 7.4 Clustering performances of the MDS methods for tree-structured data on skewness-different data sets with $h = 10$ 88

Figure 7.5 Clustering performances of the MDS methods for tree-structured data on the real data set 90

LIST OF ALGORITHMS

ALGORITHMS

Algorithm 1 VMAX 27

Algorithm 2 SMIN 32

Algorithm 3 RTC 40

Algorithm 4 MDSTGA 71

Algorithm 5 MDSTGB 77

LIST OF ABBREVIATIONS

PCA	Principal Component Analysis
MDS	Multidimensional Scaling
MILP	Mixed-Integer Linear Programming
DMIN	Distance Minimization Principal Component Analysis for Tree-Structured Data Method
VMAX	Variance Maximization Principal Component Analysis for Tree-Structured Data Method
SMIN	Similarity Minimization Principal Component Analysis for Tree-Structured Data
RTC	Random Tree Constructor
ARI	Adjusted Rand Index
PC	Principal Component
MDST	Multidimensional Scaling for Tree-Structured Data Method
MDSTGA	Multidimensional Scaling for Tree-Structured Data - Greedy Algorithm
MDSTGB	Multidimensional Scaling for Tree-Structured Data - Greedy Backward Algorithm

CHAPTER 1

INTRODUCTION

Working in high dimensional spaces has many difficulties in data analysis, such as the curse of dimensionality, computational inefficiency, and visualization complexity. Dimension reduction techniques are helpful to avoid those difficulties. These techniques are simply representing the original data on high dimensional space with lower dimensional data. Principal component analysis (PCA) and multidimensional scaling (MDS) are two of the most commonly used dimension reduction techniques.

Including the PCA and MDS, many of the classical dimension reduction techniques are designed to work with vectors in the Euclidean space. Besides, the increasing need for deeper analyses, measurement capabilities, and data collecting technology result in the collection of more complex data sets, which include different data objects such as images, shapes, and graphs [1]. The collection of such data sets led to the birth of object-oriented data analysis, where the data points in the Euclidean space are replaced with more complex data objects, including images, shapes, and graphs. The use of data objects in non-Euclidean spaces brings in additional complexity in data analysis and necessitates the use of dimension reduction techniques.

In this study, we aim to develop dimension reduction techniques for data objects that are trees. Such data objects appear in several application areas, such as neuroscience, marketing, and anatomy. Using trees as data objects, instead of the points in the Euclidean space, may help to represent the parent-child relation between the features. For instance, the branching structures of the blood vessels are represented by trees in [2]. The branching points are shown by the nodes, and the vessels between the branching points are represented by the edges. If one try to represent these blood vessels by data points in the Euclidean space, then it would be difficult to keep the

information that which vessels are connected to each other with the branching points. Such information loss may impair the success of the data analysis. Using trees as data objects can help to avoid such a loss.

Dimension reduction for tree-structured data may be useful in several ways. For example, it can be used to reduce the sizes of trees, e.g., brain artery structures of patients [2], in tree clustering problems to expedite the clustering process. Dimension reduction may be performed to be able to make relevant visualizations about trees. For example, the purchases of customers, which may be organized as trees [3], may be visualized after dimension reduction to understand the characteristics of purchasing behavior of customers. Note that for a given set of trees, the dimension reduction techniques proposed in this study return (possibly) smaller trees which are *subtrees* of the original ones. With this property, dimension reduction may be helpful in reducing the data collection effort or cost by only requiring the collection of the data in the reduced trees. This property also allows one to understand the parts of the original trees that carry the most useful information. Note that in the classical PCA (in the Euclidean space), principal components do not need to correspond to the original features of the data, which makes it difficult to interpret the results obtained using the reduced points in terms of the original features. The same conclusion also holds true for the classical MDS.

As a data object, a tree can be in different forms. It can be rooted or unrooted, and it may have some labels on the nodes. The nodes and edges of a tree can carry some attributes. The number of branches in all branching points may be limited, e.g., they are limited with two branches in binary trees. In this study, we consider rooted and labeled trees which do not carry any attributes on their nodes or edges, i.e., we only focus on the branching structure (topology) of rooted labeled trees.

Given a set of rooted labeled trees, we aim to represent these trees using smaller ones in such a way that the "useful information" is kept as much as possible. For this purpose, in this study, we propose some dimension reduction techniques for tree-structured data. We consider the classical PCA and MDS that work for data objects that are points in the Euclidean space and propose their extensions to make the methods work for data objects in the tree space. In PCA for tree-structured data, we take

the treelines proposed in [4] as the principal components. A previous study [2] proposes a PCA method for tree-structured data with treelines being the principal components. Our PCA methods are different from that of Aydin et al. [2] in terms of the objective functions. The objective functions we use are inspired by the variance maximization objective of the classical PCA, whereas the distance minimization (between original trees and the projected ones) objective is used in [2].

As a second dimension reduction technique, we propose MDS methods for tree-structured data. These methods aim to keep the Hamming distances between pairs of trees proportionally similar by selecting a subset of the edges. For this purpose, we propose a mixed-integer linear programming (MILP) model, which selects the edges to be kept in an optimal way. This model is not able to solve medium size instances to optimality in 1 hour. Therefore we also propose heuristic methods for the problem in which the edges are greedily selected.

In order to compare the performances of different dimension reduction methods proposed for tree-structured data, we design a random tree generator that can produce different types of rooted labeled trees with different cluster structures. For clustering purposes, the tree clustering algorithms proposed by [5] are used and the quality of the resulting clusters are evaluated by the adjusted Rand index [6]. After the comparison of the methods on randomly generated data sets, we compare them using a brain artery data set provided by Aylward and Bullitt [7] which is later extended by Aydin et al. [2]. There are 93 rooted binary trees in this data set, where each one represents the brain artery structure of an individual.

This study is one of the few studies on dimension reduction for graph-structured data. The main contributions of the study can be summarized as follows:

- Two novel PCA methods for tree-structured data which use variance maximization objective functions are provided. The experiments show that our methods are better than the previously proposed one in terms of clustering accuracy.
- The computational experiments also show that by using only a small subset of the treelines, high clustering accuracies can be obtained.
- This is the first study on MDS for tree-structured data that projects them in

the tree space. An MILP model and two heuristic algorithms are proposed for this purpose. The computational experiments demonstrate that the methods successfully select informative edges.

- A novel random rooted labeled tree construction method is provided, which allows us to systematically analyze the performances of the proposed methods on data with different cluster structures.

The thesis is organized as follows. We provide a literature review in Chapter 2. Chapter 3 introduces the notation used and the problems studied in the thesis. In Chapter 4, we give the details of the previous and the newly proposed PCA methods for tree-structured data. We test these methods on randomly generated and real data sets in Chapter 5. In Chapter 6, we explain the MILP model and the heuristic algorithms proposed as the MDS methods for tree-structured data and give the results of the related computational experiments in Chapter 7. Finally, in Chapter 8, the conclusion and future research directions are provided.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we give a review of the related literature in the following order. First, classical dimension reduction techniques are briefly reviewed. Second, we explain the usage areas of tree-structured data and discuss statistical studies on data analysis techniques for tree-structured data. Finally, we review the literature on dimension reduction techniques for tree-structured data.

Dimension reduction techniques are used to represent original high dimensional data in a lower-dimensional space while keeping useful information about the original data. Dimension reduction techniques can be classified as feature selection and feature projection methods. Feature selection methods decide which features are essential for the data set, and they construct the lower-dimensional data by removing the non-essential features from the data set. The feature projection methods project the data set to extract useful information. They are also called feature extraction methods. The difference between these methods is that the features of the lower-dimensional data constructed by a feature selection method are a subset of the original features while they may not be among the original features in the feature projection methods.

Principal component analysis (PCA) is one of the most commonly used dimension reduction techniques. It is a feature projection method that is introduced by Pearson [8], and independently developed and named by Hotelling [9]. PCA takes a set of vectors as an input. By projecting the vectors onto the principal components, it returns a set of projected vectors where the first principal component is the one that carries the most information. For $k \geq 2$, the $k-$th principal component carries the most information after excluding the first $k-1$ principal components from the data. Therefore, lower-dimensional data can be obtained by projecting the data onto the

first few principal components. In Section 4.1, PCA is explained in some more detail.

Multidimensional scaling (MDS) is another dimension reduction technique. It is first introduced by Torgerson [10] and is defined in [11] as a problem to represent n data objects as n points in Euclidean $t-$ dimensional space such that pairwise distances between the points are similar to the given pairwise distances between the data objects. Note that t is defined by the user, and the original data set is not explicitly given but instead implicitly given by the pairwise distances. By selecting t closer to n, a data set can be constructed such that the pairwise distances between the data points are very close to the original pairwise distances. If t is selected as a small number, lower-dimensional data can be obtained. There are many methods perform MDS, see e.g., [12] and [10]. We provide more details about MDS in Section 6.1.

Trees are connected acyclic graphs that are useful to represent different data objects in many areas such as neuroscience, marketing, and anatomy. Many studies benefit from trees in the literature, and we briefly review some of them here. Phylogenetic trees are used to show the evolutionary relationships of various biological species in many studies, see e.g., [13] and [14]]. Aydin et al. [2] represent blood vessels on brains with rooted binary trees. Similarly, Lu and Miao [15] represent retinal vessels with rooted trees. XML documents are modeled as tree-like structures by Dalamagas et al. [16]. Another example is available in the study of Chen et al. [3] where purchases of customers are organized as rooted trees.

Statistical analysis of tree-structured data objects is a relatively new research area. Wang and Marron [4] is one of the first studies on statistical analysis of trees. They extend the field of functional data analysis to object-oriented data analysis, where the data objects can be more complex objects than vectors in the Euclidean space. They provide a mathematical framework for objects in non-Euclidean spaces, including tree spaces. Shen et al. [17] provide two representations of trees which enable them to perform functional data analysis to obtain statistical properties of tree-structured data objects. They also propose a tree-pruning technique to focus on important structures of tree populations. Wang et al. [18] present an extension of nonparametric regression where the response variables are tree-structured.

Tree clustering is one of the most widely studied data analysis techniques for tree-

structured data objects, see e.g., [5], [15], and [19]. For example, in [5], Dinler et al. propose tree k-means based clustering algorithms for rooted trees. In this study, they compare the results of the classical k-means algorithm on the vector representations of the trees and the proposed tree clustering algorithm on the trees. They show that the clustering accuracies of the tree clustering methods are much higher than those obtained by clustering the vector representations with the classical k-means algorithm. This result is one of our main motivations to perform dimension reduction in the tree space instead of representing trees as vectors in the Euclidean space.

PCA is the most studied dimension reduction method for tree-structured data. Firstly, Wang and Marron [4] develop an analog of the classical PCA in the binary tree space. In their approach, they view the PCA as a sequence of one-dimensional representations and present the treeline as a one-dimensional subspace of rooted trees. Here the treeline is defined as a sequence of trees where each tree is obtained by adding a single node to the previous tree where this node is a child of the lastly added node. A definition for the projection of any tree onto a treeline is proposed as the closest tree on the treeline using the Hamming distance, and they show that this projection is unique. They formulate a notion of principal component for trees as an optimization problem whose objective function is to minimize the sum of all Hamming distances between the original trees and their projections onto a treeline.

For the optimization problem presented by [4], Aydin et al. [2] give a linear-time algorithm. This is the first PCA algorithm developed for tree-structured data, where the data objects are binary trees with no attributes on the nodes and edges. They show that minimizing the sum of all Hamming distances between the original trees and their projections onto a treeline is equivalent to maximizing the sum of the weights of the nodes on this treeline. Here the weight of a node is the number of trees in the given set of trees that have this node. In Section 4.2, we explain this algorithm in detail. The authors apply their algorithm on a set of brain artery structures of 73 patients, which is provided in the study of Aylward and Bullitt [7]. They represent the branching structure of arteries with rooted trees where the nodes represent the branch points, and the edges represent the vessels between the branching points. They provide two possible node correspondence, i.e., the choice of which child branch is put on the left and which is put on the right for the trees. In the first one, descendant

correspondence, nodes with more descendants are placed on the left-hand side. The node whose corresponding vessel has a bigger radius is placed on the left-hand side in thickness correspondence. Using these node correspondence definitions, they generate two different binary trees for each location (top, left, back, right) of each patient's brain and apply their algorithm on these eight sets of trees. On one of these eight sets of trees, they find a significant correlation between the age of patients and the number of nodes in the projections of the trees onto their first principal components. They also suggest that descendant correspondence should be the default choice for future works.

Alfaro et al. [20] extend the given algorithm in [2] for any rooted tree. They provide the generalized version of the definitions and the algorithm. In order to perform a dimension reduction on a rooted tree, they develop the backward principal components, i.e., the components that carry the minimum amount of information, and propose an algorithm to find them. Here, they use the same distance-based objective as in [4]. They show the equivalence of the backward and the forward algorithms as well.

Aydin et al. [21] introduce k-treelines as an extension of the treelines. k-treelines are similar to treelines with a single difference that while obtaining a tree, the node to add to the previous tree can be a child node of any one of the lastly added k nodes. Note that treeline is a special case of k-treeline when k is 1. They also define tree-curve, which is another special case of k-treeline when k is infinite. Note that the last tree of any tree-curve is the tree that contains all edges of all the tree objects. When k is not 1, the projection of a tree onto a k-treeline is not unique, and there may exist some edges on the projection that is not on the original tree. They provide the optimization model to find the optimal k-treelines that minimizes the similar distance-based objective function as used in [4]. Since this model cannot solve medium size instances in optimality in reasonable times, they also propose some heuristic algorithms. They consider binary and rooted trees where node correspondence is known.

In addition to the mentioned studies about statistical analysis of rooted trees, there are a few other studies on phylogenetic trees. A study about PCA on phylogenetic trees is carried out by Nye [22]. The author proposes an algorithm for PCA on phylogenetic trees. The algorithm takes a set of phylogenetic trees on some fixed set of taxa,

not vectors, as an input and finds a fixed line through the central point of the trees, which optimizes the objective function. Here the projection of a tree onto a line is the closest point on the line to the tree. For the projection and the central point calculations, the author uses geodesic metrics. There are two options for the algorithm's objective function: maximizing the variance and minimizing the sum of squared distances. Note that these objectives are equivalent in the Euclidean space. Therefore, this study is the first that uses a variance-based objective function on a dimension reduction method for tree-structured data, where the data objects are phylogenetic trees. Since the introduced optimization problem is computationally demanding, the author proposes a greedy algorithm and a Monte Carlo optimization approach.

MDS for tree-structured data is much less studied than the PCA. Hillis et al. [23] carry the first study that uses MDS for tree-structured data, where the data objects are phylogenetic trees. In this study, the authors map the phylogenetic trees onto the Euclidean space using the Robinson-Foulds distance. Observe that the projections are not trees but points in the Euclidean space. Skwerer et al. [24] also perform MDS on phylogenetic trees to map them onto the Euclidean space. Observe that there is no study in the literature that proposes an MDS method for tree-structured data that projects the trees onto the tree space.

CHAPTER 3

NOTATION AND DEFINITION OF THE PROBLEMS

In this chapter, we introduce the basic notation used in the thesis and define the problems of interest. We denote by $G = (N, E)$ a graph, where N and E denote the node and edge sets of G, respectively. If the node and edge sets of a graph G are not explicitly given, we sometimes use the notation $N(G)$ and $E(G)$ to represent these sets, respectively. Given an edge set E, the graph induced by E is the graph whose node set is the union of the endpoints of the edges in E and whose edge set is equal to E. A path is a sequence of distinct nodes and edges $v_0, e_0, v_1, e_1, \ldots, v_n$ where v_i is a node for $i \in \{1, 2, \ldots, n\}$ and e_i is the edge that joins v_i and v_{i+1} for $i \in \{1, 2, \ldots, n-1\}$. A graph is a *tree* if and only if there exists exactly one path between any two nodes. Thus, a tree is an acyclic connected graph. Objects of interest of this study are rooted and labeled trees where we focus on their topology, i.e., the branching structure of them.

A tree is rooted if one of its nodes is selected as the root. The parent of a node v is the first visited node on the path starting from v to the root on a rooted tree. If node u is the parent of v, then v is called a child of u. A node is a *leaf node* if it has no children. If two nodes have the same parent, then they are called siblings. If nodes w and u are a child and the parent of node v, respectively, then the edge $\{u, v\}$ is called the parent of the edge $\{v, w\}$. In this case, the edge $\{v, w\}$ is called a child of the edge $\{u, v\}$. An edge is a *leaf edge* if it has no children. Let v be a child of u. If an edge is on the path between the root and u, then it is called an ascending edge of $\{u, v\}$. The level of a node u is the number of edges on the path from u to the root. Note that the level of a child node is one greater than the level of its parent, and the level of the root is zero. The maximum of the levels of all nodes of a tree gives the height of the tree. A

labeled tree is a tree where a unique label is given to each node. From now on, when we say node u, we mean the node with label u, and the node and edge sets of trees are written in terms of the labels. In the rest of the thesis, all trees are rooted and labeled unless otherwise stated.

If the nodes of a tree are restricted to have at most m children, then the tree is called an m-ary tree. 2-ary trees are also named binary trees. A complete m-ary tree of height h is the tree whose all nodes except the ones in level h have exactly m children.

A set of (rooted and labeled) trees is said to be coherent if their roots have the same label and the parent-child relations are consistent with the labels, i.e., if node u is the parent of node v in a tree, and if node v exists in another tree, then its parent has to be u in this tree as well. In Figure 3.1, an example of a coherent set of trees is provided with the numbers next to the nodes representing the labels. The root is the same for all trees, which has label 1. Each tree is a binary tree of height 3. This data set is constructed with the random tree generator method explained in Section 5.1.1. t_1, t_2, t_3, and t_4 are generated with the same parameter set while another set of parameters is used for t_5, t_6, t_7, and t_8.

Let $S = \{t_1, t_2, \ldots, t_n\}$ be a coherent set of n many trees, where $t_i = (N_i, E_i)$ for each $i \in \{1, 2, \ldots, n\}$. We represent the union of two trees t_i and t_j by $t_i \cup t_j$ where we have that $t_i \cup t_j = (N_i \cup N_j\ ,\ E_i \cup E_j)$. Note that $t_i \cup t_j$ is also a tree. The intersection of two trees t_i and t_j is a tree, represented by $t_i \cap t_j$, with $t_i \cap t_j = (N_i \cap N_j\ ,\ E_i \cap E_j)$. $t_i \setminus t_j$ represents a graph that is induced by $E_i \setminus E_j$. Note that, for two trees t_i and t_j, $t_i \setminus t_j$ may not be a tree.

Given a coherent set of trees $S = \{t_1, t_2, \ldots, t_n\}$ with $t_i = (N_i, E_i)$ for each $i \in \{1, 2, \ldots, n\}$, the union of all trees in S is called the *support tree* of S and is denoted by ST. Note that ST is a tree with $N(ST) = \bigcup_{i=1}^{n} N_i$ and $E(ST) = \bigcup_{i=1}^{n} E_i$ and each tree in S is a subtree of ST. For a given edge e of ST, we denote the set of all ascending and children edges of e on the support tree by $asc(e)$ and $ch(e)$, respectively. LN and LE represent the set of all leaf nodes and leaf edges of the support tree, respectively. The frequency of an edge e with respect to set S, denoted by $freq(e)$, is the number of trees in S which include edge e. Mathematically, we

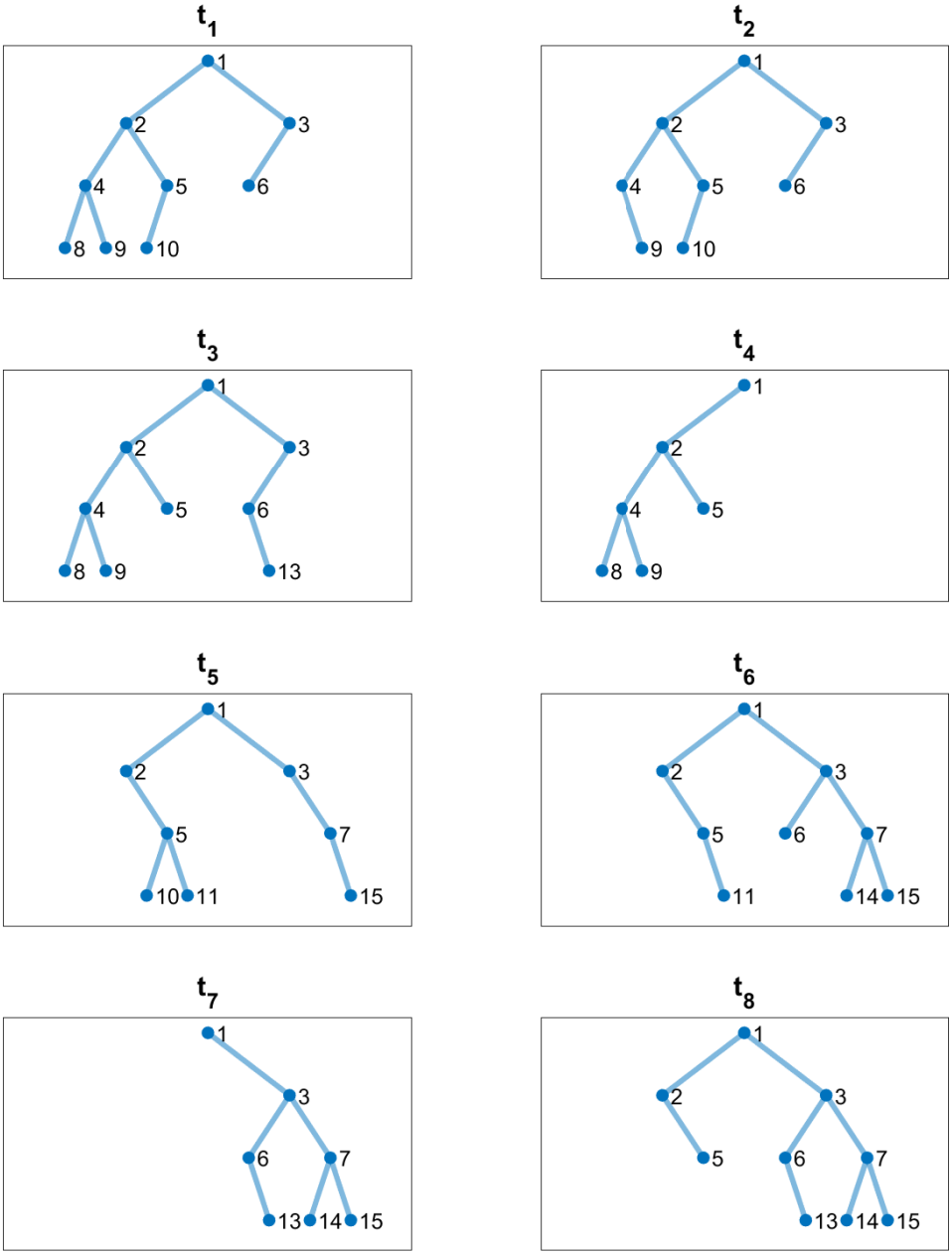

Figure 3.1: A coherent set $S = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\}$ of trees

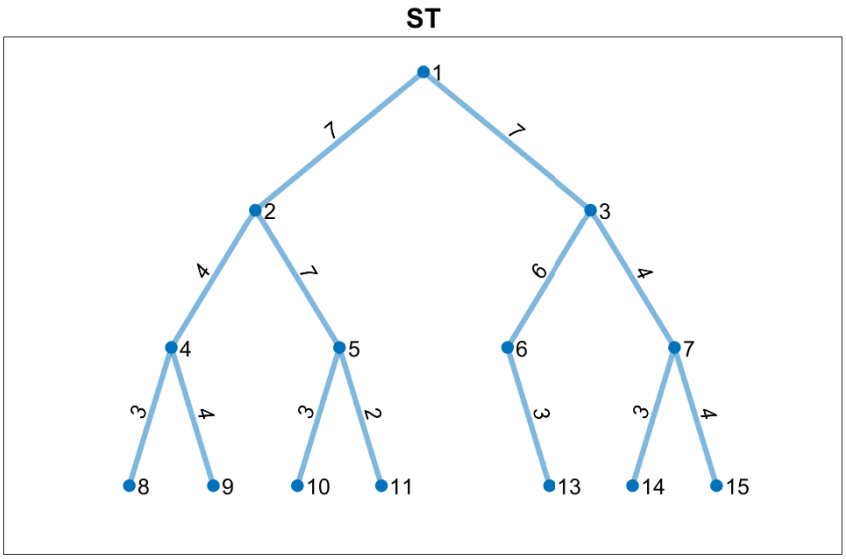

Figure 3.2: Support tree of the coherent set S given in Figure 3.1

have that

$$freq(e) = \sum_{i=1}^{n} \mid E_i \cap \{e\} \mid . \tag{3.1}$$

Observe that if $f \in asc(e)$, then $freq(f) \geq freq(e)$. Note that the dependencies of $ST, asc(e), ch(e), LN, LE$, and $freq(e)$ on S are omitted in the notation as they will be clear from the context.

