Stock-correlation Networks: Extraction and Optimisation

A thesis submitted for the degree of

Doctor of Philosophy

by

Seyed Soheil Hosseini

Supervisors:
Dr. Tianhai Tian
Prof. Nick Wormald

School of Mathematics
Monash University, Australia
December, 2021

Copyright Notice

Declaration

This thesis is an original work of my research and contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Signature:

Seyed Soheil Hosseini

2 December 2021

Acknowledgments

The completion of this thesis would not have been possible without the expertise of my amazing supervisor Dr. Tianhai Tian. I would like to express my thanks to him for supervising me throughout my Ph.D. projects and being there for me along the ups and downs of my Ph.D. journey.

I would like to express my profuse gratitude to my co-supervisor Prof. Nick Wormald. I am thrilled to have had the privilege to be given the opportunity to do my Ph.D. under his guidance. His dynamism, encouragement and sincerity inspired me to do my best. I learned so much from him in research methodology, mathematics and effective writing. Most importantly, I can say from the bottom of my heart that his patience, humility and goodwill taught me to be a better person.

I also owe a debt of my utmost gratitude to John Chan, our thoughtful, warm-hearted and caring research coordinator who would go out of his way and do way beyond his responsibilities to ensure the well-being and success of us, Ph.D. students.

I am also deeply indebted to Professor Andreas Ernst for giving me the chance to discuss some of my projects with him and gaining his invaluable input. Besides, I highly appreciate his favour to give me access to the Max-process cluster.

I would like to extend my sincere thanks to my amazing family who are always in my heart, and the geographical barriers could not disrupt their support and never ending love.

Last but not least, I would like to thank all my friends that I got the chance to meet at the School of Mathematics who made me feel at home and have wonderful memories while studying for my degree. I cannot describe in words how much I appreciate the experience of having you Ph.D. peers around.

Abstract

A complex network consists of a large number of vertices with relationships between them. One type of such a network is a stock-correlation network. To create such a network, a widely applied procedure is deriving the similarity matrix of stocks based on their *log*-returns, calculating the distance matrix based on the similarity matrix, and generating a network representing the relationships between stocks. As for the similarity matrix, the most common applied measure is Pearson correlation coefficient which can be converted to a Euclidean distance measure by a simple formula. After getting the distance (or similarity) matrix, we can consider this matrix as the adjacency matrix of a complete weighted graph K_n. The stock-correlation network is obtained after filtering the edges of K_n. In other words, stock-correlation network refers to a subgraph of the complete weighted graph. These networks have found applications in portfolio optimisation and studying the behaviour of stocks—especially during financial crises.

A "good" filtration algorithm for generating stock-correlation networks does not have a clear-cut definition in the literature. Basically, there is no specified standard to evaluate the output network of a filtration algorithm based on. Despite this, several criteria such as sparsity, scale-free behaviour, homogeneity of cliques, survival ratio, good clustering, and robustness have been mentioned in the literature as desirable aspects of a good stock-correlation generating algorithm. Of all these criteria, clustering seems to have been given the most importance. The assessment of this criterion is often based on how well the clusters match the economic sectoral classification of stocks. Factoring in the above, we have tried to achieve the following which is the overall theme of this thesis.

1. Coming up with algorithms of building a stock-correlation network to get a better clustering output compared to previously proposed algorithms
2. Utilising more information from the complete weighted graph to generate the stock-correlation network compared to the previous works in this area
3. Avoiding many of the structural limitations that exist in the networks proposed in the literature

The contribution of the thesis to this area is:

We have proposed a new stock-correlation generating algorithm: the proportional degree (PD) algorithm. In this algorithm, the degree of each vertex in the network is proportional to its total weight in K_n where the total weight of the ver-

tex v_i is defined as the total sum of the weights of edges connecting v_i to all the other vertices in K_n. We have compared the resulting network to that of a previously applied algorithm called PMFG, and we found that it demonstrates a better homogeneity of cliques, and it has a better clustering performance. Homogeneity of cliques refers to the proportion of cliques in which all the stocks belong to the same economic sector. Regarding clustering, we have used the normalised spectral clustering (NSC) on K_n, the PD network, and the PMFG network. Then we have made use of the adjusted rand index (ARI) to evaluate how well the clusters in each of the two networks (PD and PMFG) match those of K_n. Lastly, we have shown that PD is more robust than PMFG in the absence of some random edges. To be more precise, we have shown that by removing a set of random edges from both PD and PMFG, the clustering performance of PD diminishes to a less extent compared to PMFG.

We have proposed another network generating algorithm called the residual sum of squares optimal tree (RSSOT). The goal of this work is to improve upon the already existing widely used MST algorithm. In MST, in order to decide whether to connect two components through an edge, the only deciding factor is the shortest possible edge that can connect those two components. We put forward the RSSOT algorithm to use more information from the distance matrix (that is the adjacency matrix of K_n) to build the tree. We look for a tree in which the distance between any two vertices on the tree is as close as possible to their distance in the distance matrix—or equivalently in K_n. To this end, we have used the residual sum of squares (RSS) as the optimality criterion. We have demonstrated how to come up with the equations that find the edge weights for a given tree based on the tree structure. We have also used two metaheuristics, namely simulated annealing (SA) and iterated local search (ILS) to explore the search space. We have shown that ILS has a better performance compared to SA when the dispersion of the distance values in K_n is small. However, for larger dispersion in the values of distances in K_n, both SA and ILS demonstrate a similar performance. The resulting tree obtained by this work did not demonstrate a structure that is useful as a stock-correlation network.

Lastly, we have proposed an algorithm called the non-negative tree (NNT) to find an unrooted binary tree in which the edge weights are subject to non-negativity. MST is associated with the output of a hierarchical clustering method called single linkage cluster analysis (SLCA). SLCA has the same shortcomings as MST, that is, the only factor that we take into account in order to connect two components is the shortest edge that can connect those two components. To address this shortcoming,

as with the previous work above, we want to find an unrooted semi-labelled binary tree (the leaves are the stocks and the internal vertices are unlabelled with a degree of three), in which the path length between any two leaves best estimates their distance in K