Given a coherent set of trees $S = \{t_1, t_2, \ldots, t_n\}$ and its support tree ST, if t' is a subtree of ST with its root being the same as the root of ST, then we have that $\{t_1, t_2, \ldots, t_n, t'\}$ is also a coherent set of trees.

The support tree of the coherent set of trees S shown in Figure 3.1 is provided in Figure 3.2 where the numbers on the edges represent the frequencies of the edges with respect to S. Note that $|LN| = |LE| = 7$.

We borrow the idea of a treeline from [4]. With a slight change, we define a treeline as a path between the root and a leaf node in the support tree. The treelines are in one-to-one correspondence with the leaf nodes of the support tree, and we represent the set of all treelines by $L = \{\ell_1, \ell_2, \ldots, \ell_{|LN|}\}$.

The projection of a tree t onto a treeline ℓ, $P_\ell(t)$, is defined as

$$P_\ell(t) = t \cap \ell. \tag{3.2}$$

The projection of a tree t onto a set of treelines L, $P_L(t)$, is similarly defined as follows.

$$P_L(t) = t \cap \bigcup_{\ell \in L} \ell. \tag{3.3}$$

The projection of a graph G onto a treeline ℓ, denoted by $P_\ell(G)$, is the graph induced by $E(G) \cap E(\ell)$. The projection of a graph G onto a set of treelines L, denoted by $P_L(G)$, is defined as the graph that is induced by $E(G) \cap (\bigcup_{\ell \in L} E(\ell))$.

Given two graphs $G_i = (N_i, E_i)$ and $G_j = (N_j, E_j)$, the distance between them, denoted by $d(G_i, G_j)$, is measured by the *Hamming distance* [25]. Mathematically, we have that

$$d(G_i, G_j) = \mid E_i \setminus E_j \mid + \mid E_j \setminus E_i \mid . \tag{3.4}$$

The similarity of them, denoted by $s(G_i, G_j)$, is measured by the *vertex / edge overlap* [26]. It is equal to

$$s(G_i, G_j) = \frac{2 \mid E_i \cap E_j \mid +1}{2 (\mid E_i \mid + \mid E_j \mid) + 2}. \tag{3.5}$$

In this study, our objective is to reduce the sizes of the trees for a given coherent set of trees while keeping maximum information about the data set. In other words, we are interested in dimension reduction for tree-structured data. For this purpose, we try to adapt principal component analysis (PCA) and multidimensional scaling (MDS) to tree-structured data.

In PCA, we follow the steps of Aydin et al. [2] and represent the principal components by treelines. Accordingly, principal components can be selected from the set of all treelines corresponding to a given coherent set of trees. The set of the first k principal components is indicated by $L_k^* = \{\ell_1^*,...,\ell_k^*\}$ where ℓ_i^* is the treeline that represents the $i-$th principal component. The $k-$th principal component tree is defined as the

union of the first k principal components and is equal to

$$pc_k = \bigcup_{i=1}^{k} \ell_i^*. \tag{3.6}$$

With this notation, the projection of a tree t onto the first k principal components can be computed as

$$P_{L_k^*}(t) = t \cap pc_k \tag{3.7}$$

which is obtained by rewriting (3.3) for the first k principal components.

In PCA for tree-structured data, we define our problem as the problem of selecting the treelines from the set of treelines that carry the maximum amount of information on it. Note that by changing the measure of the amount of information carried by a treeline, we can obtain different PCA methods for tree-structured data. In the next chapter, we explain the PCA in the Euclidean space and the tree space, present the previous work, and propose novel methods for selecting the best treelines as the principal components.

In MDS, we represent the dimensions by edges and select the edges from the edge set of the support tree of a given coherent set of trees. The $k-$th scaling tree, sc_k, is defined as the tree induced by the set of first k selected edges. The projection of a tree t_i onto sc_k, denoted by $P_k(t_i)$, can be computed as

$$P_k(t_i) = t_i \cap sc_k. \tag{3.8}$$

In MDS for tree-structured data, our problem is to select the edges of ST to be in sc_k while keeping the pairwise Hamming distances between the projected trees proportionally similar to the pairwise Hamming distances between the original trees. We explain the multidimensional scaling in the Euclidean space and the tree space, propose an exact method to select the optimal edges, two greedy algorithms to select the edges sequentially to be able to perform MDS in the tree space.

In Figure 3.3, the flowchart of the thesis is provided. Here we have tree-structured data sets whose original partitions of clusters are known. We reduce the sizes of trees and generate the projected data sets using the PCA and MDS methods for tree-structured data. By inputting the projected data sets into a tree clustering algorithm, we obtain the partitions of the projected data sets. At the end, by comparing the

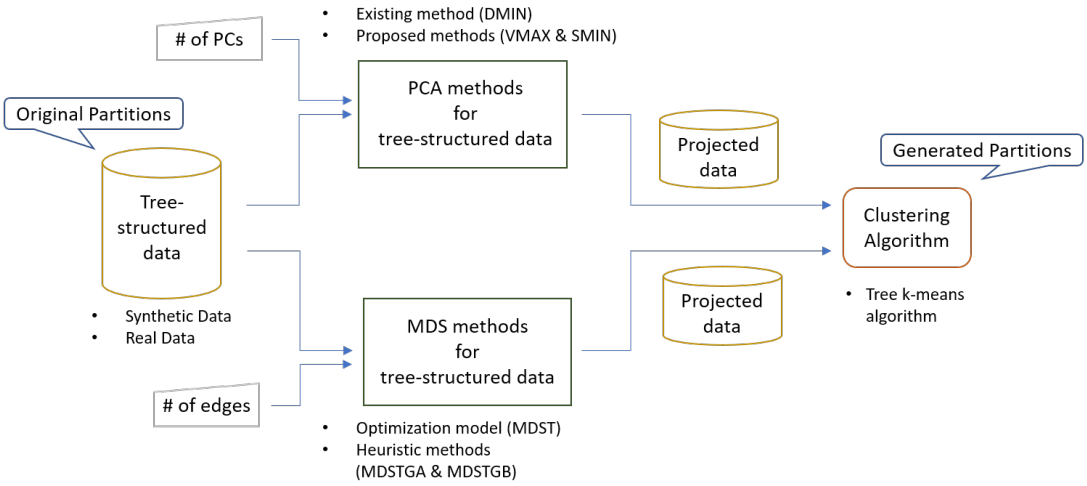

Figure 3.3: The flowchart of the thesis

original and the generated partitions, we measure the performances of the proposed methods. In the following chapters, we explain the PCA and MDS methods for tree-structured data and corresponding computational studies.

CHAPTER 4

PRINCIPAL COMPONENT ANALYSIS FOR TREE-STRUCTURED DATA

4.1 Principal Component Analysis

Principal Component Analysis (PCA) is a data analysis technique that helps reduce the number of dimensions of the data points with minimum loss of information about the data. PCA performs a change of basis on the data using an orthonormal basis constructed by linearly independent vectors. In PCA, a sequence of orthonormal vectors called principal components is obtained using the data where the first principal component is the vector that maximizes the variance of the lengths of the projections of the data points onto the line spanned by this vector. For any $k \geq 2$, the $k-$th principal component is the vector that maximizes the variance of the lengths of the projections of the updated data points onto the line spanned by this vector. Here the updated data stands for the data obtained by projecting the data orthogonal to the subspace spanned by the first $k-1$ principal component vectors. Note that the vector that maximizes the variance of the lengths of the projections of the updated data points onto the line spanned by this vector is also the one that minimizes the sum of squared Euclidean distances between the updated data points and their projections onto the same line. Thus, in the Euclidean space, we have two equivalent objective functions that can be used to get the principal components: the variance maximization and the distance minimization objectives. After the change of basis, lower-dimensional data can be obtained by using only the first few principal components. In order to decide the number of principal components, the amount of information carried by the principal components can be used. The carried amount of information is measured by the variance of the lengths of the projections of the updated data points onto the line spanned by each principal component vector.

In the analysis of tree-structured data, the concepts of dimension reduction and principal component analysis are much less studied and understood. In the study of Aydin et al. [2], the principal components are taken as the treelines of the support tree. While the classical PCA in the Euclidean space computes the principal components, the PCA method proposed in [2] for tree-structured data *selects* the principal components from a set of treelines. For this reason, it can also be considered as a feature selection method. Projecting the trees onto only the first few treelines results in (possibly) smaller trees while hopefully keeping as much useful information as possible about the original tree-structured data. Our study, however, shows that the method proposed in [2] does not always keep enough useful information about the data. It is mainly because the variance maximization and distance minimization objectives are not equivalent in the tree space. The method proposed in [2] is based on the distance minimization objective. In contrast, the main objective should be variance maximization since it is related to the amount of useful information kept after the PCA. In this study, we propose PCA methods for tree-structured data that are based on the variance maximization objective and show their superiority over the method proposed in [2]. In the rest of this chapter, we give the details of the method proposed in [2] and the newly developed methods.

4.2 Motivation and Previous Work

The PCA method proposed by Aydin et al. [2] for tree-structured data uses a distance minimization objective. We name this method as the *Distance Minimization PCA for Tree-Structured Data* and abbreviate it as the *DMIN* method. Although it is called a PCA method for trees in [2], this method can also be interpreted as a feature selection method as there is no change of basis. The method selects some treelines and removes the edges that are not in the selected treelines from the trees to reduce the dimension. It seems similar to selecting a subset of relevant features and removing the remaining features from the data, as in feature selection. Note that our proposed methods are based on a similar idea, so they can also be seen as feature selection methods. The steps of DMIN method are inspired by the steps of the classical PCA in the Euclidean space. Moreover, the objective used in DMIN method is the same

as one of the alternative objectives used in the classical PCA. Therefore, the authors in [2] called DMIN method a PCA method. In this thesis, we also call our methods as PCA methods as they are based on the ideas developed in [2], and due to their similarity to the classical PCA in terms of the variance maximization objective, while we are aware of the similarities between our methods and feature selection.

In [2], an initialization step for DMIN method is given. In this step, before selecting any treeline, common edges of all trees in the tree set are put into the edge set of the principal component tree. Thus, common edges are also counted as selected on each principal component in addition to the selected treelines. It is in contrast with the idea of dimension reduction while at the same time resulting in keeping uninformative edges. For this purpose, we do not consider this additional step in our implementation of DMIN method. Note that if there is no common edge in the data set, our modification and the original method give exactly the same results.

Now, we provide the details of DMIN method. Let $S = \{t_1, t_2, \ldots, t_n\}$ be a coherent set of trees where $t_i = (N_i, E_i)$. Let ST be the corresponding support tree and $L = \{\ell_1, \ell_2, \ldots, \ell_{|LN|}\}$ be the set of all treelines in ST where LN is the set of leaf nodes of ST. The first principal component, ℓ_1^*, selected by DMIN method, is the treeline that minimizes the sum of all Hamming distances between trees and their projections onto this treeline. Mathematically, we have that

$$\ell_1^* = \arg\min_{\ell \in L} \sum_{i=1}^{n} d(t_i, P_\ell\ (t_i)). \tag{4.1}$$

For all $k \geq 2$, the $k-$th principal component, ℓ_k^*, selected by DMIN method, is the treeline that minimizes the sum of all Hamming distances between the updated trees and their projections onto ℓ_k^*. After the first k-1 principal components are computed, the tree t_i is updated as $t_i \setminus pc_{k-1}$, where pc_{k-1} is as it is defined in (3.6). In other words, to update the tree t_i, we subtract from it the edges of t_i that are among the edges of the first k-1 principal components. Therefore, selection of ℓ_k^* in DMIN method is similar to minimizing the sum of squared Euclidean distances between the updated data points and their projections onto the line spanned by the $k-$th principal component vector as in the classical PCA in the Euclidean space. In DMIN method, we can write the $k-$th principal component as

$$\ell_k^* = \underset{\ell \in L \setminus L_{k-1}^*}{\arg\min} \sum_{i=1}^{n} d((t_i \setminus pc_{k-1}) \ , \ P_\ell(t_i \setminus pc_{k-1})), \tag{4.2}$$

where L_{k-1}^* is the set of first k-1 principal components and pc_{k-1} is the principal component tree as defined in (3.6). Since all the edges in $P_\ell(t_i \setminus pc_{k-1})$ are also in $(t_i \setminus pc_{k-1})$, we have that $d((t_i \setminus pc_{k-1}) \ , \ P_\ell(t_i \setminus pc_{k-1})) = |E(t_i \setminus pc_{k-1})| - |E(P_\ell(t_i \setminus pc_{k-1}))|$. Since the term $|E(t_i \setminus pc_{k-1})|$ is independent of ℓ, we have that $\ell_k^* = \underset{\ell \in L \setminus L_{k-1}^*}{\arg\min} \sum_{i=1}^{n} (-|E(P_\ell(t_i \setminus pc_{k-1}))|)$ which is equivalent to

$$\ell_k^* = \underset{\ell \in L \setminus L_{k-1}^*}{\arg\max} \sum_{i=1}^{n} \mid (E(t_i) \cap (E(\ell) \setminus E(pc_{k-1})) \mid . \tag{4.3}$$

Note that, if we define $E(pc_0)$ and L_0^* as the empty sets, then (4.3) is applicable when $k = 1$ as well. We can rewrite (4.3) as $\ell_k^* = \underset{\ell \in L \setminus L_{k-1}^*}{\arg\max} \sum_{e \in E(\ell) \setminus E(pc_{k-1})} \sum_{i=1}^{n} \mid E_i \cap \{e\} \mid$ which can be simplified to

$$\ell_k^* = \underset{\ell \in L \setminus L_{k-1}^*}{\arg\max} \sum_{e \in E(\ell) \setminus E(pc_{k-1})} freq(e), \tag{4.4}$$

using the definition of frequency in (3.1). (4.4) shows that the $k-$th principal component is the treeline that carries most frequencies on its edges that are not in pc_{k-1}. For DMIN method, in the selection process of the $k-$th principal component, we define the score of the treeline ℓ, denoted by $s_k^{DMIN}(\ell)$, as $\sum_{e \in E(\ell) \setminus E(pc_{k-1})} freq(e)$. This means that the $k-$th principal component (after selecting the first $k-1$ principal components) is the treeline which has the largest score.

An illustrative example of applying this method on the set of trees given in Figure 3.1 is provided in Figure 4.1. At the beginning, $s_1^{DMIN}(\ell)$ values of the treelines in the left-to-right order as displayed in Figure 4.1 (i.e., from the treeline joining the root and node 8 to the treeline joining the root and node 15) are 14, 15, 17, 16, 16, 14, and 15. Since the treeline joining the root and node 10 has the maximum score, it is selected as the first principal component by DMIN method. In Figure 4.1, the first principal component is represented by the dotted treeline. After selecting the first principal component, the scores of the treelines, i.e., $s_2^{DMIN}(\ell)$ values, are calculated as 7, 8, -, 2, 16, 14, 15, respectively, where - represents a previously selected treeline. Hence, the second principal component is selected as the treeline joining the root and node 13. This treeline is represented by the dashed treeline in Figure 4.1. Note that

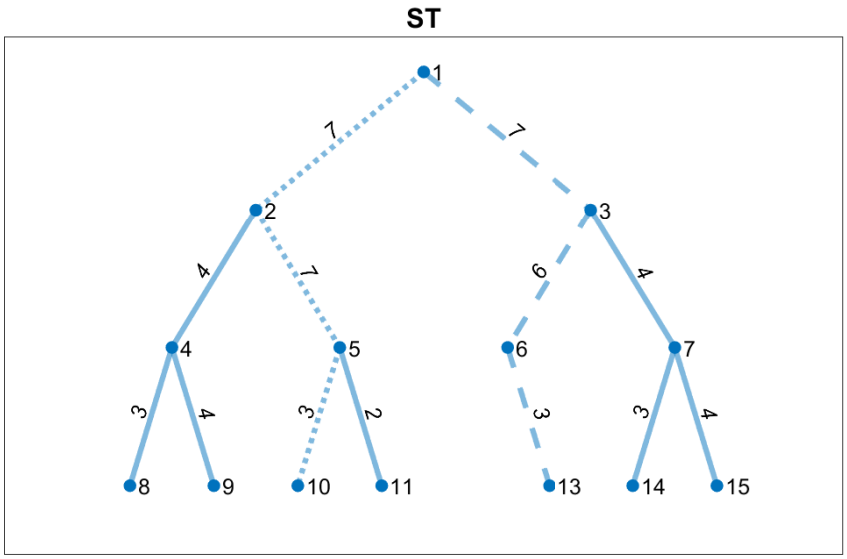

Figure 4.1: The first two principal components selected by DMIN method (dotted and dashed treelines, respectively) for the set of trees given in Figure 3.1

since there is no common edge for all trees, applying the initialization step of DMIN method would not affect the principal component trees.

The projections of the trees given in Figure 3.1 onto the first two treelines selected by DMIN method are given in Figure 4.2. Here the projection of tree t_i onto the first k principal components selected by DMIN method is represented as $P_{DMIN(k)}(t_i)$. Recall that the set of trees given in Figure 3.1 has a bi-cluster structure with the first four and the last four trees being in different clusters. Note that the existing bi-cluster structure is not visible in Figure 4.2. Even though there are many uncommon edges of t_1 and t_5, their projections have four common edges and only one uncommon edge. Moreover, although their original trees are different, projections of t_3 and t_8 are the same. DMIN cannot keep the cluster structure of the data in this example. This motivates us to develop other PCA methods to keep useful information about the data after the dimension reduction process. We also observe that (4.4) results in selecting the treeline that maximizes the total number of edges in the projections. However, this objective does not consider variance, which is the measure of information in the classical PCA. Thus, the method proposed in [2] may result in selecting the treelines that make the projections of trees similar to each other, as in Figure 4.2. Since such

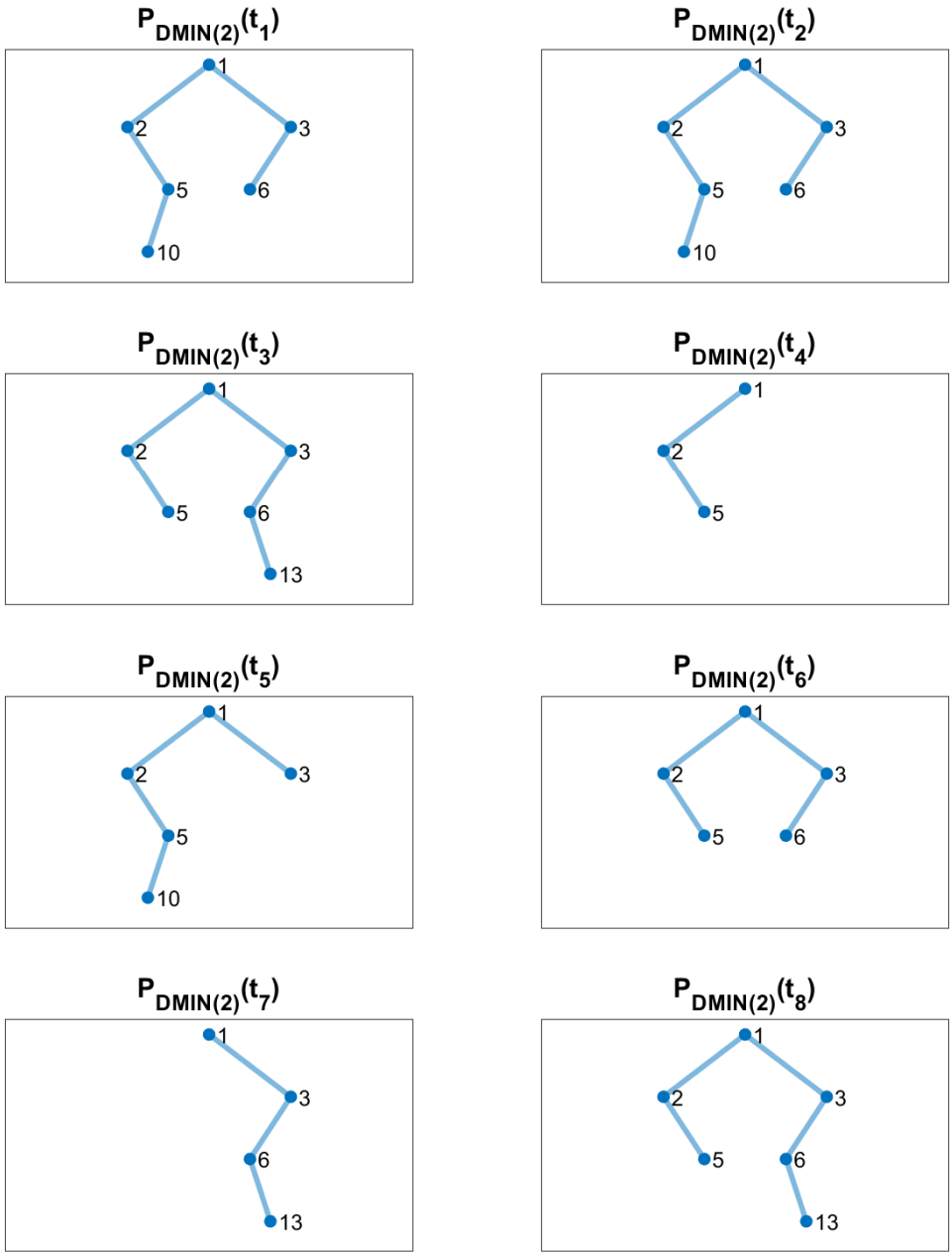

Figure 4.2: Projections of trees in Figure 3.1 onto the first two principal components selected by DMIN method

projections may not keep the individual characteristics of the trees, we propose novel methods in Section 4.3 and 4.4 to select treelines using variance-based objectives, as the variance maximization objective in the classical PCA.

4.3 Variance Maximization PCA for Tree-Structured Data

For the problem of generating principal components for tree-structured data, we propose two novel methods. The first one is named as the *Variance Maximization PCA for Tree-Structured Data* and is abbreviated as the *VMAX* method. This method is basically an extension of the classical PCA which uses a variance maximization objective. VMAX method selects the first principal component as the treeline ℓ_1^* that maximizes the variance of the lengths of the projections of the trees onto ℓ_1^*. Mathematically, we have that

$$\ell_1^* = \underset{\ell \in L}{\arg\max}\, Var(\ [\ |\ E(P_\ell(t_i))\ |\ ]_{i=1}^n\ ), \tag{4.5}$$

where $Var(\cdot)$ stands for the standard sample variance. For all $k \geq 2$, the $k-$th principal component, ℓ_k^*, selected by VMAX method, is the treeline that maximizes the variance of the lengths of the projections of the updated trees onto ℓ_k^*. Here the updated tree is used as it is explained in Section 4.2, i.e., the tree t_i is updated as $t_i \setminus pc_{k-1}$, where pc_{k-1} is as it is defined in (3.6). Note that selection of ℓ_k^* in VMAX method is similar to maximizing the variance of the lengths of the projections of the updated data points onto the line spanned by the $k-$th principal component vector in the Euclidean space. In VMAX method, the $k-$th principal component can be computed as

$$\ell_k^* = \underset{\ell \in L \setminus L_{k-1}^*}{\arg\max}\, Var(\ [\ |\ E(P_\ell(t_i \setminus pc_{k-1}))\ |\ ]_{i=1}^n), \tag{4.6}$$

where L_{k-1}^* is the set of first $k-1$ principal components. If we define $E(pc_0)$ and L_0^* as the empty sets, then (4.6) becomes applicable when $k = 1$ as well. For VMAX method, in the selection process of the $k-$th principal component, we define the score of the treeline ℓ, denoted by $s_k^{VMAX}(\ell)$, as $Var(\ [\ |\ E(P_\ell(t_i \setminus pc_{k-1}))\ |\ ]_{i=1}^n)$. This means that the $k-$th principal component (after selecting the first $k-1$ principal

components) is the treeline which has the largest score. We can rewrite (4.6) as

$$\ell_k^* = \underset{\ell \in L \setminus L_{k-1}^*}{\arg\max} \sum_{i=1}^{n} (\mid E(t_i \setminus pc_{k-1}) \cap E(\ell) \mid - \mu_\ell)^2, \tag{4.7}$$

where $\mu_\ell = \frac{1}{n} \sum_{i=1}^{n} \mid E(t_i \setminus pc_{k-1}) \cap E(\ell) \mid$ stands for the sample mean of the lengths of the projections of the updated trees onto ℓ. Note that, $(n-1)$ is removed from the denominator of (4.7) as it is a constant for all $\ell \in L$.

Observe that the objective in (4.6) is variance-based, unlike the objective of DMIN method in (4.4). VMAX method selects a treeline as the first principal component for which the variability of the projections of the trees onto this treeline is the maximum. In DMIN method, however, the treeline which has the largest sum for the lengths of the projections is selected as the first principal component. Therefore, we expect the dimension reduction performed by VMAX method to keep more useful information about the original data set than DMIN method.

The steps of VMAX method are detailed in Algorithm 1. The inputs of the algorithm are a coherent set of trees S and the number of principal components npc. The output of the algorithm is the set of principal components $\{\ell_1^*, \ell_2^*, \ldots, \ell_{npc}^*\}$. The algorithm starts with an initialization step which generates all possible principal components. First, it builds the corresponding support tree ST and its leaf edge set LE as explained in Chapter 3. It initializes the treeline set as the empty set and pc_0 as the tree having only the root but no edges. Then it generates a treeline for each leaf and puts them into the treeline set L.

After the initialization step, for all $k \in \{1, \ldots, npc\}$, the selection and update steps follow. The selection step selects the $k-$th principal component ℓ_k^* from the treeline set L that maximizes the variance of the lengths of the projections of the updated trees onto ℓ_k^* using (4.6). In the update step, L_k^* and the $k-$th principal component tree, pc_k, are constructed using L_{k-1}^* and pc_{k-1}, respectively. They are then used to update the trees. At the end of the algorithm, the set of the first npc principal components are returned. Using this set, for all $k \in \{1, \ldots, npc\}$, the projection of a tree t_i onto the first k principal components can be obtained as in (3.7).

We now explain VMAX method on an example. As inputs to the algorithm, we take $npc = 2$ and the trees given in Figure 3.1. Firstly, $s_1^{VMAX}(\ell)$ values of the treelines in

Algorithm 1: VMAX

Inputs: Coherent set of trees $S = \{t_1, t_2, \ldots, t_n\}$ where $t_i = (N_i, E_i)$ for $i = 1, \ldots, n$ and the number of principal components npc

Initialization: Generate all possible principal components.

Let ST be the support tree of S.

Let LE be the set of leaf edges of S.

Let $L = \emptyset$, $L_0^* = \emptyset$ and pc_0 be the tree where $N(pc_0)$ has only the root and $E(pc_0) = \emptyset$.

for $e \in LE$ **do**

  Let ℓ be the treeline induced by $E(\ell) = asc(e) \cup \{e\}$.

  $L = L \cup \{\ell\}$

end

for *k=1* **to** npc **do**

  Selection: Select the principal component.

  $\ell_k^* = \underset{\ell \in L \setminus L_{k-1}^*}{\arg\max}\, Var(\, [\, |\, E(P_\ell(t_i \setminus pc_{k-1}))\, |\,]_{i=1}^n)$

  Update: Construct the principal component tree.

  $pc_k = pc_{k-1} \cup \ell_k^*$

  $L_k^* = L_{k-1}^* \cup \{\ell_k^*\}$

end

return $\{\ell_1^*, \ell_2^*, \ldots, \ell_{npc}^*\}$

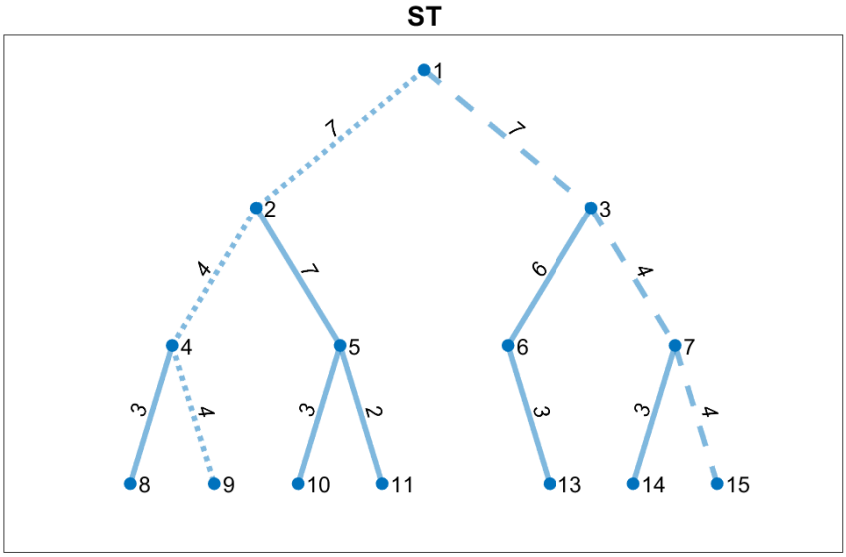

Figure 4.3: The first two principal components selected by VMAX method (dotted and dashed treelines, respectively) for the set of trees given in Figure 3.1

the left-to-right order as displayed in Figure 4.3 (i.e., from the treeline joining the root and node 8 to the treeline joining the root and node 15) are evaluated as 1.36, 1.55, 0.98, 0.86, 1.14, 1.36, 1.55. Since the maximum score is shared with two treelines, one of them is chosen arbitrarily. In this case, we choose the treeline joining the root and node 9 as the first principal component, which is represented by the dotted treeline in Figure 4.3. After selecting the first principal component, $s_2^{VMAX}(\ell)$ values are calculated as 0.27, -, 0.5, 0.41, 1.14, 1.36, and 1.55, respectively, where - represents a previously selected treeline. Thus, the treeline joining the root and node 15 is selected as the second principal component. This treeline is represented by the dashed treeline in Figure 4.3.

In Figure 4.4, the projections of the trees given in Figure 3.1 onto the first two principal components selected by VMAX method are given. Here the projection of tree t_i onto the first k principal components selected by VMAX method is represented as $P_{VMAX(k)}(t_i)$. Observe that the first four projections and the last four have different structures where the projections of the trees of the first cluster (i.e., the first four trees) have longer paths on the left side while the projections of the other trees have longer paths on the right side. For both clusters, within similarity of clusters is high. Even

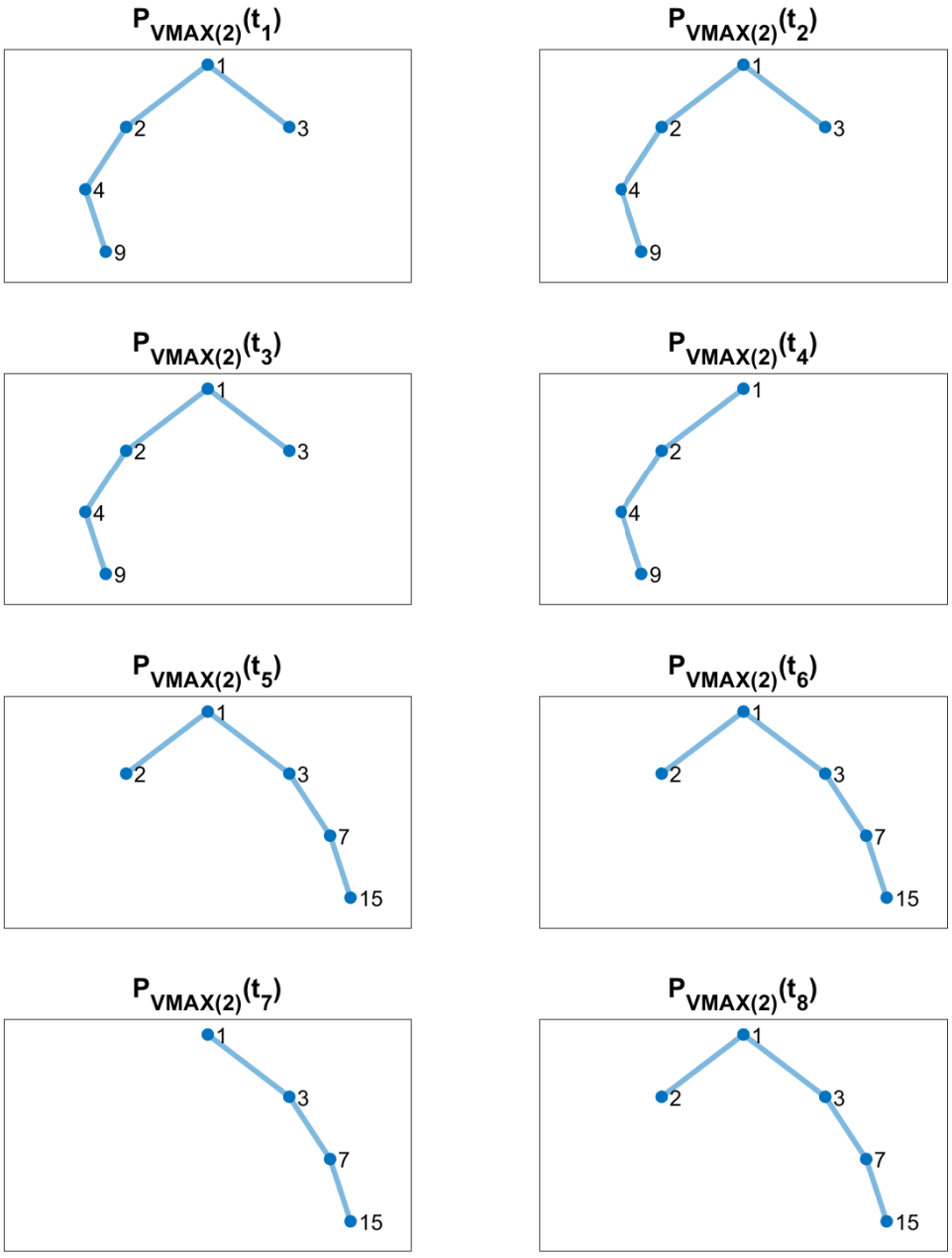

Figure 4.4: Projections of trees in Figure 3.1 onto the first two principal components selected by VMAX method

three of the four projections are the same in both clusters. For this instance, VMAX method seems to be more successful in keeping the cluster structure of the data than DMIN method. In order to broaden our understanding of the performances of different PCA methods for tree-structured data, we carry out extensive computational experiments using different data sets in Chapter 5.

4.4 Similarity Minimization PCA for Tree-Structured Data

Another variance-based objective function can be constructed using the fact that the standard sample variance in the Euclidean space is equivalent to the sum of squares of all pairwise distances between the data points divided by the square of the number of data points. Note that by this equivalence, $k-$th principal component vector selection of classical PCA in the Euclidean space can be performed by taking the vector v that maximizes the sum of squares of all pairwise distances between the projections of data points onto the union of first $k-1$ principal component vectors and v.

Inspired by the fact that maximizing the distances is equivalent to minimizing the similarities, we develop another PCA method for tree-structured data that minimizes the sum of all pairwise similarities of the trees, a variance-based objective function. We name this method it as *Similarity Minimization PCA for Tree-Structured Data* ($SMIN$) method. SMIN method selects the first principal component as the treeline ℓ_1^* that minimizes the pairwise similarities of the projections of the trees onto ℓ_1^*. Mathematically, we have that

$$\ell_1^* = \arg\min_{\ell \in L} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} s(P_\ell(t_i), P_\ell(t_j)). \tag{4.8}$$

For all $k \geq 2$, the $k-$th principal component, ℓ_k^*, selected by SMIN method, is the treeline that minimizes the pairwise similarities of the projections of the trees onto $pc_{k-1} \cup \ell_k^*$ where pc_{k-1} is used as defined in (3.6). Note that selection of ℓ_k^* in SMIN method is similar to maximizing the sum of squares of all pairwise distances between the projections of data points onto the union of the first $k-1$ principal component vectors and $k-$th principal component vector in the Euclidean space. In SMIN method,

the $k-$th principal component can be computed as

$$\ell_k^* = \underset{\ell \in L \setminus L_{k-1}^*}{\arg\min} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} s(\ t_i \cap (pc_{k-1} \cup \ell)\ ,\ (t_j \cap (pc_{k-1} \cup \ell))\ . \tag{4.9}$$

For SMIN method, the score of the treeline while selecting the $k-$th principal component is denoted by $s_k^{SMIN}(\ell)$ and is defined as $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (s\ (\ t_i \cap (pc_{k-1} \cup \ell)\ ,\ (t_j \cap (pc_{k-1} \cup \ell)))$. This means that the $k-$th principal component (after selecting the first $k-1$ principal components) is the treeline which has the smallest score. Observe that, the objective in (4.9) is variance-based as in (4.6), unlikely to the objective of DMIN method in (4.4).

The steps of SMIN method are detailed in Algorithm 2. The inputs of the algorithm are a coherent set of trees S and the number of principal components npc. The output of the algorithm is the set of principal components $\{\ell_1^*, \ell_2^*, \ldots, \ell_{npc}^*\}$. The algorithm starts with an initialization step which generates all possible principal components. First, it builds the corresponding support tree ST and its leaf edge set LE as explained in Chapter 3. It initializes the treeline set as the empty set and pc_0 as the tree having only the root but no edges. It constructs the edge existence matrix $D = [D_{ij}]$, where D_{ij} takes the value 1 if and only if $j-$th edge on ST (in any order), denoted by e_j, is on t_i. Then the algorithm generates a treeline for each leaf and puts it into the treeline set L.

After the initialization step, for all $k \in \{1, \ldots, npc\}$, the selection and update steps follow. The selection step calculates the scores of all treelines in the treeline set. To compute all pairwise similarities of trees for all k we use matrices. For any treeline, ℓ, firstly, the matrix R is constructed as the edge existence matrix of $\ell \cup pc_{k-1}$. Then, the matrix $I = [I_{im}]$ is obtained by matrix multiplication of R and the transpose of R where I_{im} represents the number of edges in $P_{\ell \cup pc_{k-1}}(t_i) \cap P_{\ell \cup pc_{k-1}}(t_m)$. If two trees, t_i and t_m, have the edge $e_j \in E(\ell \cup pc_{k-1})$, then $R_{ij} = R_{mj} = 1$ and RR^T would increment the value of I_{im} by 1. Otherwise, since at least one of their values in the matrix R would be 0, the value of I_{im} would not increase. Moreover, the diagonal values of the matrix represent the number of edges on the trees. Now all the terms that are needed to calculate the similarity of any two trees are obtained by constructing the matrix I. The similarity matrix $A = [A_{im}]$ can be constructed as $A_{im} = (2\ I_{im} + 1)/(2\ (I_{ii} + I_{mm}) + 2)$. Then, the sum of all A_{im}s is recorded

Algorithm 2: SMIN

Inputs: Coherent set of trees $S = \{t_1, t_2, \ldots, t_n\}$ where $t_i = (N_i, E_i)$ for $i = 1, \ldots, n$ and the number of principal components npc.

Initialization: Generate all possible principal components.

Let ST be the support tree of S and e_j be the j-th edge on ST for $j \in \{1, 2, \ldots, |E(ST)|\}$.

Let LE be the set of leaf edges of S.

Let $L = \emptyset$, $L_0^* = \emptyset$ and pc_0 be the tree where $N(pc_0)$ has only the root and $E(pc_0) = \emptyset$.

Let D be a matrix where $D_{ij} = 1$ if $e_j \in E_i$, 0 otherwise.

for $e \in LE$ **do**

    Let ℓ be the treeline induced by $E(\ell) = asc(e) \cup \{e\}$.

    $L = L \cup \{\ell\}$

end

for *k=1* **to** npc **do**

    Selection: Select the principal component.

    for $\ell \in L \setminus L_{k-1}^*$ **do**

        Let R be the matrix D whose j-th vector is removed if $e_j \notin E(\ell \cup pc_{k-1})$.

        $I = RR^T$

        Construct the matrix A where $A_{im} = (2\ I_{im} + 1)/(2\ (I_{ii} + I_{mm}) + 2)$.

        $s_k^{SMIN}(\ell) = \sum_{i=1}^{n-1} \sum_{m=i+1}^{n} A_{im}$

    end

    $\ell^* = \underset{\ell \in L \setminus L_{k-1}^*}{\arg\min} s_k^{SMIN}(\ell)$

    Update: Construct the principal component tree and update the score array and set of treelines.

    $pc_k = pc_{k-1} \cup \ell_k^*$

    $L_k^* = L_{k-1}^* \cup \{\ell_k^*\}$

end

return $\{\ell_1^*, \ell_2^*, \ldots, \ell_{npc}^*\}$

on $s_k^{SMIN}(\ell)$ to keep the score of the treeline ℓ. The selection step selects the $k-$th principal component ℓ_k^* from the treeline set $L \setminus L_{k-1}^*$ that has the smallest score. In the update step, the $k-$th principal component tree is constructed as the union of the previous principal component tree, pc_{k-1}, and the last principal component, ℓ_k^*. The $k-$th principal component tree is then used to construct the matrix R, and the L_k^* is constructed using the lastly selected treeline.

At the end of the algorithm, the set of the first npc principal components are returned. Using this set, for all $k \in \{1, \ldots, npc\}$, the projection of a tree t_i onto the first k principal components can be obtained using (3.3).

We now explain Algorithm 2 on an example. As inputs to the algorithm, we take $npc = 2$ and the set of trees given in Figure 3.1. At the beginning, the edge existence matrix D is constructed as

$$D = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

where rows indicate the trees while the columns are for edges. Since there are eight trees in the tree set and 13 edges of the support tree, the size of matrix D is 8×13. This matrix is permanent, but the other matrices are computed for all principal components and each treeline. Therefore, we show the matrices R, I, and A for the treeline joining the root and node 9 when $k = 1$. The matrix R, i.e., the existence of the edges on pc_k tree if the corresponding treeline would be selected as the $k-$th principal component,

is constructed as

$$R = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

whose columns are 1^{st}, 3^{rd}, and 7^{th} columns of the matrix D where the edges of the treeline joining the root and node 9 are the 1^{st}, 3^{rd}, and 7^{th} edges of the support tree. The matrix I, which shows the number of common edges of the projections of trees onto $\ell \cup pc_{k-1}$, is constructed as RR^T which is given below

$$I = \begin{bmatrix} 3 & 2 & 3 & 3 & 1 & 1 & 0 & 1 \\ 2 & 2 & 2 & 2 & 1 & 1 & 0 & 1 \\ 3 & 2 & 3 & 3 & 1 & 1 & 0 & 1 \\ 3 & 2 & 3 & 3 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

Using matrix I, the matrix A that consists of pairwise similarities of the projections of trees onto $\ell \cup pc_{k-1}$ can be constructed as

$$A = \begin{bmatrix} 0.5 & 0.42 & 0.5 & 0.5 & 0.3 & 0.3 & 0.13 & 0.3 \\ 0.42 & 0.5 & 0.42 & 0.42 & 0.38 & 0.38 & 0.17 & 0.38 \\ 0.5 & 0.42 & 0.5 & 0.5 & 0.3 & 0.3 & 0.13 & 0.3 \\ 0.5 & 0.42 & 0.5 & 0.5 & 0.3 & 0.3 & 0.13 & 0.3 \\ 0.3 & 0.38 & 0.3 & 0.3 & 0.5 & 0.5 & 0.25 & 0.5 \\ 0.3 & 0.38 & 0.3 & 0.3 & 0.5 & 0.5 & 0.25 & 0.5 \\ 0.13 & 0.17 & 0.13 & 0.13 & 0.25 & 0.25 & 0.5 & 0.25 \\ 0.3 & 0.38 & 0.3 & 0.3 & 0.5 & 0.5 & 0.25 & 0.5 \end{bmatrix}.$$

The score of the treeline joining the root and node 9 can be computed by summing all elements on the upper diagonal matrix of A, which is 9.37. If $s_1^{SMIN}(\ell)$ values of

all treelines are calculated in the same way, then their scores are obtained in the left-to-right order as displayed in Figure 4.3 (i.e., from the treeline joining the root and node 8 to the treeline joining the root and node 15) as 9.37, 9.35, 10.54, 10.75, 9.90, 9.37, and 9.35. Since the minimum score is shared with two treelines, one of them is chosen arbitrarily. In this case, we choose the treeline joining the root and node 9. After the selection of the first principal component, the $s_2^{SMIN}(\ell)$ are evaluated as 12.75, -, 11.90, 12.14, 9.90, 9.37, and 9.35, respectively, where - represents a previously selected treeline. Then the treeline joining the root and node 15 is selected as the second principal component since it has the smallest score.

Observe that the first two principal components selected by VMAX and SMIN methods are the same. Hence, it can be concluded that SMIN method is also successful in keeping the information for this instance. In order to broaden our understanding of the difference between their performances, we carry out detailed computational experiments in Chapter 5.

CHAPTER 5

COMPUTATIONAL EXPERIMENTS FOR PCA METHODS FOR TREE-STRUCTURED DATA

In this chapter, we test the performances of the proposed PCA methods (VMAX and SMIN), and the PCA method of Aydin et al. [2] (DMIN) for tree-structured data on synthetic and real data sets. In our computational experiments, for each method, we generate one projected data set for each number of principal components, i.e., if the number of principal components is specified as npc, then there are npc many projected data sets for each method. Then, the projected data sets are inputted into a clustering algorithm to obtain the partitions of the projected trees into the clusters (generated partitions). We calculate the similarities between the original partitions of the trees and the generated partitions of the projected trees. Note that the original partitions of the trees are known. If two partitions are similar, it can be concluded that the projected tree set carries a good amount of information about the original tree set. Therefore, the similarity scores show how much the method keeps the useful information about the data during the dimension reduction process.

The tree k-means algorithm that is proposed by Dinler et al. [5] is used to cluster the trees. It is inspired by the classical k-means clustering algorithm in the Euclidean space and developed specifically for tree-structured data. In the initialization step, as in the *standard k-means algorithm* proposed in [27], the tree k-means algorithm assigns the trees randomly to clusters. It is assumed that the number of clusters is known a priori. After that, there are assignment and update steps that process iteratively. Each tree is assigned to the closest cluster in the assignment step, and centroid trees of the clusters are recalculated in the update step. The algorithm stops when there is no change in the assignments of the trees. In [5], it is shown that using *vertex*

/ edge overlap to measure the proximity of the trees gives better solutions than using Hamming distance as the distance measure or using the classical k-means/k-modes algorithms after representing each tree as a vector in the Euclidean space. Therefore, in the computational experiments to compare the PCA methods for tree-structured data, the *tree k-means* algorithm with *vertex / edge overlap* is used. In the rest of the thesis, we refer to this algorithm as tkm^{VEO}.

To measure the similarities between two partitions of n trees into k clusters, adjusted Rand index (ARI) is used [6]. Mathematically, it is equivalent to

$$Adj.\ Rand\ Index = \frac{\sum_{i=1}^{k}\sum_{j=1}^{k}\binom{p_{ij}}{2} - \frac{\sum_{i=1}^{k}\binom{p_{i\cdot}}{2}\sum_{j=1}^{k}\binom{p_{i\cdot}}{2}}{\binom{n}{2}}}{\frac{1}{2}(\sum_{i=1}^{k}\binom{p_{i\cdot}}{2} + \sum_{j=1}^{k}\binom{p_{\cdot j}}{2}) - \frac{\sum_{i=1}^{k}\binom{p_{i\cdot}}{2}\sum_{j=1}^{k}\binom{p_{\cdot j}}{2}}{\binom{n}{2}}}. \tag{5.1}$$

Here $p_{i\cdot}$, $p_{\cdot j}$, and p_{ij} represent the number of trees in cluster i in the original partition, the number of trees in cluster j in the generated partition, and the number of trees in cluster i in the original partition and in cluster j in the generated partition, respectively.

The PCA methods for tree-structured data, RTC and tkm^{VEO} are coded in MATLAB R2020b. They are run on an Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz, 8GB RAM Windows 10 PC. Computation times are measured by CPU time. The details of the computational experiments on synthetic data are given in Section 5.1 and on real data in Section 5.2.

5.1 The Performances of PCA Methods on Synthetic Data Sets

In order to conduct computational experiments on synthetic data, a random tree set generator is developed. This generator is able to generate sets of trees with different *skewness*, *density*, and *height* properties. We use this generator to create synthetic data sets as explained in 5.1.1.

5.1.1 A Random Tree Generator

To generate random rooted labeled tree sets, we propose a novel algorithm and name it as *Random Tree Constructor* (RTC) algorithm. It simply starts with constructing a complete tree which is named as the *generator tree*, denoted by GT, whose edges are then used to induce the trees, and then computes the number of edges for each tree. For each tree, it constructs an edge set which is a subset of $E(GT)$ and includes a predetermined number of edges. At the end, trees are constructed as trees induced by the corresponding edge set. The steps of the RTC algorithm are given in Algorithm 3.

There are six input parameters for the RTC algorithm. The first one is m, limiting the maximum number of children of any node on a tree, e.g., if $m = 2$, then the generated trees would be binary trees. n specifies the number of trees in the generated set. h represents the maximum height of a generated tree. d and s are the mean and the standard deviation of the normal distribution, which is used to compute the number of edges in the trees. Lastly, the skewness factor, f, determines the dispersion of the edges. The output of the algorithm is a generated coherent set of trees. The algorithm starts with the initialization step where the generator tree GT is built as the complete m-ary tree with height h.

After the initialization step, the algorithm generates n many random integers by rounding a sample of n normally distributed random numbers with mean $d\ |E(GT)|$ and standard deviation $s\ |E(GT)|$. Note that the number of edges on a tree can be calculated as a product of the number of edges on the generator tree and a ratio between 0 and 1. This ratio shows the fullness of the complete tree, and it is named as the *density* in the RTC algorithm.

The construction step starts with initializing each edge set as the empty set. After that, it selects a predetermined number of edges from $E(GT)$ for all trees. Note that an edge cannot be added if its parent edge is not in the edge set. Selection of the edges starts with initializing the sequence of the possible edges, PE, as the edges of GT that joins the root and its m children, in the left-to-right order. Then the iterative edge selection process starts with assigning probabilities to the edges where the sum

Algorithm 3: RTC

Input: The limit of the number of children m, the number of trees n,
the height of the generator tree h,
the density distribution parameters d and s, and the skewness factor f.
Initialization: Build the generator tree.
Let GT be the generator tree as the complete $m-$ary tree with height h.
Determination: Compute the number of edges for each tree.
for *i=1* **to** *n* **do**
    Assign c_i as the rounded value of a number randomly generated with $N(\, d\,|E(GT)|\,,\ (s\,|E(GT)|)^2\,)$
end
Construction: Construct the trees.
Let $E_i = \emptyset$ be the edge set of $i-$th tree for $i = 1, \ldots, n$.
for *i=1* **to** *n* **do**
    Let PE be the sequence of the possible edges where for $k = 1, \ldots, m$.
    for *j=1* **to** c_i **do**
        Let p_k be the probabilty of selecting the $k-$th edge on PE where
        $\sum_{k=1}^{|PE|} p_k = 1$ and $p_k = f p_{k-1}$ for $k = 2, \ldots, |PE|$.
        Using p_ks select a random integer between 1 and $|PE|$
        and assign k^* as this integer.
        $E_i = E_i \cup \{PE_{k^*}\}$
        if $ch(PE_{k^*}) \neq \emptyset$ **then**
            Assign $PE_{k^*+m}, \ldots, PE_{|PE|+m-1}$ as $PE_{k^*+1}, \ldots, PE_{|PE|}$.
            Assign $PE_{k^*}, \ldots, PE_{k^*+m-1}$ as the children set of selected edge $ch(PE_{k^*})$ in the left-to-right order.
        else
            Remove PE_{k^*} from PE.
        end
    end
    Construct t_i as the tree that is induced by E_i.
end
return $\{t_1, t_2, \ldots, t_n\}$

of all probabilities is equal to 1. The probability of the $k-$th edge on PE, denoted by p_k, is equal to fp_{k-1}, for $k \geq 1$ values. If the skewness factor is less than one, then the trees tend to be *left-skewed*, and if it is more than one, then the trees tend to be *right-skewed*. Hence, the skewness factor shows the dispersion of the edges on a tree, and it is referred to as *skewness* in the RTC algorithm. Using p_ks, the edge from PE is randomly selected and added to the corresponding set of edges. In order to update the sequence of the possible edges, if the selected edge is not a leaf edge on GT, then its children edges are added to PE in the left-to-right order, and following edges on PE are shifted, otherwise only the selected edge is removed from PE. At the end of the iterative edge selection, the corresponding tree is induced by the edge set.

By controlling the parameters of the RTC algorithm, one can create trees having different properties. Thus, the performances of the methods given in Chapter 4 can be tested on these controllable randomly generated tree sets. In Section 5.1.2, we design different computational experiments, test the performances of the methods on them, and give the computational results and interpretations of the results.

5.1.2 Computational Experiments on Randomly Generated Tree Sets

Data sets with two clusters are considered for testing the performances of the methods. For each data set, two groups of trees are generated by the RTC algorithm. Hence, each tree on a data set belongs to one of the two groups, and thus we have the original partition of the trees. For constructing the data sets, the input parameter set of the RTC algorithm is constructed as follows. For each group of trees, m and n are taken as 2 and 100, respectively, i.e., there are 100+100=200 binary trees on each data set. h is considered as 5 or 10, and the standard deviation of the normal distribution, s, is taken as 0.01 and 0.001 with respect to the value of h. The *density* levels are considered as a number between 0 and 1, and the *skewness* levels are taken as one of the following values: 0.25 (*Too Left*), 0.5 (*Left*), 1 (*Uniform*), 2 (*Right*), and 4 (*Too Right*). One can generate a data set with two clusters by inputting different parameter sets for two groups of trees. Therefore, different types of instances can be generated by changing the parameters.

In this chapter, we generate 42 instances by systematically changing the *density*, *skew-*

ness, and *height* parameters to analyze the performances of the methods on data sets having different cluster structures. To account for randomness, ten replications are made for each instance.

Each data set is inputted to VMAX, SMIN, and DMIN methods. For the data sets having small and big trees, the number of principal components is given as 32 and 30, respectively. Then each projected data set is inputted into tkm^{VEO}. Since this algorithm is dependent on the initialization, for each projected data set, tkm^{VEO} is run for ten initializations, and the best solution is reported as the generated partition of the projected data set. Note that in tkm^{VEO}, each initialization is constructed with a randomly selected centroid tree which is obtained by using the Bernoulli distribution where the probability of an edge is obtained by dividing the frequency of the edge by the number of trees. The generated partitions and the original partitions are compared using the ARI, and the average ARI scores out of ten replications are presented as the performances of the methods on an instance for a specific number of principal components.

As an example, the performance results of the methods for a given number of principal components (PCs) in an instance are given in Table 5.1. This instance is generated with the following parameters. For both of the groups of trees, the height of the trees h and the skewness level are fixed as 5 and *Too Left*, respectively. In each data set, there are two clusters in different densities, and the pair of density levels are considered as (0.99-0.95), where the first number refers to the density level in the first group of trees, and the latter is for the second group.

Since h is fixed as 5 in the instance, the results of all possible principal components are given in Table 5.1. This is because the complete binary tree with $h = 5$ has 32 leaves. Therefore, when the number of principal components is 32, the ARI scores of all methods show the performance of the clustering algorithm on the original data set, i.e., the detectability of the cluster structure in the original data set. Observe that these ARI scores of each method are the same, 0.70. Therefore for this instance, it can be concluded that the cluster structure in the data sets is hard to detect. However, it can be seen in Table 5.1 that after 4 principal components, the ARI scores of both proposed methods (VMAX and SMIN) are more than 0.6, while the ARI score of the

Table 5.1: The adjusted Rand index scores of the PCA methods for tree-structured data for the instance where $h = 5$, skewness: *Too Left* for all trees, and the pair of density levels: (0.99-0.95)

# of PCs	**VMAX**	**SMIN**	**DMIN**	**# of PCs**	**VMAX**	**SMIN**	**DMIN**
1	0.38	0.38	0.00	17	0.70	0.65	0.42
2	0.52	0.55	0.00	18	0.66	0.70	0.42
3	0.55	0.55	0.00	19	0.71	0.66	0.42
4	0.61	0.61	0.00	20	0.71	0.71	0.42
5	0.65	0.65	0.00	21	0.70	0.70	0.42
6	0.65	0.67	0.00	22	0.70	0.70	0.42
7	0.65	0.62	0.00	23	0.70	0.70	0.42
8	0.67	0.65	0.03	24	0.70	0.71	0.42
9	0.70	0.66	0.03	25	0.70	0.70	0.42
10	0.65	0.65	0.03	26	0.70	0.70	0.42
11	0.65	0.73	0.03	27	0.70	0.70	0.46
12	0.70	0.71	0.03	28	0.71	0.70	0.46
13	0.70	0.70	0.03	29	0.70	0.70	0.46
14	0.70	0.70	0.03	30	0.70	0.70	0.46
15	0.64	0.67	0.03	31	0.70	0.70	0.43
16	0.70	0.70	0.42	32	0.70	0.70	0.70

existing method (DMIN) is almost zero for the first 15 principal components. Hence, for this instance, it can be concluded that the performances of the proposed methods are better than the existing method, and they are successful in keeping the useful information.

To investigate the effects of the *density*, *skewness*, and *height* on the performances of the methods, 42 instances are grouped as four patterns. These patterns are

1. Density-Different Clusters on Small Trees ($h = 5$),
2. Density-Different Clusters on Big Trees ($h = 10$),
3. Skewness-Different Clusters on Small Trees ($h = 5$),
4. Skewness-Different Clusters on Big Trees ($h = 10$).

In the first two patterns, different density levels for two groups of trees are considered while fixing the skewness parameter. We aim to analyze the performances of the methods on density-different clustered data sets on these patterns. In the last two patterns, different skewness levels are considered while the density levels are fixed to discover the performances of the methods on skewness-different clustered data sets. Since we aim to discover the effect of the sizes of the trees on the performances, the trees in the first and the third patterns are small (limited with height 5), while the trees are bigger ($h = 10$) in the second and the fourth patterns.

In Sections 5.1.2.1 - 5.1.2.4, each of the four patterns is explained in detail and the corresponding results are presented in Figures 5.1 - 5.4, respectively. These figures include panels where each one of them presents the results on an instance. In these figures, all panels that are in the same row are for the same skewness levels, e.g., in Figure 5.1, all top three panels presents the results of the instances with *Too Left* skewness level, and these levels are written in bold on the left-hand side of the figures. Similarly, all panels that are in the same column are for the same density levels, e.g., all three panels in the left column in Figure 5.1 presents the results of the instances with the pairs of density levels (0.99-0.95). In these panels, the x-axis and y-axis represent the number of principal components and adjusted Rand index, respectively. The solid, dotted, and dashed lines show the performances of VMAX, SMIN, and

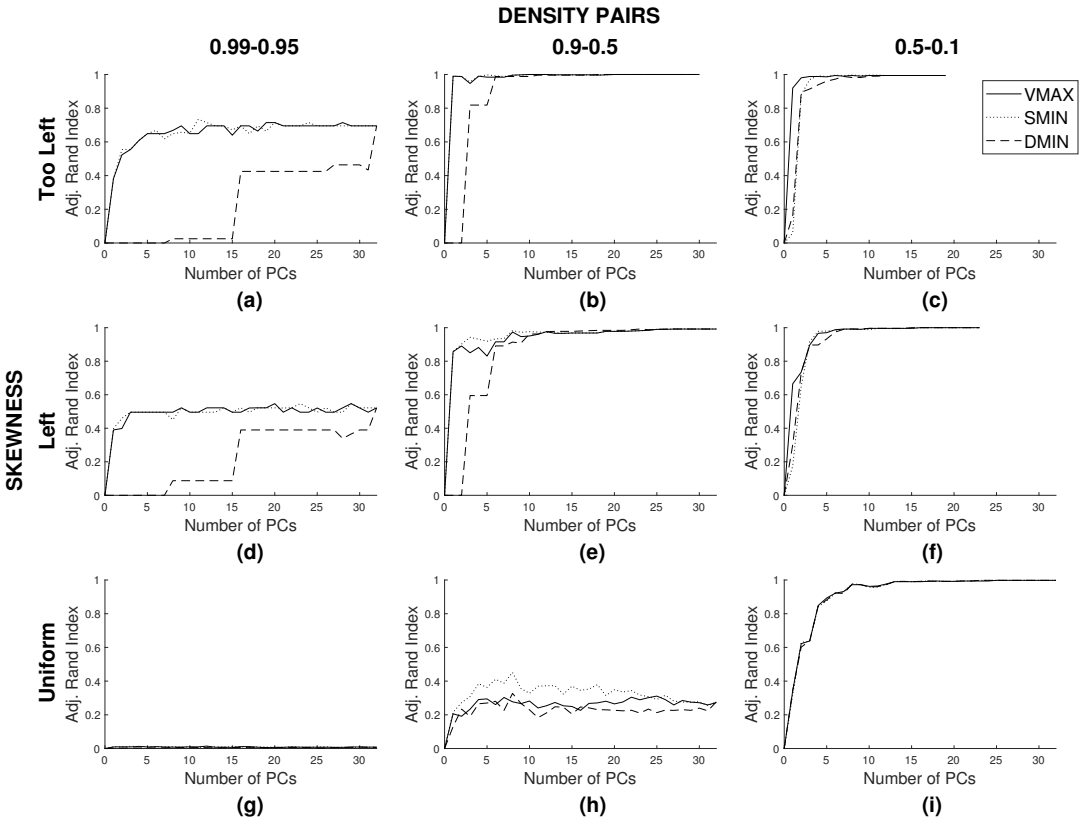

Figure 5.1: Clustering performances of the PCA methods for tree-structured data on density-different data sets with $h = 5$

DMIN methods, respectively.

5.1.2.1 PCA Methods on Density-Different Clusters on Small Trees

In this set of experiments, the maximum possible height of a tree is limited to 5. For both of the groups, the skewness level is fixed as *Too Left*, *Left*, and *Uniform*. In each data set, there are two groups in different densities, and the pairs of density levels are taken as (0.99-0.95), (0.9-0.5), and (0.5-0.1). We obtain 9 instances using the combinations of these parameters. As explained in 5.1.2, the ARI scores of the methods are obtained, and they are given in Figure 5.1.

In the instances with *Uniform* skewness level, there is no cluster structure (see Figure 5.1-(g),(h),(i)). It is because the number of common edges within a cluster is not enough to make trees from the same group similar. In these data sets, the ARI scores

of the methods are negligible, as expected. Figure 5.1-(i) is an exception since in this instance, the ratio of the density levels (0.5/0.1=5) is extremely high. In this instance, all methods are successful in leading high clustering accuracies using about 5 principal components.

In the instances with skewness and (0.99-0.95) density level, there is an unclear cluster structure in the data sets as they are presented in Figure 5.1-(a),(d). In these instances, while VMAX and SMIN generate projected data sets that can be well clustered using less than 5 principal components, DMIN method needs to use 16 principal components to get a similar result. Observe that for 16 principal components, still the ARI score of DMIN method is lower than the ARI scores of the proposed methods. The successful performances of the proposed methods are due to the way they select the treelines as their principal components. Since the principal component selections in the proposed methods are variance-based, the trees from the group generated with 0.99 density level have all edges of their principal component, but some of the trees from the group generated with 0.95 density level do not have some edges of this principal component. Hence, the projections of the trees from different groups of trees become different from each other, and it will be easy to cluster the projected trees. However, DMIN method selects its principal component such that all trees from both groups have all edges of this principal component. This is because of the objective of DMIN method, which is maximizing the total number of edges on the projections of trees.

There is a clear cluster structure in the instances with *Too Left* and *Left* skewness levels and (0.9-0.5) and (0.5-01) pairs of density levels (see Figure 5.1-(b),(c),(e),(f)). In these instances, projected trees generated by each of the methods are successfully clustered using a few principal components. Still, there is a difference between the ARI scores of proposed methods and DMIN, but it is not as significant as in the data sets having unclear cluster structure. It is because, in the data sets having a cluster structure, DMIN method selects the treelines that are common in all trees, which results in no variety in the projections of the trees. Nevertheless, there are only a few common treelines in the data sets having a clear cluster structure. Hence, the ARI score of DMIN method reaches the ARI scores of VMAX and SMIN methods using less than 10 principal components.

Therefore, it can be concluded that the proposed methods perform better than DMIN method in revealing (if any) the cluster structure of the density-different cluster structured data sets having small trees using fewer principal components. However, it becomes more challenging to make this conclusion as the cluster structure becomes clearer.

5.1.2.2 PCA Methods on Density-Different Clusters on Big Trees

Again, here 9 instances are obtained using the following parameters. The height of trees is up to ten ($h = 10$) in this set of experiments. For both groups in the data sets, the skewness level is fixed as *Too Left*, *Left*, and *Uniform*. The pairs of density levels are considered as (0.099-0.095), (0.09-0.05), and (0.05-0.01). Here the maximum number of principal components (30) is less than the number of treelines on the support tree of the data sets. Thus, the ARI score for the highest principal component does not show the clustering performance of the original data, but this value can give an idea about the detectability of the cluster structure in the data set. The ARI scores of the methods are computed with a similar procedure as in Section 5.1.2 which are presented in Figure 5.2.

For Figure 5.2, similar comments made for Figure 5.1 can be made because the presented results are similar. Therefore, it can be concluded that for density-different data sets (having small or big trees), the amount of the useful information carried by the principal components is higher on the proposed methods than the existing method. Note that there is no significant difference in the performances of VMAX and SMIN methods on these data sets.

For a data set having density-different clusters, the running time of methods to generate the projected trees for a given number of principal components is less than 3 seconds. However, the computation time for tkm^{VEO} is significant, and it increases with the number of edges on the trees. For example, for the instance with $h = 10$, fixed skewness level *Left* and the pair of density levels (0.09-0.05), tkm^{VEO} consumes 105 seconds on average to cluster the original data, while it needs only 3.8, 4.6, and 7.0 seconds in average to cluster the projected data sets generated by VMAX, SMIN, DMIN, respectively, using 10 principal components. It concludes that clustering the

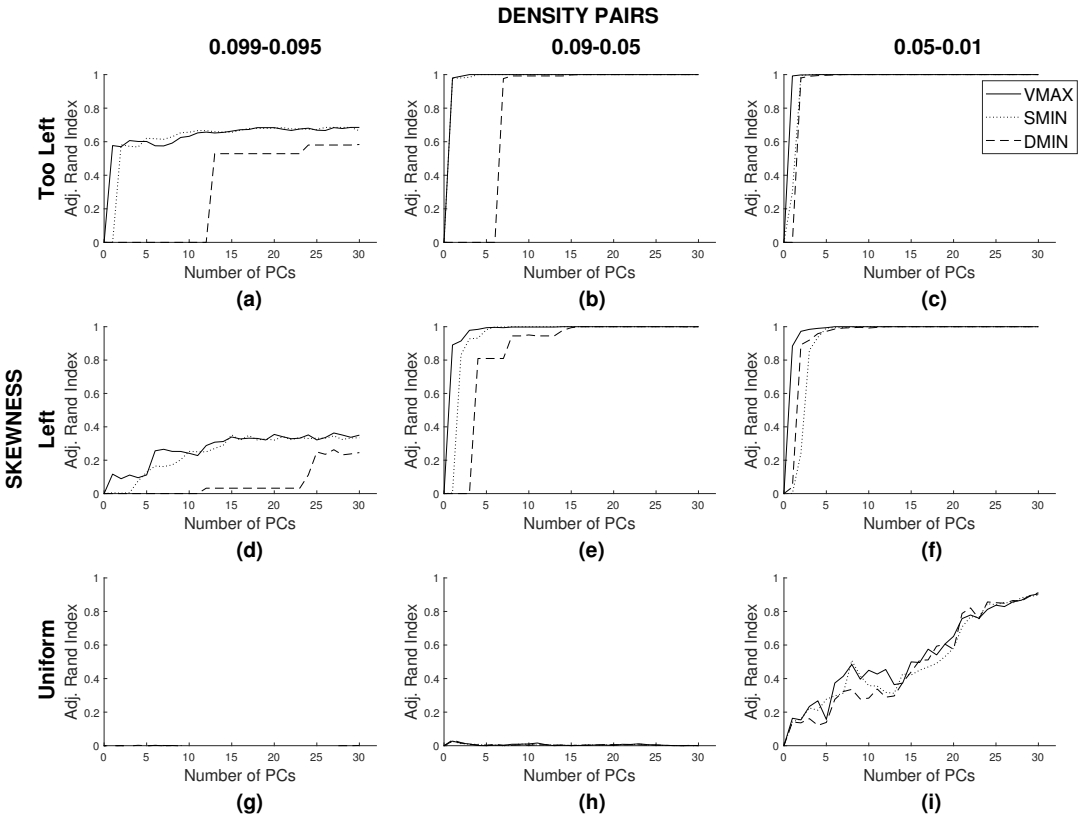

Figure 5.2: Clustering performances of the PCA methods for tree-structured data on density-different data sets with $h = 10$

data set through trees generated by the methods decreases computation time for clustering without sacrificing the quality of cluster structure.

5.1.2.3 PCA Methods on Skewness-Different Clusters on Small Trees

In this set of experiments, 12 instances are generated with the following parameter settings. h is taken as 5 and the density level is fixed as 0.7 and 0.3 for both of the groups in a data set. The pairs of skewness levels are considered as (*Left-Uniform*), (*Left-Right*), (*Too Left-Left*), (*Too Left-Uniform*), (*Too Left-Right*), and (*Too Left-Too Right*). The ARI scores of the methods are computed with a similar procedure as in Section 5.1.2 which are displayed in Figure 5.3.

In the instances with close skewness levels, there is no cluster structure (see Figure 5.3-(e),(f)). It is because the number of common edges between two groups of trees are so close to the number of common edges within the groups of trees. As expected, the ARI scores of the methods are insignificant in these instances.

There is an unclear cluster structure in the instances with (*Left-Uniform*) and (*Too Left-Uniform*) pairs of skewness levels (see Figure 5.3-(a),(b),(g),(h)). It is because some of the trees from the group with *Uniform* skewness level may be *left-skewed*, and separating them from the other group of trees is difficult. In these instances, using only 2 principal components, the proposed methods reach the ARI score of the clustering performance of the original data. However, DMIN method needs more than 5 principal components to obtain a similar result. This difference between the performances is more significant in the instances with 0.7 density level than in the ones with 0.3 (see Figure 5.3-(a),(g)). When using less than 15 principal components, DMIN method fails in generating projected data to be clustered as the original data.

In the remaining instances, a clear cluster structure exist as it can be seen in Figure 5.3-(c),(d),(i),(j),(k),(l). All methods generate projected data sets that can be well clustered using a few principal components. However, in the instances with 0.7 density level, while 1 principal component is sufficient for VMAX and SMIN methods to reach the clustering performance of the original data, DMIN needs about 5 principal components since it initially selects the common treelines of the trees. This difference

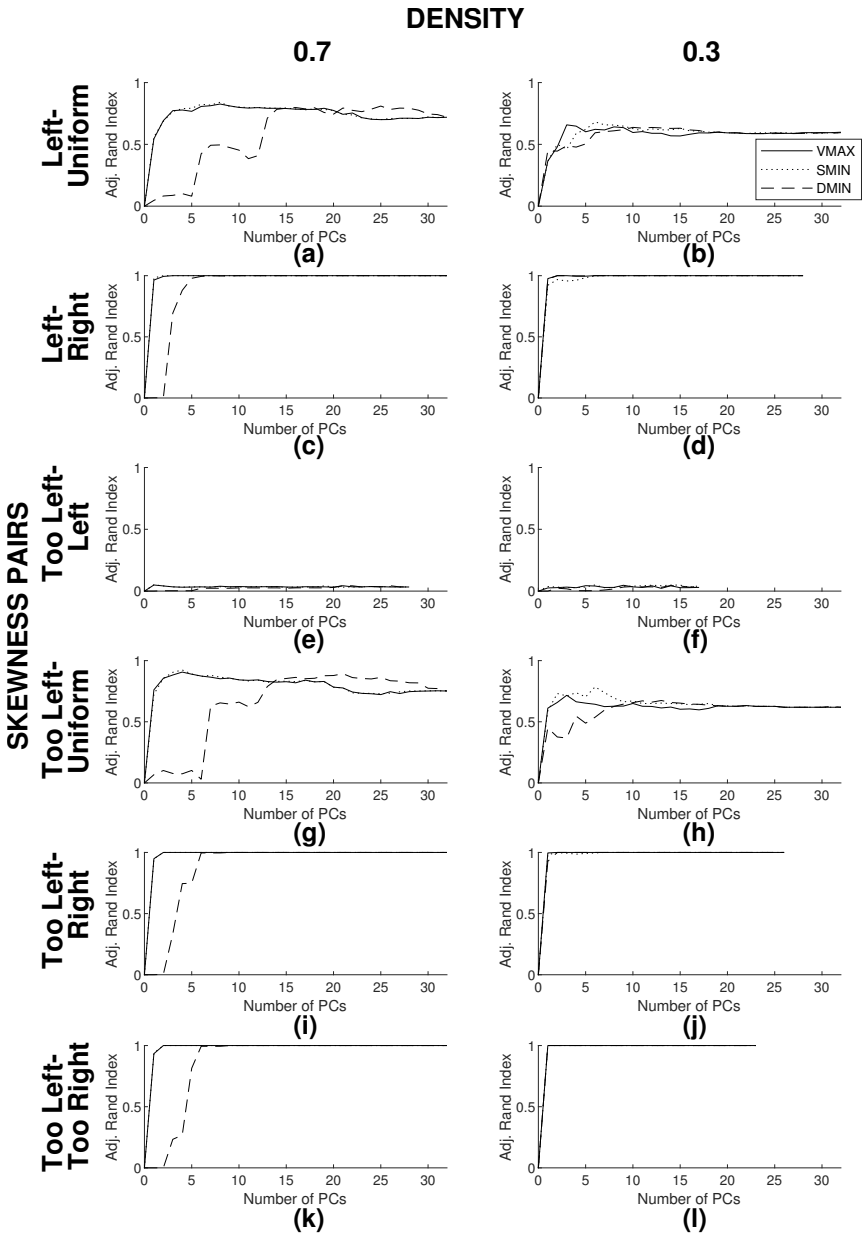

Figure 5.3: Clustering performances of the PCA methods for tree-structured data on skewness-different data sets with $h = 5$

is not observed in the instances with 0.3 density level because there is no common treeline in these instances.

It can be concluded that if there is a skewness-different cluster structure in a data set having small trees, then the principal components selected by the proposed methods are much useful than the ones selected by the existing method. As the cluster structure becomes clearer, i.e., the skewness levels of the sets of trees differ from each other or their density level decreases, the gap between the methods gets narrow.

5.1.2.4 PCA Methods on Skewness-Different Clusters on Big Trees

Again, here 12 instances are obtained using the following parameters. The height of trees is up to ten ($h = 10$), and the density level is fixed as 0.3 and 0.07. The pairs of skewness levels are considered the same as in Section 5.1.2.3. Here the number of treelines on the support tree of the data sets are more than the maximum number of principal components (30), as in 5.1.2.2. The ARI scores of the methods are computed with a similar procedure as in Section 5.1.2 which are presented in Figure 5.4.

For Figure 5.4, similar comments made for Figure 5.3 can be made because their results are similar (see Figures 5.3 and 5.4). Therefore it can be concluded that the proposed methods perform better in keeping the useful information in skewness-different data sets (having small or big trees) than the existing method. As the cluster structure becomes clearer, the significance of this difference between the performances disappears. Note that there is no remarkable difference in the performances of VMAX and SMIN methods on the skewness-different data sets.

For a data set having skewness-different clusters, the running time of methods to generate the projected trees for a given number of principal components is less than 3 seconds. However, the computation time consumed by tkm^{VEO} is remarkable, and it increases with the sizes of the trees. For example, for the instance with $h = 10$, fixed density level 0.3 and the pair of skewness levels (*Left-Right*), tkm^{VEO} needs more than 15000 seconds on average to cluster the original data, while it consumes only 3.97, 4.11, and 4.37 seconds in average to cluster the projected data sets generated by VMAX, SMIN, DMIN, respectively, using 5 principal components. It can be con-

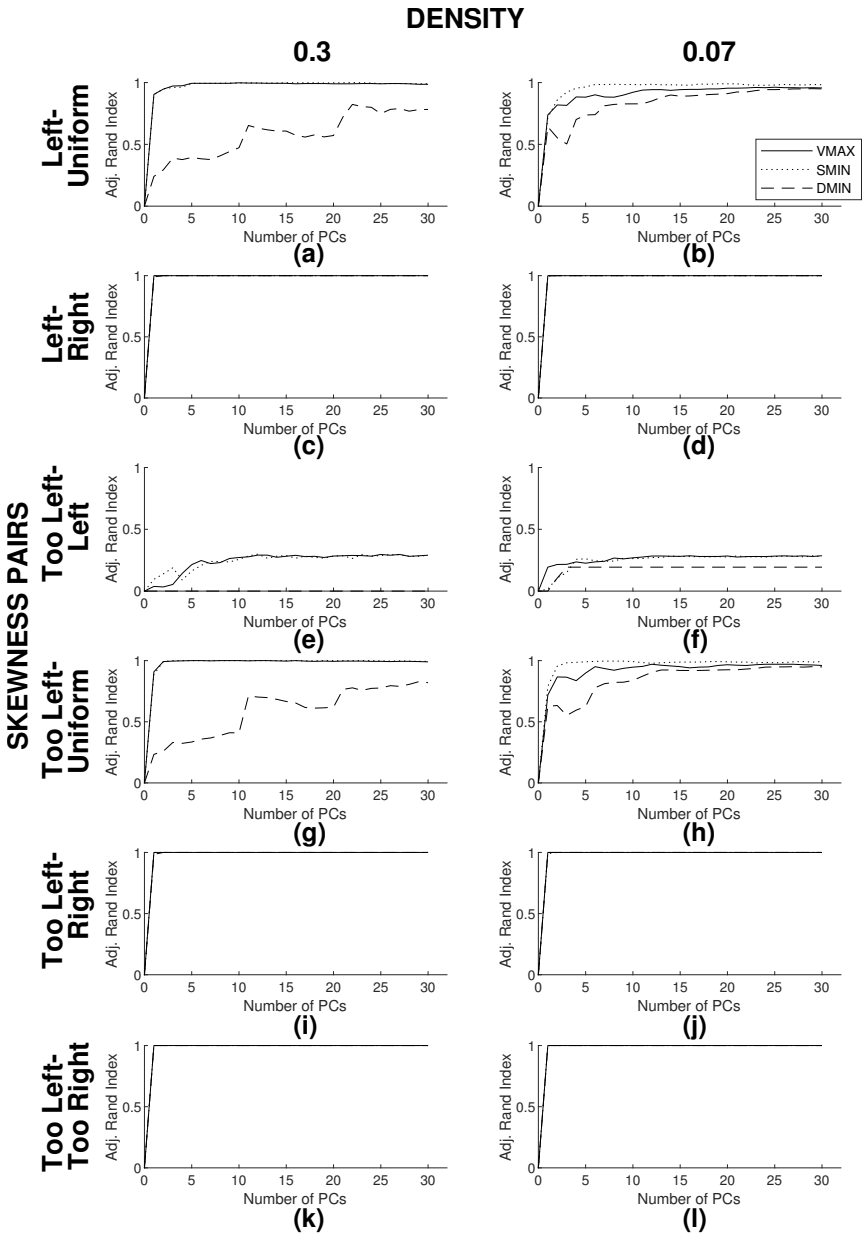

Figure 5.4: Clustering performances of the PCA methods for tree-structured data on skewness-different data sets with $h = 10$

cluded that clustering the data set through trees generated by the methods decreases the clustering time without sacrificing the quality of cluster structure.

All in all, it can be concluded that if a (density or skewness different) cluster structure exists in a data set, projected data sets generated by each of the methods can be successfully clustered using smaller trees and less computation time. In such data sets, the performances of the proposed methods are better than the performance of the existing method. Also, the ARI scores of the proposed methods are not worse than the ARI score of the existing method, in any case. As the cluster structure become clearer, the difference between the performances of the methods decreases. No significant difference is observed between the performances of VMAX and SMIN methods in synthetic data sets.

5.2 The Performances of PCA Methods on a Real Data Set

In this section, we test the performances of the methods on real data. The set of brain artery structures of 98 individuals, shared by Bendich et al. [28] is used as the real-world data here. Descriptive features sex, age, and handedness for each patient are also given in the data set. As a preprocessing, we remove transgender and ambidextrous individuals as is done by Dinler et al. [5], and work with 93 individuals. As discussed by Aydin et al. [2], for each brain artery structure, a rooted binary tree is constructed by representing the branching points as nodes and the vessels as edges. In these trees, only the vessels whose median radius is more than 0.5 mm are represented. The labels of the nodes are obtained as follows: the child with more descendant nodes is placed as the left child, and for a given node, all left children get the same label in all trees.

In this set of trees, the maximum height of a tree is 25, where the average is 15.27. The number of edges in the support tree is 942, there are 451 leaves, and the height is 25. The average number of edges is 95.65, where the maximum is 188. The average density of trees is almost 0.000003, and thereby the density level of the set is more sparse than the generated data sets in Section 5.1.

In Figure 5.5, the support tree of the tree set is given. Since the nodes with more

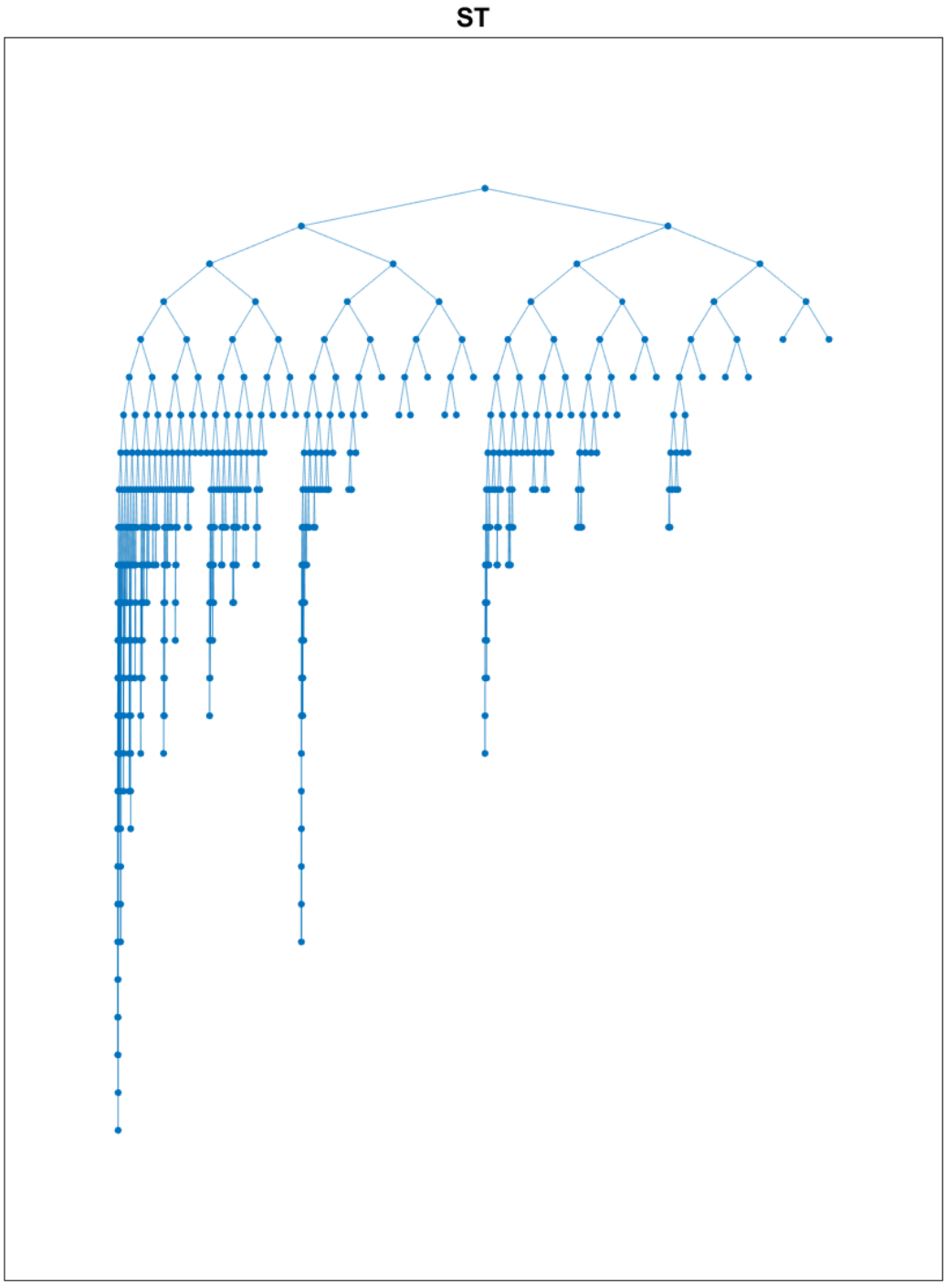

Figure 5.5: The support tree of the real data set

descendants are placed on the left, the support tree is *left-skewed*. Computational results of the PCA methods for tree-structured data on the brain artery data set are presented in the following section.

5.2.1 Computational Experiments on Real Data

Original partitions of the trees in simulated data sets are known as they are constructed with given input parameters. Therefore, it is possible to compare the generated partitions of the projected trees with the original partitions. However, for the real data set introduced in Section 5.2, the clustering partitions of the original trees are unknown. Moreover, the number of clusters in the data set is unknown, and even there may exist no cluster structure in the data set. Hence, first, we analyze the cluster structure in the brain artery data set. In order to do that, from 2 to 6 clusters, we cluster the data set with tkm^{VEO}. For each number of clusters between 2 and 6, 100 initializations are made, and the best objective function value is considered. For detecting the right number of clusters, we use the well-known elbow method and try to observe the biggest change in the objective function value. From 2 to 6 cluster case, the objective function values are 29.38, 30.25, 30.96, 31.53, and 31.99, respectively. Since the objective function value follows a linear increase with the number of clusters, we accept that the right number of clusters in the brain artery data set as 2. Later, the data set is clustered with the number of clusters 2, and the clustering result is accepted as the original partitions.

To monitor the performance of all methods on the real data, we follow the same procedure as in Section 5.1.2, with a slight change that generated partitions are obtained with 100 initializations on tkm^{VEO} instead of 10. We first use VMAX, SMIN, and DMIN to generate projected trees for a given number of principal components. Then the projected trees are clustered into 2 clusters by using tkm^{VEO}. Again the clustering algorithm is run for 100 initializations. ARI score for a generated partition is reported considering the original partitions of the trees in the brain artery data set.

In Figure 5.6, the first three principal components of VMAX, SMIN, and DMIN are shown. The first principal component selected by DMIN is the leftmost treeline in the figure and shown by the dashed line. The first principal component selected by

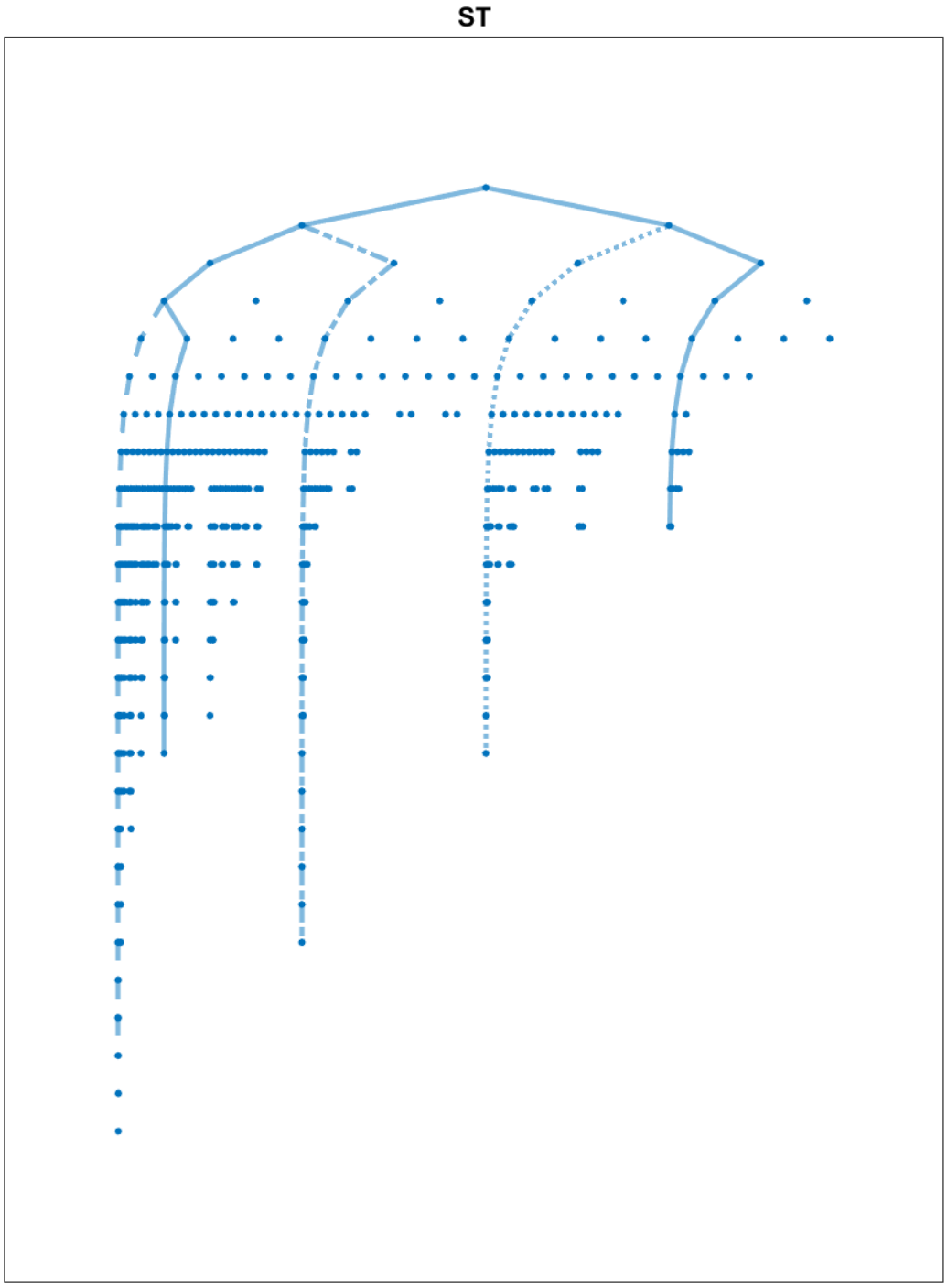

Figure 5.6: Principal components of the PCA methods for tree-structured on the support tree of the real data set

VMAX and SMIN is the same, and it is shown by the dotted line. This treeline is also the second principal component selected by DMIN method. Moreover, the second principal component selected by the proposed methods is the first principal component selected by DMIN. Therefore, the first two principal components constructed by the methods are the same, just except the order. Hence we expect that the clustering performances of all three methods are the same on two principal components. The third principal component selected by DMIN is the solid line on the left-hand side. VMAX selects the treeline that is represented by the dash-dotted line as its third principal component. The treeline represented by the rightmost solid line in Figure 5.6 is selected as the third principal component by SMIN.

The ARI scores of VMAX, SMIN, and DMIN on the brain artery data set are given in Figure 5.7. Here each method is presented with a different type of line the solid line for VMAX, the dotted line for SMIN, and the dashed line for DMIN. In this plot, the results of the first 50 principal components for each method are shown. The ARI scores of all methods are above 0.7, for 7 and more principal components. In addition, all principal component trees of the three methods have less than only 110 edges for the first 7 principal components, while the whole support tree has 930 edges. Hence, it can be concluded that the projected trees generated by any of the methods can be successfully clustered using about 12 percent of the total number of edges on the support tree.

Since the first principal components are the most informative ones in PCA and after 7 principal components, all methods successfully keep the cluster structure of the data set, we analyze the first 6 principal components of the methods. For the first principal component, while the clustering of the projected trees generated by DMIN is not successful with 0.15 ARI score, clustering of projected trees generated by VMAX and SMIN has a higher ARI score of 0.72. This big gap between the methods shows that our proposed methods overperform the existing method in the literature on selecting the first principal component and the first principal component of the proposed methods is more informative so that the cluster structure is easily be detected via this principal component. For 2 principal components, the ARI scores of all methods are 0.76, which is better than the one in the first principal component as expected. Again, as in the classical PCA method, the information gain with the second principal com-

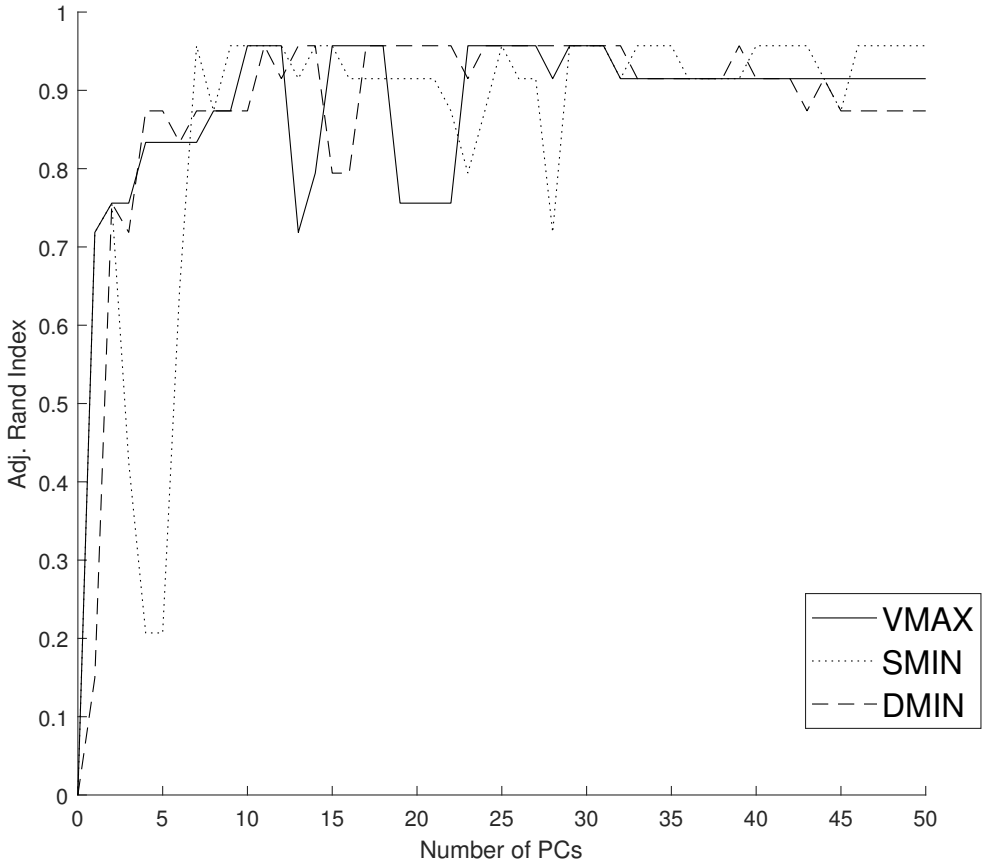

Figure 5.7: Clustering performances of the PCA methods for tree-structured data on the real data set

ponent in the proposed methods is gradual. But differently in DMIN second principal component is more informative than the first one.

While the performances of SMIN and DMIN decrease in the third principal component, the performance of VMAX remains the same. For the 4^{th} to 6^{th} principal components, both ARI scores of VMAX and DMIN are more than 0.83. However, the ARI score of SMIN is less than 0.7 on these principal components. The numbers of edges on the 7^{th} principal component tree constructed by VMAX, SMIN, and DMIN are 103, 88, and 105, respectively. Note that 7 principal components are sufficient for all methods to reveal the cluster structure.

It can be concluded that all three methods are successful in keeping the clustering information on the projected data, while our proposed methods perform better than the existing method in selecting the first principal component, and only 1 treeline is sufficient for VMAX to extract the cluster structure of the real data.

CHAPTER 6

MULTIDIMENSIONAL SCALING FOR TREE-STRUCTURED DATA

6.1 Multidimensional Scaling

For given pairwise distances between data objects and a user defined positive integer t, the multidimensional scaling (MDS) maps the data objects into data points in $\mathbb{R}^t$ in such a way that the pairwise distances between the generated data points are as close as possible to the original pairwise distances. MDS can be used to visualize the data objects or to project high-dimensional data onto a lower dimension. There are different methods to perform the MDS. While classical MDS [10] uses eigenvectors to extract the coordinate matrix, formulating an optimization problem to find the optimal coordinates of the data objects is also commonly used in the literature, see e.g, the generalized MDS [12]. There are different objective functions used in these optimization models for MDS. One of the most commonly used objective functions is minimizing the sum of squares of pairwise differences between the given distances and the distances between the generated data points.

MDS for tree-structured data is a less studied area. Therefore, inspired by the classical MDS, we develop the concepts of MDS for tree-structured data. We assume that we are given a coherent set of trees. In our implementation of the MDS for tree-structured data, we consider the edges as the dimensions. We select some edges from the support tree and call the tree that is induced by the selected edges the scaling tree. We project all trees onto the scaling tree in such a way that the pairwise distances between the projections are proportionally close to the original pairwise distances between the trees.

Since MDS for tree-structured data *selects* the edges to be on the projected trees, it

can be interpreted as a feature selection method. Note that the user defines the number of edges on the scaling tree and thereby limits the number of edges in the projected trees. The objective function aims to keep as much information as possible about the original tree-structured data utilizing only k edges in the scaling tree. In the rest of the chapter, we detail our proposed MDS definition for tree-structured data and propose exact and heuristic methods to perform it.

6.2 MDS for Tree-Structured Data - Problem Definition

We are given a set of coherent trees $S = \{t_1, t_2, \ldots, t_n\}$, where $t_i = (N_i, E_i)$, and a user defined positive integer k. Let ST be the corresponding support tree. We aim to select k edges from ST in such a way that the graph induced by the selected edges forms a subtree of ST with the same root. This subtree is called as the scaling tree and is denoted by sc_k. In our implementation of the MDS, we try to find the best sc_k such that the original pairwise distances and the distances between the projections of the trees onto sc_k, i.e., the distances between the trees $P_k(t_i) = t_i \cap sc_k$, are as close as possible. For the distance measure, we use the Hamming distance. Note that for two projections $P_k(t_i)$ and $P_k(t_j)$, the Hamming distance between them can be at most k. Therefore, if $d(t_i, t_j)$ is much bigger than k, then there would be no way to keep the pairwise distances close after the projection. For this reason, instead of keeping the pairwise distances close, we aim to keep them proportionally close.

We are now ready to provide the optimization problem we define to be able to perform MDS for tree-structured data. Given a set of coherent trees $S = \{t_1, t_2, \ldots, t_n\}$ and a positive integer k, find a scaling tree sc_k with $|sc_k| = k$ and a scaling parameter $\alpha \geq 0$ such that

$$z = \sum_{i=1}^{n-1} \sum_{m=i+1}^{n} \mid \alpha \, d(t_i, t_m) - d(P_k(t_i), P_k(t_m)) \mid \tag{6.1}$$

is minimized. Here the first term inside the absolute value $\alpha \, d(t_i, t_m)$ scales original pairwise distances. The second term $d(P_k(t_i), P_k(t_m))$ is the distances between the projections of the trees. An alternative way to perform the MDS would be scaling

the distances between the projections instead of scaling the original distances. In this case, the objective function would be

$$z' = \sum_{i=1}^{n-1} \sum_{m=i+1}^{n} \mid d(t_i, t_m) - \alpha d(P_k(t_i), P_k(t_m)) \mid . \tag{6.2}$$

In this thesis, we study the first alternative, i.e., the objective function (6.1), while we have observed after some computational experiments that the second alternative may be more useful in some cases. We next provide a mixed-integer programming formulation followed by two greedy heuristics to minimize (6.1).

6.3 Multidimensional Scaling for Tree-Structured Data - Optimization Model

For the problem of performing MDS for tree-structured data as explained in Section 6.2, we propose an exact method and name it as *Multidimensional Scaling for Tree-Structured Data - Optimization Model* (*MDST*) method. It is basically an extension of the classical MDS algorithm, which selects the edges to be on the scaling tree while keeping the pairwise distances between trees proportionally close. Let $S = \{t_1, t_2, \ldots, t_n\}$ be a coherent set of trees, where $t_i = (N_i, E_i)$. ST be the corresponding support tree, and k be a user defined positive integer. The edge selection process of MDST is formulated as a mathematical model, which can be written in closed form as follows.

(MDST closed form)

$$\text{Minimize } z = \sum_{i=1}^{n-1} \sum_{m=i+1}^{n} \mid \alpha \; d(t_i, t_m) - d((t_i \cap sc_k), (t_m \cap sc_k)) \mid \tag{6.3}$$

subject to

$$\mid E(sc_k) \mid = k \tag{6.4}$$

$$asc(e) \in E(sc_k) \qquad \forall e \in sc_k \tag{6.5}$$

$$\alpha \geq 0 \tag{6.6}$$

where the objective function (6.3) is equivalent to (6.1). The scaling parameter is not limited by an upper bound in this model but it is expected to take a value between zero and one. The first constraint of the model (6.4) assigns the number of edges on the scaling tree to the user defined number k. Since all combinations of selected edges may not induce a tree, we write the second constraint of the model which provides that an edge can be selected only if its ascending edges are selected.

Let $D = [D_{ij}]$ be the matrix where D_{ij} takes the value 1 if $i-$th tree has the $j-$th edge of ST (in any order) and 0 otherwise. If we define x_j as the decision variable which takes the value 1 if and only if $j-$th edge of ST is on the scaling tree sc_k, then the open form of the mathematical model of MDST can be written as

(MDST open form)

$$\text{Minimize } z = \sum_{i=1}^{n-1} \sum_{m=i+1}^{n} \mid \alpha \, ((\sum_{j=1}^{|E(ST)|} D_{ij} + D_{mj}) - 2 \, (\sum_{j=1}^{|E(ST)|} D_{ij} \, D_{mj}))$$
$$- ((\sum_{j=1}^{|E(ST)|} (D_{ij} + D_{mj}) \, x_j) - 2 \, (\sum_{j=1}^{|E(ST)|} D_{ij} \, D_{mj} \, x_j)) \mid \tag{6.7}$$

subject to

$$\sum_{j=1}^{|E(ST)|} x_j = k \tag{6.8}$$

$$x_j \leq x_{par(j)} \qquad \forall j \in E' \tag{6.9}$$

$$\alpha \geq 0 \tag{6.10}$$

$$x_j \in \{0, 1\} \qquad \forall j \in \{1, \ldots, |E(ST)|\}. \tag{6.11}$$

Here the number of edges in the union and intersection of two trees t_i and t_j are computed as $\sum_{j=1}^{|E(ST)|} D_{ij} + D_{mj}$ and $\sum_{j=1}^{|E(ST)|} (D_{ij} \, D_{mj})$, respectively. Note that if we multiply the summed terms with x_j, then they would be equal to the union and intersection of the projections of t_i and t_j on sc_k. The first constraint (6.8) is for assigning the number of edges on the scaling tree. The second constraint (6.9) is for keeping the tree-structure of the scaling tree, where $par(j)$ is the index of the parent edge of $j-$th edge, and E' is defined as the set of all indices of the edges in ST that have a parent edge. (6.9) provides that an edge cannot be on the scaling tree if its parent is not on it. This constraint also satisfies that an edge cannot be on the scaling

tree if its all ascending edges are not on it. (6.10) is the same as (6.6), and (6.11) is written since the x_j is a binary variable for any value of j.

Since this mathematical model can easily be linearized, we obtain a mixed-integer linear programming (MILP) model. We define a set of decision variables, d_{im}, to represent the absolute difference between $\alpha d(t_i, t_m)$ and $d(P_k(t_i), P_k(t_m))$ to linearize the model. Here the objective function value of the MILP model is the sum of d_{im}s for all pairs of trees. We add two constraint sets (6.13) and (6.14) to compute d_{im} values. The resulting MILP model of MDST method is given below.

(MDST MILP)

$$\text{Minimize } z = \sum_{i=1}^{n-1} \sum_{m=i+1}^{n} d_{im} \tag{6.12}$$

subject to

$$d_{im} \geq \alpha \left(\left(\sum_{j=1}^{|E(ST)|} D_{ij} + D_{mj}\right) - 2 \left(\sum_{j=1}^{|E(ST)|} D_{ij}\, D_{mj}\right)\right)$$
$$- \left(\left(\sum_{j=1}^{|E(ST)|} (D_{ij} + D_{mj})\, x_j\right) - 2 \left(\sum_{j=1}^{|E(ST)|} D_{ij}\, D_{mj}\, x_j\right)\right)$$
$$\forall i \in \{1, \ldots, n-1\} \;\&\; \forall m \in \{i+1, \ldots, n\} \tag{6.13}$$

$$d_{im} \geq - \left[\alpha \left(\left(\sum_{j=1}^{|E(ST)|} D_{ij} + D_{mj}\right) - 2 \left(\sum_{j=1}^{|E(ST)|} D_{ij}\, D_{mj}\right)\right)\right.$$
$$\left. -\left(\left(\sum_{j=1}^{|E(ST)|} (D_{ij} + D_{mj})\, x_j\right) - 2 \left(\sum_{j=1}^{|E(ST)|} D_{ij}\, D_{mj}\, x_j\right)\right)\right]$$
$$\forall i \in \{1, \ldots, n-1\} \;\&\; \forall m \in \{i+1, \ldots, n\} \tag{6.14}$$

$$\sum_{j=1}^{|E(ST)|} x_j = k \tag{6.15}$$

$$x_j \leq x_{par(j)} \qquad \forall j \in E' \tag{6.16}$$

$$\alpha \geq 0 \tag{6.17}$$

$$x_j \in \{0, 1\} \qquad \forall j \in \{1, \ldots, |E(ST)|\} \tag{6.18}$$

The MILP model of MDST is explained on an example. For the set of trees given in Figure 3.1, the model is run for $k \in \{1, \ldots, 13\}$ values. Note that the matrix D is the

Table 6.1: The results of MDST model on the set of trees given in Figure 3.1

k	**Edges of** sc_k	α	z	z'
1	2	0	7	—
2	2,4	0.13	9.1	70
3	3,7,14	0.25	11.3	45.2
4	2,3,7,14	0.29	11.6	40
5	2,3,4,7,14	0.38	13.9	36.6
6	2,3,4,6,7,13	0.43	12.9	30
7	2,3,6,7,13,14,15	0.50	12.5	25
8	2,3,4,5,7,8,10,11	0.56	12.9	23
9	2,3,4,5,7,8,10,11,14	0.67	12	17.9
10	2,3,4,5,6,7,8,10,11,14	0.71	12.4	17.5
11	2,3,4,5,6,7,8,9,10,11,14	0.82	11	13.4
12	2,3,4,5,6,7,8,9,10,11,13,14	0.89	6.6	7.4
13	2,3,4,5,6,7,8,9,10,11,13,14,15	1	0	0

same as the matrix D defined in Section 4.4. The results are shown in Table 6.1. The first column is for the k values given to the model that assigns the number of edges on the scaling tree. In the second column, an edge on the scaling tree that joins a node u with its parent v is represented by u, e.g., "5" represents the edge joining node 5 with node 2, which is the parent of node 5. The results of the scaling parameter and the objective function values are given in the third and fourth columns, respectively. In the last column, objective function values of (6.2), z', are given, where $z' = z/\alpha$. Note that in terms of z', these solutions are not optimal.

From Table 6.1, it can be concluded that α values tend to increase as the number of edges on the scaling tree increases, as expected. z' values decrease as the number of edges on the scaling tree increases. Note that, for $k \geq 2$, sc_{k-1} does not have to be a subtree of sc_k, e.g., sc_2 has the edge joining the root and node 2 while sc_3 does not have it.

As illustrative examples, the scaling tree with 5 edges, sc_5, and with 8 edges, sc_8, constructed by MDST method are shown in Figures 6.1 and 6.2, respectively. In

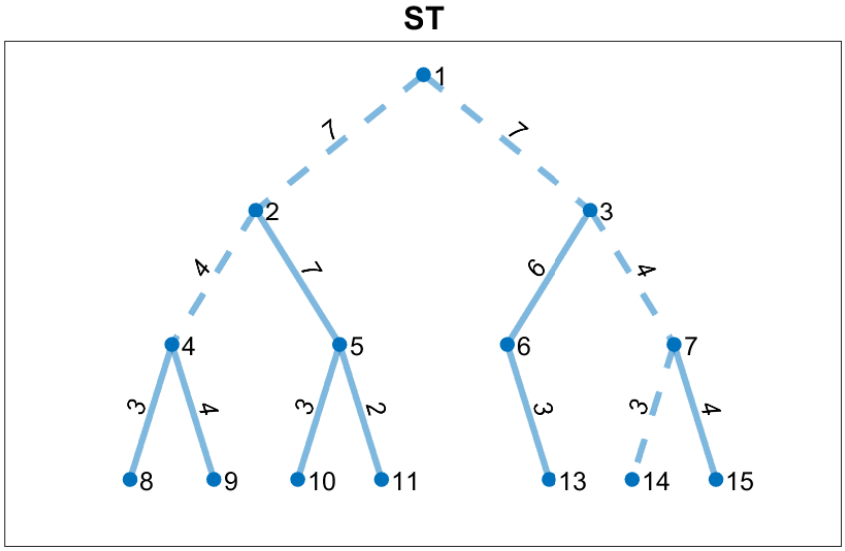

Figure 6.1: The scaling tree with 5 edges constructed by MDST method (dashed edges) for the set of trees given in Figure 3.1

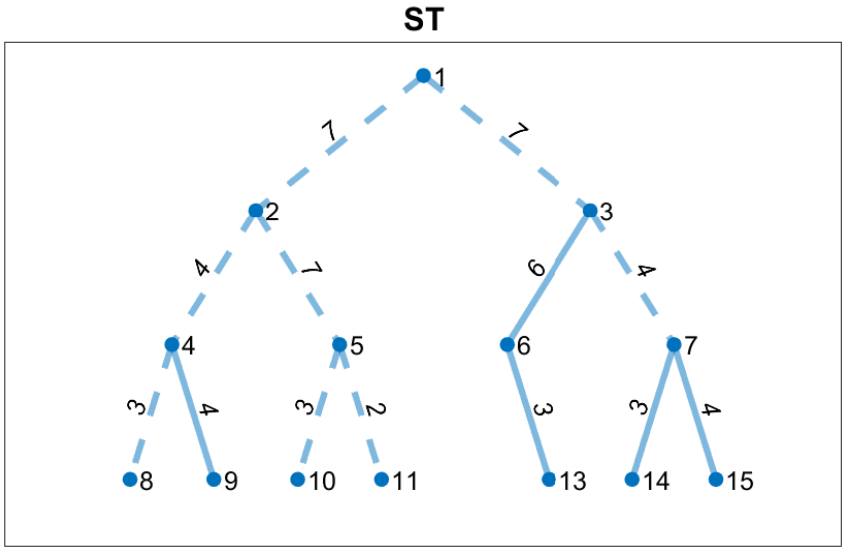

Figure 6.2: The scaling tree with 8 edges constructed by MDST method (dashed edges) for the set of trees given in Figure 3.1

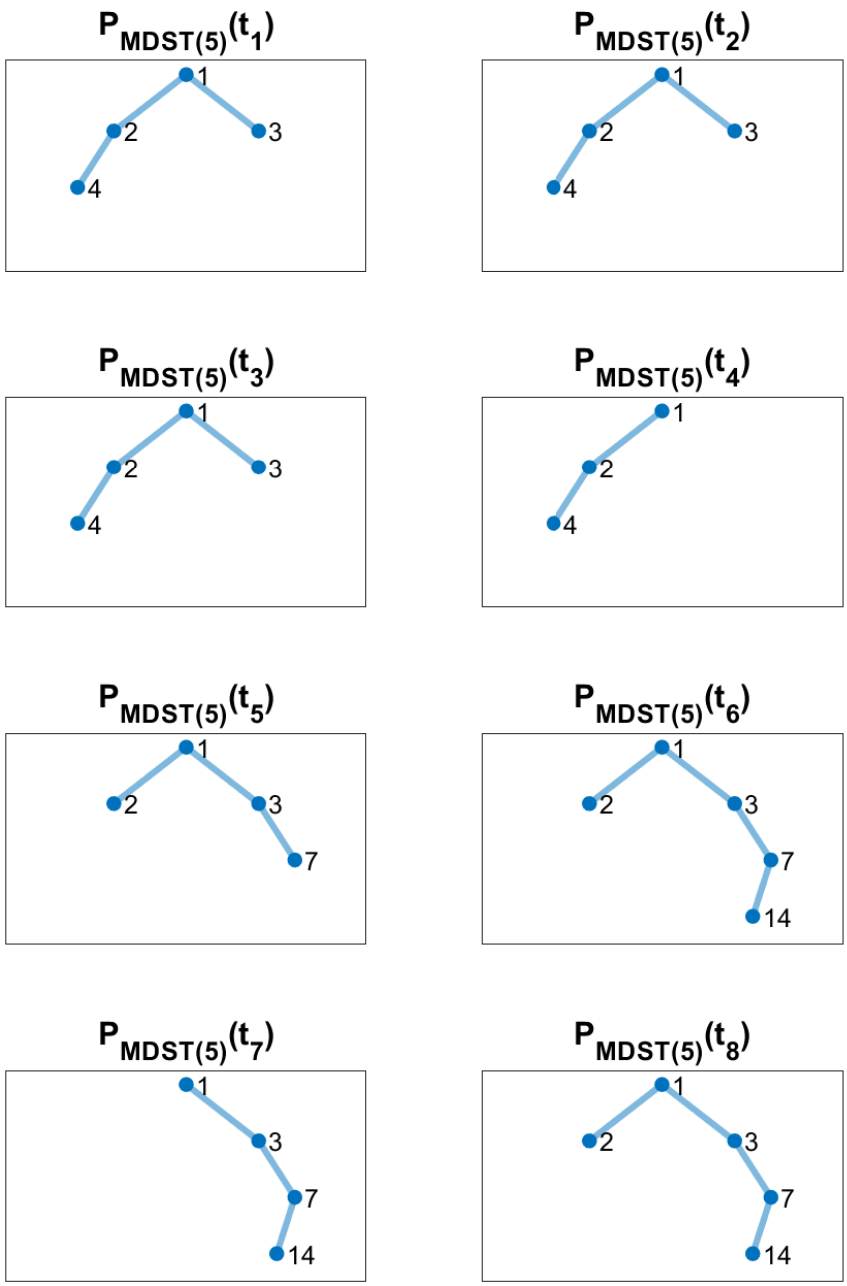

Figure 6.3: Projections of trees given in Figure 3.1 onto the scaling tree shown in Figure 6.1

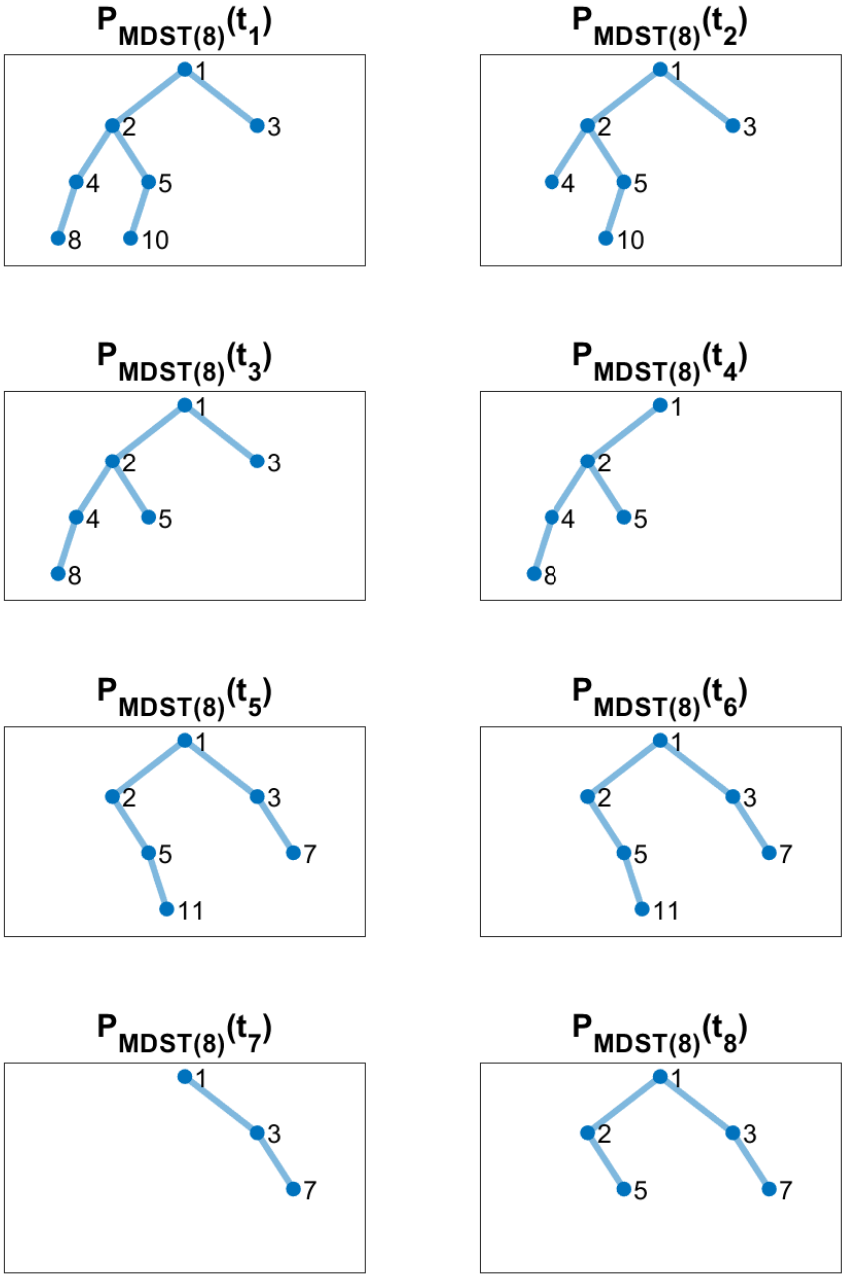

Figure 6.4: Projections of trees given in Figure 3.1 on the scaling tree shown in Figure 6.2

these figures, the dashed edges show the edges of the scaling trees. In Figures 6.3 and 6.4, the projections of the trees given in Figure 3.1 onto the scaling tree with five edges and with eight edges constructed by MDST method are given, respectively. Here the projections of t_i on sc_k is represented as $P_{MDST(k)}(t_i)$. Observe that, in both figures, the first four projections and last four have different structures where trees in the first cluster have longer paths on the left side while the longer paths are on the right side for other trees. Within similarity of clusters are high for both of them. For this instance, MDST method seems to be successful in keeping the cluster structure of the data.

Note that, in Figure 6.3 the trees that are in the same cluster are more similar to each other than the trees in Figure 6.4. This is expected because as the number of edges on the scaling tree increases, the edges that do not represent the cluster structure would be selected by the scaling tree as well. Although the objection function value z is smaller when $k = 8$, the cluster structure is more noticeable when $k = 5$. Thus, the objection function value may not represent the amount of useful information carried by the scaling tree. Therefore, computational studies are needed to broaden our understanding of the performances of MDST methods in different cases, which are presented in Chapter 7.

6.4 Multidimensional Scaling for Tree-Structured Data - Greedy Forward Algorithm

Since MDST method explained in Section 6.3 finds the optimal solution using an MILP formulation, it may be impractical on larger data sets. Hence, we propose two heuristic methods to find satisfactory solutions for multidimensional scaling for tree-structured data problems. Note that these solutions are not guaranteed to return optimal selections. We name the first proposed heuristic method as *Multidimensional Scaling for Tree-Structured Data - Greedy Forward Algorithm* (*MDSTGA*) method. MDSTGA method constructs sc_1 as the tree that keeps the pairwise distances proportionally closest where sc_1 is induced by the edge set that has only one edge which has no ascending edges, i.e., $E(sc_1) = \{e\}$ where $asc(e) = \emptyset$. Note that, for any m-ary trees, the number of possible sc_1 trees is limited with m since the root has at most m

many children. For all $k \geq 2$, MDSTGA method constructs sc_k as the tree that keeps the pairwise distances proportionally closest where sc_{k-1} is a subtree of sc_k.

Algorithm 4: MDSTGA

Inputs: Coherent set of trees $S = \{t_1, t_2, \ldots, t_n\}$ where $t_i = (N_i, E_i)$ for $i = 1, \ldots, n$ and the number of edges on the scaling tree k.
Initialization: Initialize the possible edges to be in the scaling tree.
Let ST be the support tree of S.
Let PE be the set of possible edges where $e \in PE$ if and only if $asc(e) = \emptyset$
$E(sc_0) = \emptyset$
for *k=1* **to** K **do**
    Selection: Select the edges of the scaling trees.
    for $e \in PE$ **do**
        sc'_k is induced by $E(sc_{k-1}) \cup \{e\}$
        Find α^e that minimizes the objective function value for given sc'_k
        Assign z^e as the objective function value obtained by using α^e
    end
    $e_k^* = \underset{e \in \{PE\}}{argminz^e}$
    Update: Construct the scaling tree and update the possible edges.
    sc_k is induced by $E(sc_k) \cup \{e_k^*\}$
    Remove e_k^* from PE.
    Insert the children set (if any) of the selected edge, $ch(e_k^*)$, into PE
end
return $\{sc_1, sc_2, \ldots, sc_K\}$

The steps of MDSTGA method are given in Algorithm 4. The inputs of the algorithm are a coherent set of trees S and the number of edges on the scaling tree K. The output of the algorithm is the set of scaling trees $\{sc_1, sc_2, \ldots, sc_K\}$. The algorithm starts with an initialization step which initializes the set of possible edges to be in the first scaling tree. First, it builds the corresponding support tree ST. Then it initializes the set of possible edges with the edges joining the root with its children. At the end, it defines $E(sc_0)$ as the empty set.

Table 6.2: The results of MDSTGA method on the set of trees given in Figure 3.1

k	**Edges of** sc_k	α	z	z'
1	2	0	7	—
2	2,4	0.13	9.1	70
3	2,4,3	0.17	11.5	67.7
4	2,4,3,7	0.29	12.3	42.4
5	2,4,3,7,14	0.38	13.9	36.6
6	2,4,3,7,14,6	0.43	14.9	34.7
7	2,4,3,7,14,6,13	0.50	12.5	25
8	2,4,3,7,14,6,13,9	0.63	13.1	20.8
9	2,4,3,7,14,6,13,9,8	0.70	13.9	19.9
10	2,4,3,7,14,6,13,9,8,5	0.75	14.8	19.7
11	2,4,3,7,14,6,13,9,8,5,11	0.80	12.4	15.5
12	2,4,3,7,14,6,13,9,8,5,11,10	0.89	6.6	7.4
13	2,4,3,7,14,6,13,9,8,5,11,10,15	1	0	0

After the initialization step, the selection and update steps follow for all $k \in \{1, \ldots, K\}$. The selection step first finds the best scaling parameter value for each edge e in the set of possible edges and assigns them as α^e. Then it selects the edge to be in sc_k, denoted by e_k^*, as the edge e that minimizes the objective function (6.3) where sc_k is induced by $E(sc_{k-1}) \setminus e$ and $\alpha = \alpha^e$. In the update step, sc_k is induced by the union of e_k^* and $E(sc_{k-1})$. Then PE is updated by removing e_k^* and adding children edges of e_k^*. At the end of the algorithm, the set of the scaling trees are returned. Using this set, for all $k \in \{1, \ldots, K\}$, the projection of a tree t_i onto sc_k can be obtained as $P_k(t_i) = t_i \cap sc_k$.

Here we explain how the algorithm finds the best scaling parameter for a given sc_k. For $i \in \{1, 2, \ldots, n-1\}$ and $m \in \{i, i+1, \ldots, n\}$, the difference term in the objective function (6.3) $(\alpha\, d(t_i, t_m) - d((t_i \cap sc_k), (t_m \cap sc_k)))$ takes a positive value when α is greater than $\frac{d((t_i \cap sc_k),(t_m \cap sc_k))}{d(t_i,t_m)}$ (the critical point of $i\&m$) and a negative value when α is less than the critical point of $i\&m$. Suppose p many unique critical points of all $i\&m$ pairs are obtained and sorted in ascending order. Then by selecting any two

consecutive critical points as the bounds, $p-1$ intervals for α can be constructed. If the intervals that are bounded by the biggest critical point and ∞, and smallest critical point and 0, are added to these $p-1$ intervals, then it is obtained that all possible α values are in these intervals and in each interval, the signs of the value of the difference terms do not change. Since α value in an interval that minimizes the objective function value (6.3) is one of the bounds of this interval, for a given sc_k, the best value of α is the critical point which gives the minimum objective function value. Note that best $\alpha = \infty$ is not possible, and if there is no critical point which is 0, then best α cannot take the value 0. For sorted critical points, one can find an alternative best value of α as the critical point that gives the objective function value that is not greater than the objective function values given by its adjacent critical points and less than at least one of these values. Therefore, the best α can be found by calculating and comparing the objective function values of the critical points one by one in an ascending order starting from the smallest critical point. Moreover, since we know that, for consecutive values of k, the best α values are close to each other, for $k \geq 2$, we start searching the best α value for sc_k from the closest critical point to the best α value for sc_{k-1}. Note that the searching is performed in descending order if the objective function value of the smaller adjacent critical point is less than the objective function value of the closest critical point to the best α value for sc_{k-1}.

We now explain MDSTGA method on given trees in Figure 3.1. At the beginning, there are two possible edges to be selected: the edges joining the root with node 2 and with node 3. For the first edge, which joins the root and node 2, the sorted unique critical points $\left(\frac{d(P_k(t_i),P_k(t_j))}{d(t_i,t_j)}\right)$ can be calculated as 0, 0.09, 0.1, 0.11, 0.13, 0.14, 0.25, and 0.5. Then, the objective function value for $\alpha = 0$ is calculated as 7. Then it is calculated as 13.1 for $\alpha = 0.09$. Since it is greater than 7, α value for the edge joining the root and node 2 is found as 0. Consequently, the objective function value for this edge is found as 7. When the same steps are followed for the edge joining the root and node 3, α and the corresponding objection function values are found as 0 and 7, respectively. Since the objective function values of two possible edges are equal, we arbitrarily select the edge joining the root and node 2 to be the edge of sc_1. Then the set of possible edges, PE, is updated by removing the selected edge from PE and inserting the children edges of the selected edge into PE, i.e., the

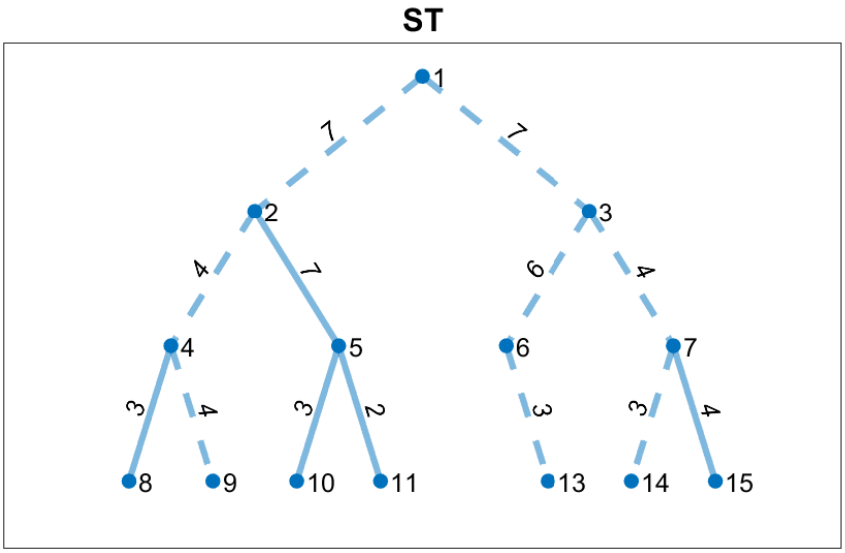

Figure 6.5: The scaling tree with 8 edges constructed by MDSTGA method (dashed edges) for the set of trees given in Figure 3.1

edges joining the node 2 with node 4 and with node 5. When the steps are repeated for all $k \in \{2, \ldots, 13\}$ values, the results are obtained as shown in Table 6.2. The organization of this table is the same with that of Table 6.1. Note that, for all $k \geq 2$, sc_{k-1} is a subtree of sc_k. Since the selected scaling trees by MDST and MDSTGA methods are the same for $k \in \{1, 2, 5, 12, 13\}$, their objective function values z are the same for these values of k. However, for the remaining k values, the objective function values z of the scaling trees selected by MDSTGA method are worse than the ones in Table 6.1.

As an illustrative example, the scaling tree with eight edges, sc_8, constructed by MDSTGA method is shown in Figure 6.5. In this figure, the dashed edges show the edges of the scaling tree. In Figure 6.6, the projections of given trees in Figure 3.1 onto the scaling tree with eight edges constructed by MDSTGA method are given. Here the projections of tree t_i onto sc_k is represented as $P_{MDSTGA(k)}(t_i)$. Observe that the first four projections and the last 4 have different structures where trees in the first cluster have a longer path on the left side while the other trees have a smaller path on this side. Within similarity of clusters are high but not as the ones in Figure 6.3 and Figure 6.4. For this instance, MDSTGA method seems to be successful in keep-

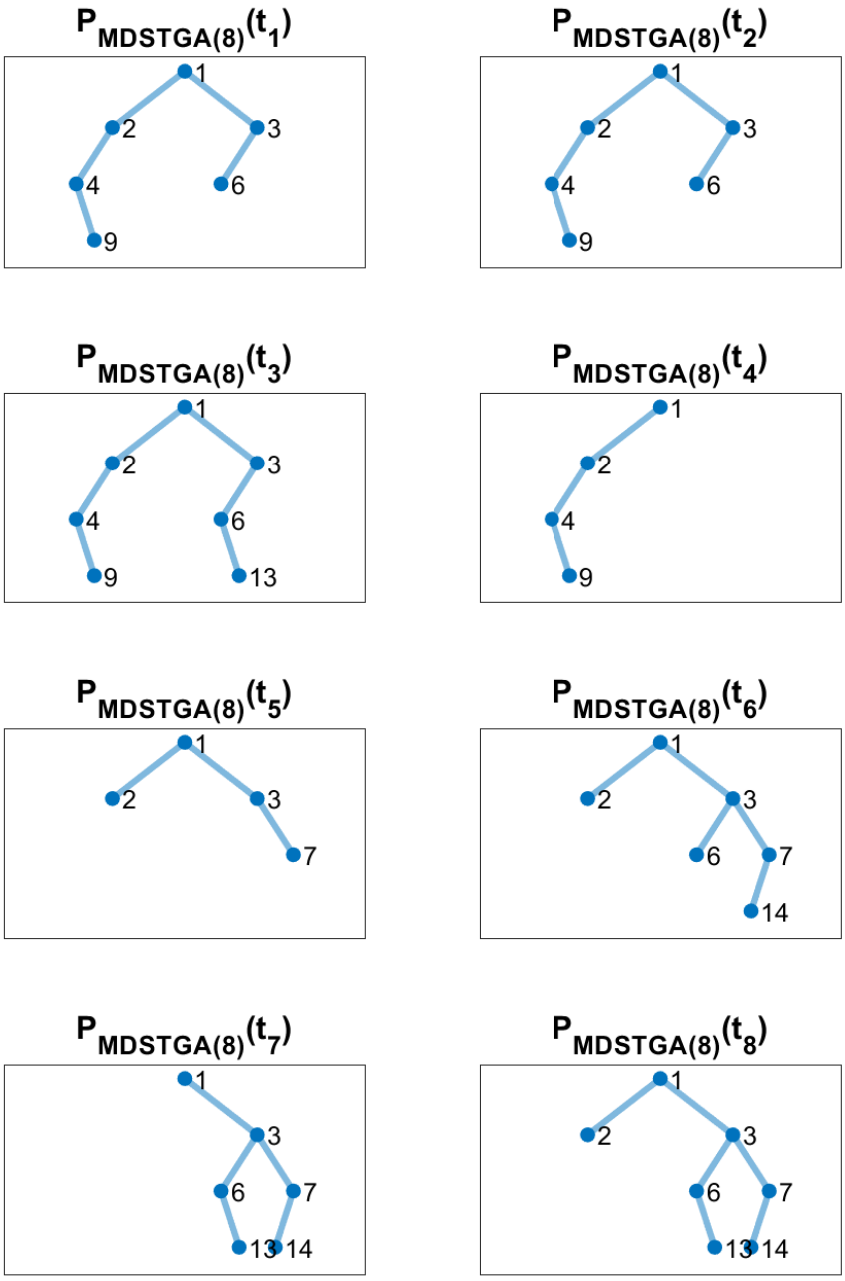

Figure 6.6: Projections of trees in Figure 3.1 on the scaling tree shown in Figure 6.5

ing the cluster structure of the data, but it is less successful than MDST method as expected. For further information about the performances of the methods, we design and explain the computational experiments in Chapter 7.

6.5 Multidimensional Scaling for Tree-Structured Data - Greedy Backward Algorithm

Since MDSTGA method may not select the parents of an informative edge on earlier steps, we propose its backward version and name it as *Multidimensional Scaling for Tree-Structured Data - Greedy Backward Algorithm* (*MDSTGB*) method. MDSTGB method is very similar to MDSTGA method, except it starts from $sc_{|E(ST)|} = ST$ and selects the least informative edges to remove them. For all $k \leq |E(ST)| - 1$, MDSTGB method constructs sc_k as the tree that keeps the pairwise distances proportionally the closest where sc_k is a subtree of sc_{k+1} with the same root.

The steps of MDSTGB method are given in Algorithm 5. Since it is very similar to Algorithm 4, we only explain the differences between the two methods. In the initialization step, it initializes $sc_{|E(ST)|}$, not sc_0. The selection and update steps are repeated for $k \in \{K, \ldots, E(ST)-1\}$ values in the opposite way. Therefore, sc'_k is constructed using sc_{k+1}, not sc_{k-1}. To update the set of possible edges, the parent of the selected edge is inserted, not its children. Note that the parent is not inserted if any of its children is on sc_k. Lastly, at the end of the algorithm, $\{sc_K, sc_{K+1}, \ldots, sc_{|E(ST)|}\}$ are returned.

We now explain MDSTGB method on the trees given in Figure 3.1. At the beginning, there are seven possible edges to be selected to remove since there are seven leaf edges of the support tree. For the first edge, which joins node 4 and node 8, the sorted unique critical points $\left(\frac{d(P_k(t_i),P_k(t_j))}{d(t_i,t_j)}\right)$ are calculated as 0, 0.67, 0.75, 0.83, 0.86, 0.88, 0.89, 0.90, 0.91, and 1. As a result of the local search around $\alpha = 1$, which is the scaling parameter for $sc_{|E(ST)|}$, the starting critical point of the best α value searching is determined as 0.91. This is because it gives the objective function value (6.3), 21.1, which is smaller than the one that $\alpha = 1$ gives, which is 30. Since critical point 0.91 is the smaller adjacent of critical point 1, the searching is performed in descending order.

Algorithm 5: MDSTGB

Inputs: Coherent set of trees $S = \{t_1, t_2, \ldots, t_n\}$ where $t_i = (N_i, E_i)$ for $i = 1, \ldots, n$ and the number of edges on the scaling tree k.

Initialization: Initialize the possible edges to be in the scaling tree.

Let ST be the support tree of S.

Let PE be the set of possible edges where $e \in PE$ if and only if $ch(e) = \emptyset$

$sc_{|E(ST)|} = ST$

for $k = |E(ST)| - 1$ **to** K **do**

  Selection: Select the edges of the scaling trees.

  for $e \in PE$ **do**

    sc'_k is induced by $E(sc_{k+1}) \setminus \{e\}$

    Find α^e that minimizes the objective function value for given sc'_k

    Assign z^e as the objective function value obtained by using α^e

  end

  $e^*_k = \underset{e \in PE}{argminz^e}$

  Update: Construct the scaling tree and update the possible edges.

  sc_k is induced by $E(sc_{k+1}) \setminus \{e^*_k\}$

  Remove e^*_k from PE.

  Insert the parent edge of e^*_k into PE if its any children is not on sc_k

end

return $\{sc_K, sc_{K+1}, \ldots, sc_{|E(ST)|}\}$

Table 6.3: The results of MDSTGB method on the set of trees given in 3.1

k	**Edges of** sc_k	α	z	z'
1	3	0	7	—
2	3,7	0.13	10.1	77.7
3	3,7,2	0.17	11.5	67.7
4	3,7,2,4	0.29	12.3	42.4
5	3,7,2,4,5	0.33	16.7	50.6
6	3,7,2,4,5,10	0.38	15.9	41.8
7	3,7,2,4,5,10,11	0.46	14.1	30.7
8	3,7,2,4,5,10,11,8	0.56	12.9	23
9	3,7,2,4,5,10,11,8,14	0.67	12	17.9
10	3,7,2,4,5,10,11,8,14,6	0.71	12.4	17.5
11	3,7,2,4,5,10,11,8,14,6,15	0.82	11	13.4
12	3,7,2,4,5,10,11,8,14,6,15,13	0.89	6.6	7.4
13	3,7,2,4,5,10,11,8,14,6,15,13,9	1	0	0

Then, the objective function value that $\alpha = 0.90$ gives is calculated as 21.1. After calculating them for the other α values, it is found that the best α value for the edge joining node 4 and node 8 is 0.89. Accordingly, the objective function value for this edge is found as 10.2. When the same steps are followed for the other possible edges, the objective function values are obtained as 10.2, 6.6, 13.3, 12, 12.7, 8.9, and 6.6, in the left-to-right order. Since two edges share the least objective function values, we arbitrarily select the edge joining node 4 and node 9 to remove it from the scaling tree. Therefore, sc_{12} is induced by the edges of ST except the selected edge. Then the set of possible edges, PE, is updated by removing the selected edge from it. Note that since the edge joining node 4 and node 8 is still on the scaling tree, the parent edge (joining node 2 and node 4) is not inserted into PE. When the steps are repeated for all $k \in \{1, \ldots, 11\}$ values, the results are obtained as shown in Table 6.3. Note that, for all $k \leq |E(ST)| - 1$, sc_k is a subtree of sc_{k+1}. Since the selected scaling trees by MDST and MDSTGB methods are the same for $k \in \{8, 9, 10, 13\}$, their objective function values are the same for these values of k. In addition to them, z values for $k = 11$ and $k = 12$ are also equal to each other. However, for the remaining

k values, the objective function values of the scaling trees selected by MDSTGB method are worse than the ones in Table 6.1. In order to broaden our understanding of the performances of the three MDS methods proposed for tree-structured data, we carry out extensive computational experiments using different data sets in Chapter 7.

CHAPTER 7

COMPUTATIONAL EXPERIMENTS FOR MDS METHODS FOR TREE-STRUCTURED DATA

In this chapter, the performances of the proposed MDS methods for tree-structured data are tested on synthetic and real data sets. To compare the performances of the methods, we follow a similar procedure as explained in Chapter 5. However, since these methods use the Hamming distance to measure the distances between the trees, in the tree k-means algorithm, we consider using *Graph Edit Distance* as a distance measure, which is equivalent to the Hamming distance in rooted labeled trees. We refer to this algorithm as tkm^{GED}.

The MILP model of MDST method is solved using CPLEX Studio IDE 12.10.0. The heuristic MDS methods for tree-structured data, RTC and tkm^{GED} are coded in MATLAB R2020b. They are run on an Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz, 8GB RAM Windows 10 PC. Computation times are measured by CPU time. The details of the computational experiments on synthetic data are given in Section 7.1 and on real data in Section 7.2.

7.1 The Performances of MDS Methods on Synthetic Data Sets

In this section, the performances of MDS methods for tree-structured data are tested on synthetic data sets. To obtain the performances, we follow a similar procedure as in Section 5.1.2. Unlike in Section 5.1.2, each group of trees has 25 trees.

We design 23 instances different *density*, *skewness*, and *height* parameters to analyze the performances of the methods on data sets having different cluster structures.

Similarly, four patterns are used to investigate the performances of the methods and effects of *density*, *skewness*, and *height* of trees. 23 instances belong to the following patterns:

1. Density-Different Clusters on Small Trees ($h = 5$),
2. Density-Different Clusters on Big Trees ($h = 10$),
3. Skewness-Different Clusters on Small Trees ($h = 5$),
4. Skewness-Different Clusters on Big Trees ($h = 10$).

In the rest of the section, computational results in each pattern are discussed, and results are presented in Figures 7.1 - 7.4. The organization of the panels in these figures is the same with that of the panels in the figures in Chapter 5. In these panels, the x-axis and y-axis represents the number of edges and adjusted Rand index, respectively. The solid, dashed, and dotted lines show the performance of MDST, MDSTGA, and MDSTGB methods, respectively.

7.1.1 MDS Methods on Density-Different Clusters on Small Trees

In this set of experiments, the height of trees is up to five ($h = 5$). The skewness level is fixed as *Too Left*, *Left*, and *Uniform*. In each data set, there are two groups in different densities, and the pairs of density levels are (0.99-0.95), (0.9-0.5), and (0.5-0.1), where the first number refers to the density level in the first group of trees and the latter is for the second group. We obtain 9 instances, and each instance is replicated ten times. As explained in Section 5.1.2, the generated partitions are obtained by tkm^{GED}, and the adjusted Rand index (ARI) scores are used to compare the accuracies of the generated partitions with respect to the original ones. The computational results of the proposed MDS methods for each instance are presented in Figure 7.1.

In the instances with *Too Left* and *Left* skewness levels, there is a cluster structure in the data sets as they are presented in Figure 7.1-(a),(b),(c),(d),(e),(f). In these instances, the projected trees generated by MDSTGB are clustered with having a high ARI score using less than 10 edges. Note that this is equivalent to about 15 percent

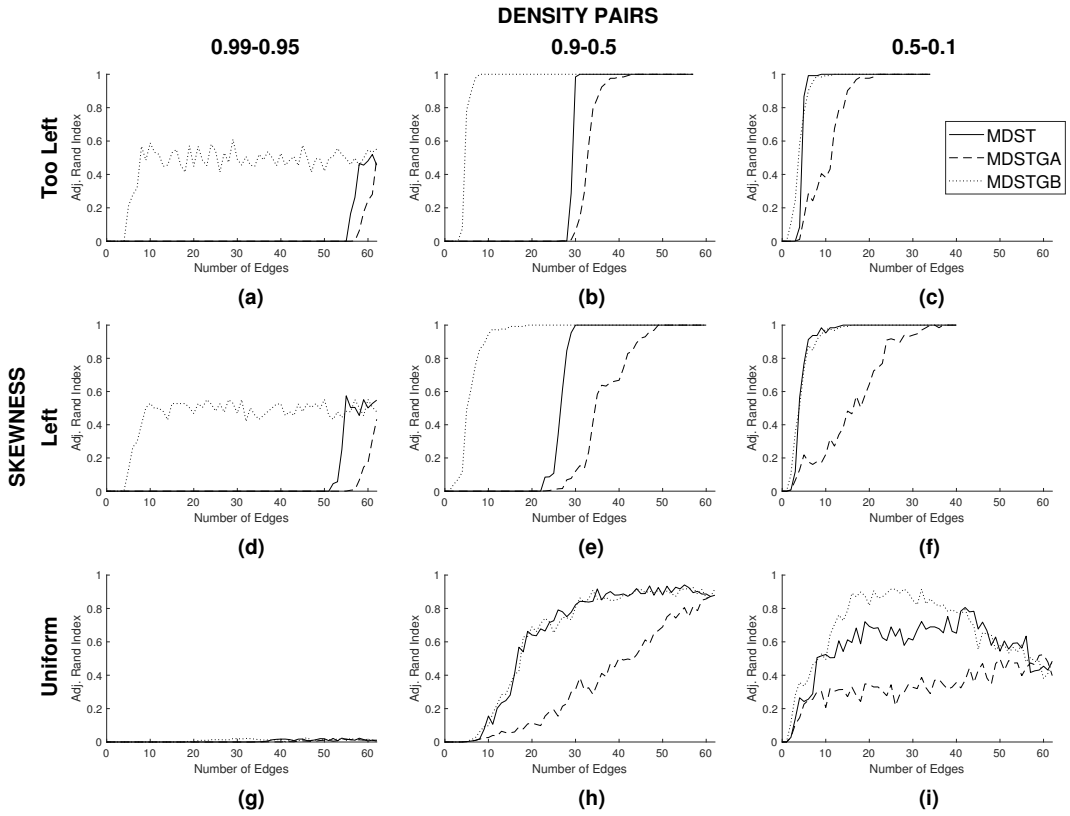

Figure 7.1: Clustering performances of the MDS methods for tree-structured data on density-different data sets with $h = 5$

of the total number of edges on the support tree. The ARI scores of MDSTGB get better while the ratio of density pairs increases (see the first two rows of Figure 7.1). However, since MDST and MDSTGA methods initially select the common edges of all trees, they need more edges to lead to higher clustering accuracies. If there are p common edges in the data set, the ARI score of MDST is high when $k = p+1$. This is because the scaling trees by MDST method do not have to include previously selected edges, unlike to MDSTGA method. Hence, the ARI score of MDSTGA method is not as high as MDST when $k = p + 1$.

For the instance having no cluster structure (see Figure 7.1-(g)), the ARI scores of the methods are negligible. In the instances with (0.9-0.5) and (0.5-0.1) pairs of density levels and *Uniform* skewness level, projected trees generated by MDST and MDSTGB methods are successfully clustered using half of the total number of edges on the support tree (see Figure 7.1-(h),(i)). Note that, in Figure7.1-(i), the projected trees generated by MDST and MDSTGB can be clustered easier than the original data for k values between 10 and 55. This may be because these methods remove the noises from the original data set.

Therefore, it can be concluded that the proposed backward heuristic algorithm MDSTGB performs better in leading to successful clustering by generating projected trees using only a few edges of the support tree. In general, the proposed optimization model needs a few more edges than the number of common edges to reach higher ARI scores, while the proposed forward heuristic algorithm MDSTGA usually needs more edges since the selected common edges cannot be removed in the following iterations.

7.1.2 MDS Methods on Density-Different Clusters on Big Trees

In this section, only one instance is considered. Here the height of trees is up to 10. The skewness level is fixed as *Left* for all trees, and the pair of density levels is selected as (0.09-0.05). The ARI scores of the methods are computed with a similar procedure as in Section 5.1.2. Here we run MDST method with the number of edges that are multiples of 25 in between 1 and 225 and MDSTGA and MDSTGB methods that are multiples of 5 in between 1 and 225. Note that all support trees of the pro-

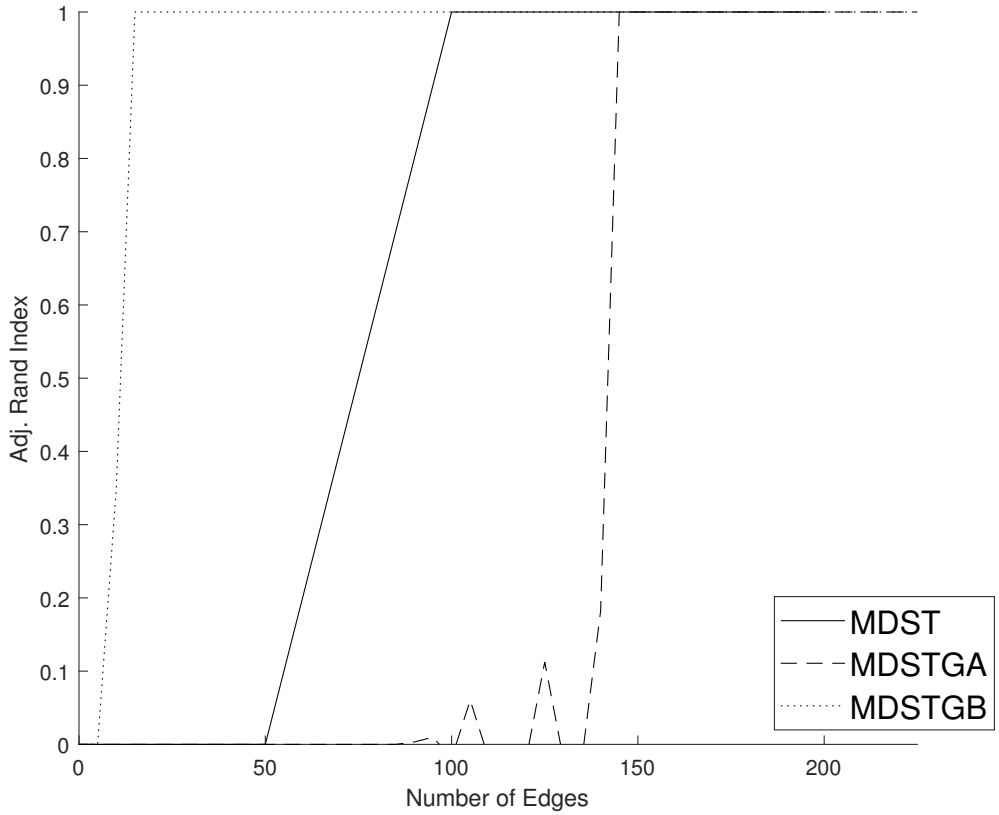

Figure 7.2: Clustering performances of the MDS methods for tree-structured data on density-different data sets with $h = 10$

duced data sets have more than 225 and less than 250 edges. The results are presented in Figure 7.2. In Figure 7.2, the lines between the results are also plotted, e.g., for MDST method the results for 25 and 50 number of edges are obtained, and these two points are connected with an artificial line.

It can be seen in Figure 7.2 that the projected trees generated by MDSTGB are well clustered using less than 10 edges which is about 5 percent of the total number of edges of the support tree, but MDSTGA method needs approximately 145 edges for a similar result. The ARI score of MDST is sufficient when $k = 100$ while it is zero when $k = 50$. Note that the bad performance of MDST on 50 edges is due to the selection of the common edges. The results displayed in Figure 7.2 are consistent with our earlier results. On data sets having density-different clusters, MDSTGB performs the best, while MDST outperforms MDSTGA.

7.1.3 MDS Methods on Skewness-Different Clusters on Small Trees

In this set of experiments, 12 instances are generated with the following parameter settings. h is taken as 5 and the density level is fixed as 0.7 and 0.3 for all trees. In each data set there are 2 clusters in different skewness levels and the pairs of skewness levels are (*Left-Uniform*), (*Left-Right*), (*Too Left-Left*), (*Too Left-Uniform*), (*Too Left-Right*), and (*Too Left-Too Right*). The ARI scores of the methods are computed with a similar procedure as in Section 5.1.2 which are displayed in Figure 7.3.

For the instance having no cluster structure (see Figure 7.3-(e),(f)), the ARI scores of the methods are negligible. In the instances having a cluster structure with 0.3 density level, the projected trees generated by each of the methods are successfully clustered using about 5 edges which is less than 10 percent of the total number of edges on the support tree (see Figure 7.3-(b),(d),(h),(j),(l)). However, it can be inferred from Figure 7.3-(a),(c),(g),(i),(k) that all methods need more edges for a similar result, in the instances having a cluster structure with 0.7 density level. In these instances, while MDST and MDSTGA methods require more than the number of common edges, MDSTGB needs less than 20 edges to perform well.

Hence, it can be concluded that if there is a skewness-different cluster structure on a data set having small trees, then the projected trees generated by the proposed backward heuristic algorithm successfully keep this structure data using only a few edges on the support tree, where this number decreases as the density of the data set decreases. For a similar result, the proposed optimization model needs a few more edges than the number of common edges, and the proposed forward heuristic algorithm demands more edges.

7.1.4 MDS Methods on Skewness-Different Clusters on Big Trees

In this section, only one instance is considered. The height of trees is up to five ($h = 5$), the density level is 0.3 for all trees, and the pair of skewness levels is (*Left-Right*). MDST method is run with the number of edges that are multiples of 50 in between 1 and 450, and MDSTGA and MDSTGB methods are run with $k \in \{5, 10, \ldots, 450\}$. By following a similar procedure as in Section 7.1.1, with a slight change that only

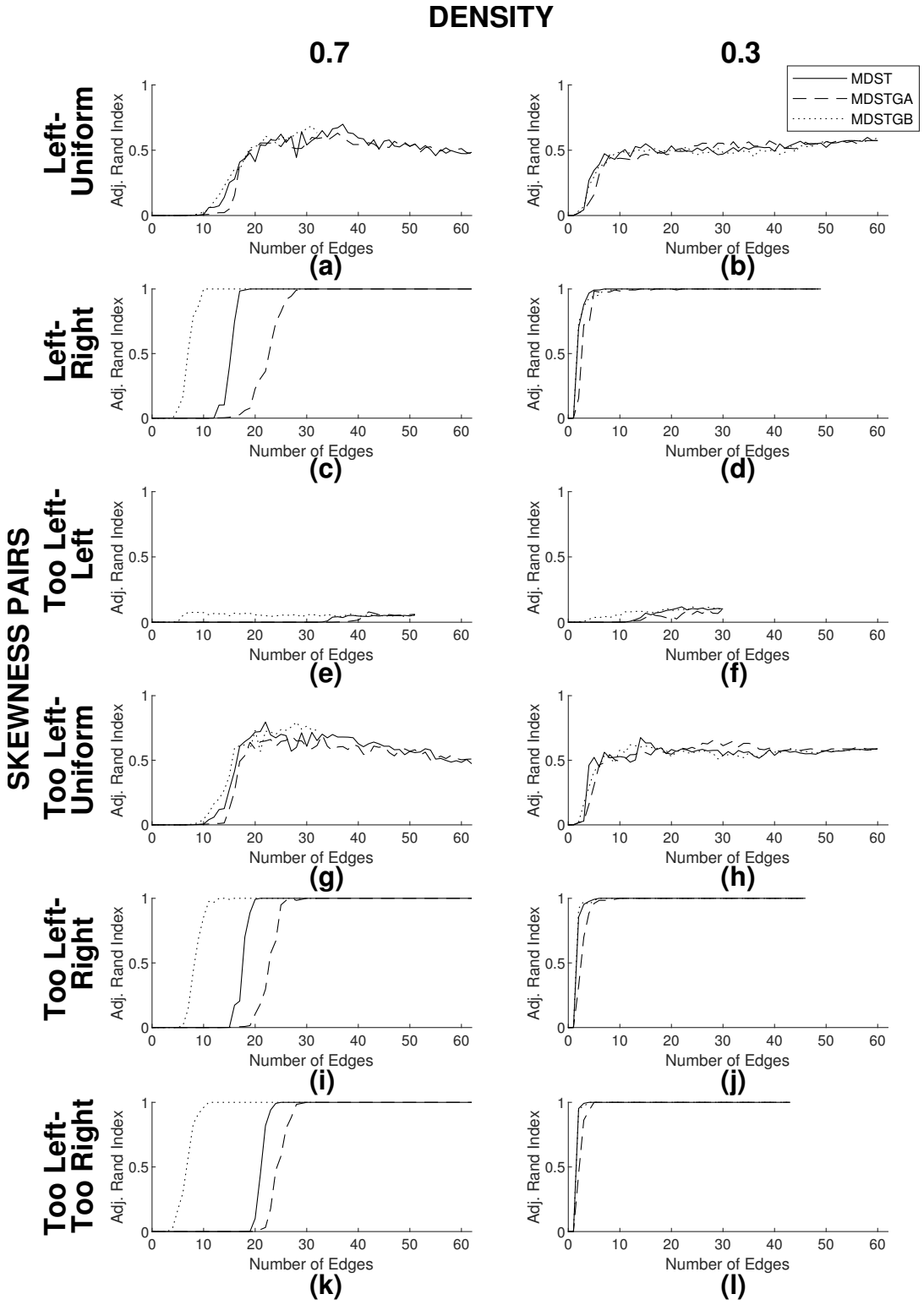

Figure 7.3: Clustering performances of the MDS methods for tree-structured data on skewness-different data sets with $h = 5$

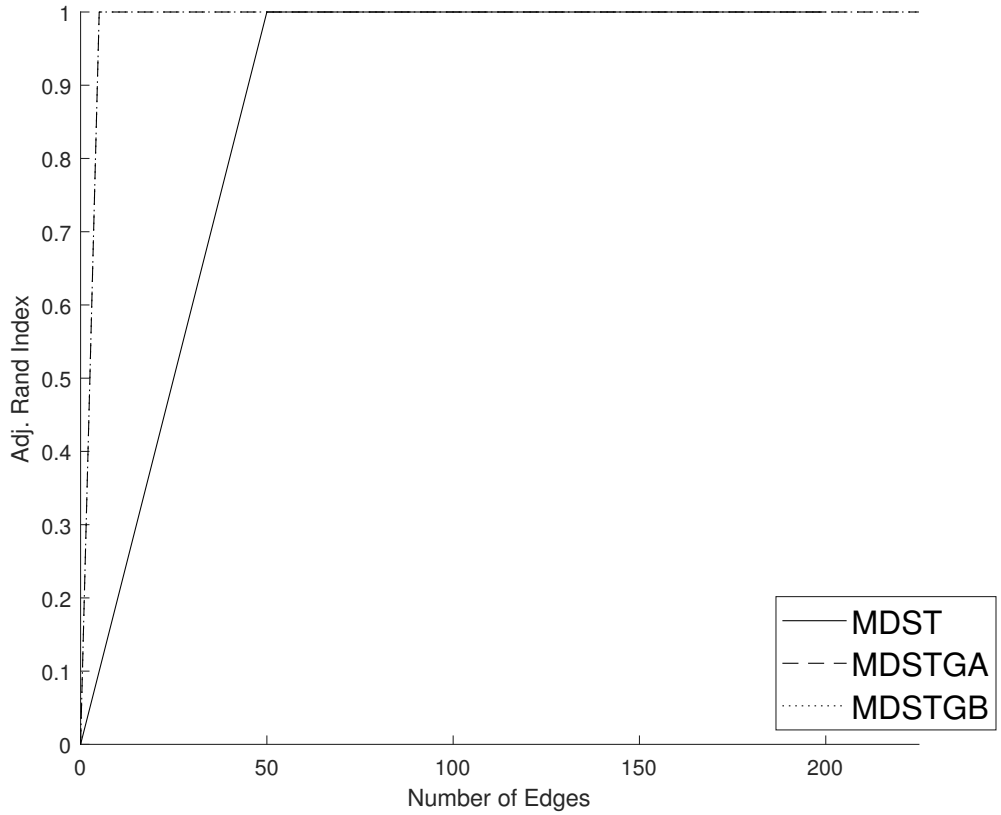

Figure 7.4: Clustering performances of the MDS methods for tree-structured data on skewness-different data sets with $h = 10$

one replication is made instead of 10, the adjusted Rand index scores of the methods are computed, which are presented in Figure 7.4. Here the lines between the results are also plotted.

The support tree of the produced data set has 1290 edges. As it can be seen in Figure 7.4, the ARI scores of all three methods are equal to 1 for the smallest given k values whose are 5 for the heuristic methods and 50 for the optimization method. The results presented in Figure 7.4 are consistent with our earlier results in Section 7.1.3.

For this instance, the solution time of the MILP model for $k \in \{50, 100, \ldots, 450\}$ values are obtained as 853, 1261, 2382, 1905, 3660, 3866, 2919, 3265, and 3080 seconds, respectively. MDSTGA and MDSTGB find the edges and project the trees onto them for the same k values in 61 and 2180 seconds in total, respectively. The

computational time consumed by tkm^{GED} is not remarkable. For example, it takes 28 seconds to cluster the original data set of this instance. Projected data sets are clustered in shorter times.

7.2 The Performances of MDS Methods on a Real Data Set

In this section, we test the performances of the MDS methods for tree-structured data on the brain artery data set, which is introduced in Section 5.2. We assume that this data set has a bi-cluster structure, and the original partitions of the trees are obtained as follows. tkm^{GED} is run with 100 initializations, and out of 100 the best solution is taken as the original partition. Note that in this case, the original partition is not the same as the one found in Section 5.2.1. MDST method is run with the number of edges that are multiples of 50 in between 1 and 450 and MDSTGA and MDSTGB methods that are multiples of 5 in between 1 and 450. MDST method is limited to 10800 seconds, and for each k value, it does not find the optimal solution in this limited time. For $k \in \{50, 100, \ldots, 450\}$ values, the gap between the best found objective value and the best possible bound values, which are calculated as the difference between these values divided by the best found objective function value, are 40, 61, 56, 56, 69, 47, 64, 87, and 66 percent, respectively. By following the same procedure that is explained in Section 5.1.2, with a slight change that generated partitions are obtained with 100 initializations on tkm^{GED} instead of 10, the adjusted Rand index scores of the methods are computed. The results are displayed in Figure 7.5. Here the lines between the results are also plotted.

From Figure 7.5, it can be seen that the ARI score of MDSTGB is more than 0.9 after $k = 150$. For $k \in \{115, 120, \ldots, 360\}$ values, the ARI score of MDSTGA is more than 0.4. However, after $k = 365$, it decreases because MDSTGA begins to select noisy edges of the data set. Although for 150, 300, 400, and 450 values of k, the ARI score of MDST is more than 0.9, it is not stable for all k values. It may be because the results of MDST are not optimal. For example, when the MILP model is run for $k = 350$ with the time limit of 86400 seconds, the gap, objective function value z, and the ARI score of the method become 22 percent, 10024, and 0.95, respectively.

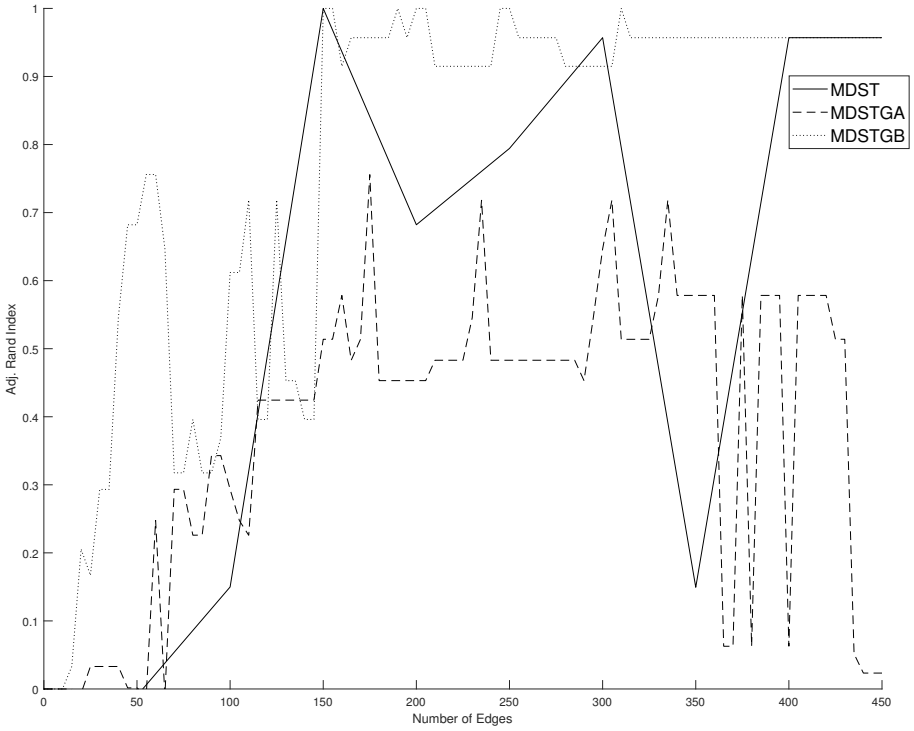

Figure 7.5: Clustering performances of the MDS methods for tree-structured data on the real data set

It can be concluded that the proposed optimization model is not able to solve medium size instances in 3 hours, the forward heuristic method cannot represent the cluster structure using even half of the total number of edges on the support tree, and the backward heuristic method is successful in keeping the cluster structure on the projected data using about 15 percent of the total number of edges on the support tree.

7.3 Comparison of the Performances of PCA and MDS Methods for Tree-Structured Data

Since the performances of PCA methods are calculated for the number of treelines, unlike edges, comparing the results of the PCA and MDS methods is not straightforward. However, if the ARI scores of PCA methods are represented for the number of edges on the principal component trees, then it may be possible to compare the ARI scores of PCA and MDS methods. Note that, for a given number of treelines, the number of edges on the principal component trees may differ on each replication. Hence, we provide the average number of edges on principal component trees. Be aware that in Chapter 4, 100 trees are generated for each group of trees, and tkm^{VEO} is used for clustering while the number of trees is 25, and tkm^{GED} is used in this chapter.

To compare the performances of PCA and MDS methods, we choose the instance with $h = 5$, the skewness level *Too Left* and the pair of density levels (0.5-0.1). In this instance, the performances of PCA and MDS methods can be found in Figures 5.1-(c) and 7.1-(c), respectively. It can be seen from the figures that all methods are successful in keeping the cluster structure using a few treelines or edges. The minimum number of required treelines to lead 0.95 ARI score are 2, 3, and 5 for VMAX, SMIN, and DMIN methods, respectively. The corresponding numbers of edges are 8, 12.3, and 17 on average. On the other hand, MDST, MDSTGA, and MDSTGB need 6, 17, and 7 edges for similar results. Therefore, it can be concluded that VMAX, MDST, and MDSTGB methods performs well in this instance. Among these methods, the MDS methods are better than the PCA method with a slight difference. It is reasonable since the MDS methods can select their scaling trees from a less bounded area.

CHAPTER 8

CONCLUSION

The increasing need for deeper analyses, measurement capabilities, and data collecting technology result in the collection of complex data sets. Tree-structured data sets, where the data objects are trees, are one of these complex data sets. The use of such complex data objects brings in difficulties in data analysis and necessitates the use of dimension reduction techniques. However, many of the classical dimension reduction techniques are developed to work with vectors in the Euclidean space. Therefore, in this thesis, we aim to develop dimension reduction methods for tree-structured data where the data objects are rooted labeled trees. The classical PCA and MDS are considered, and their extensions are proposed for tree-structured data.

In PCA for tree-structured data, principal components are taken as the treelines. In the literature, all studies use distance-based objective functions to find the principal components. In this study, two methods with variance-based objective functions are developed. Computational experiments show that the proposed methods are able to select more informative treelines as principal components.

Since there is no study that uses MDS to project trees in the tree space, the concepts of MDS for tree-structured data are developed in this study. The edges are considered as the dimensions in our implementation of the MDS for tree-structured data, and the aim is to keep the Hamming distances between pairs of trees proportionally close. For this purpose, an MILP model is proposed, but it is not able to solve medium size instances to optimality in 1 hour. Therefore two heuristic methods, in which the edges are greedily selected, are proposed as well. From the computational experiments, it can be concluded that the proposed backward heuristic is successful in keeping useful information as high clustering accuracy is achieved with only a fraction of the edges.

However, since MDST and MDSTGA methods select the common edges of all trees firstly, they need more edges to achieve high clustering accuracies.

As a future work, different definitions of principal components for trees can be considered, such as k-treelines or edges, and the proposed PCA methods can be applied to these new problems. Variance-based PCA methods can be extended for graph-structured data where the data objects are more general than rooted labeled trees. Moreover, the second objective function z' introduced in this thesis can be studied, and exact & heuristic methods can be developed in future studies.

REFERENCES

[1] J. S. Marron and A. M. Alonso, "Overview of object oriented data analysis," *Biometrical Journal*, vol. 56, no. 5, pp. 732–753, 2014.

[2] B. Aydın, G. Pataki, H. Wang, E. Bullitt, J. S. Marron, *et al.*, "A principal component analysis for trees," *The Annals of Applied Statistics*, vol. 3, no. 4, pp. 1597–1615, 2009.

[3] X. Chen, Y. Fang, M. Yang, F. Nie, Z. Zhao, and J. Z. Huang, "Purtreeclust: A clustering algorithm for customer segmentation from massive customer transaction data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 3, pp. 559–572, 2017.

[4] H. Wang, J. Marron, *et al.*, "Object oriented data analysis: Sets of trees," *The Annals of Statistics*, vol. 35, no. 5, pp. 1849–1873, 2007.

[5] D. Dinler, M. K. Tural, and N. E. Ozdemirel, "Centroid based tree-structured data clustering using vertex/edge overlap and graph edit distance," *Annals of Operations Research*, vol. 289, no. 1, pp. 85–122, 2020.

[6] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical association*, vol. 66, no. 336, pp. 846–850, 1971.

[7] S. R. Aylward and E. Bullitt, "Initialization, noise, singularities, and scale in height ridge traversal for tubular object centerline extraction," *IEEE transactions on medical imaging*, vol. 21, no. 2, pp. 61–75, 2002.

[8] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[9] H. Hotelling, "Analysis of a complex of statistical variables into principal components.," *Journal of educational psychology*, vol. 24, no. 6, p. 417, 1933.

[10] W. S. Torgerson, "Multidimensional scaling: I. theory and method," *Psychometrika*, vol. 17, no. 4, pp. 401–419, 1952.

[11] J. B. Kruskal, "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis," *Psychometrika*, vol. 29, no. 1, pp. 1–27, 1964.

[12] A. M. Bronstein, M. M. Bronstein, and R. Kimmel, "Generalized multidimensional scaling: a framework for isometry-invariant partial surface matching," *Proceedings of the National Academy of Sciences*, vol. 103, no. 5, pp. 1168–1172, 2006.

[13] C. R. Woese, "Interpreting the universal phylogenetic tree," *Proceedings of the National Academy of Sciences*, vol. 97, no. 15, pp. 8392–8396, 2000.

[14] M. Van Oven and M. Kayser, "Updated comprehensive phylogenetic tree of global human mitochondrial dna variation," *Human mutation*, vol. 30, no. 2, pp. E386–E394, 2009.

[15] N. Lu and H. Miao, "Clustering tree-structured data on manifold," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 10, pp. 1956–1968, 2015.

[16] T. Dalamagas, T. Cheng, K.-J. Winkel, and T. Sellis, "Clustering xml documents using structural summaries," in *International Conference on Extending Database Technology*, pp. 547–556, Springer, 2004.

[17] 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*, vol. 23, no. 2, pp. 418–438, 2014.

[18] Y. Wang, J. Marron, B. Aydin, A. Ladha, E. Bullitt, and H. Wang, "A nonparametric regression model with tree-structured response," *Journal of the American Statistical Association*, vol. 107, no. 500, pp. 1272–1285, 2012.

[19] A. Flesia, "Unsupervised classification of tree structured objects," in *BIOMAT 2008*, pp. 280–299, World Scientific, 2009.

[20] C. A. Alfaro, B. Aydın, C. E. Valencia, E. Bullitt, and A. Ladha, "Dimension

reduction in principal component analysis for trees," *Computational Statistics & Data Analysis*, vol. 74, pp. 157–179, 2014.

[21] B. Aydın, G. Pataki, H. Wang, A. Ladha, E. Bullitt, and J. Marron, "New approaches to principal component analysis for trees," *Statistics in Biosciences*, vol. 4, no. 1, pp. 132–156, 2012.

[22] T. M. Nye *et al.*, "Principal components analysis in the space of phylogenetic trees," *The Annals of Statistics*, vol. 39, no. 5, pp. 2716–2739, 2011.

[23] D. M. Hillis, T. A. Heath, and K. S. John, "Analysis and visualization of tree space," *Systematic biology*, vol. 54, no. 3, pp. 471–482, 2005.

[24] S. Skwerer, E. Bullitt, S. Huckemann, E. Miller, I. Oguz, M. Owen, V. Patrangenaru, S. Provan, and J. S. Marron, "Tree-oriented analysis of brain artery structure," *Journal of mathematical imaging and vision*, vol. 50, no. 1, pp. 126–143, 2014.

[25] R. W. Hamming, "Error detecting and error correcting codes," *The Bell system technical journal*, vol. 29, no. 2, pp. 147–160, 1950.

[26] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina, "Web graph similarity for anomaly detection," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 19–30, 2010.

[27] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[28] P. Bendich, J. S. Marron, E. Miller, A. Pieloch, and S. Skwerer, "Persistent homology analysis of brain artery trees," *The annals of applied statistics*, vol. 10, no. 1, p. 198, 2016.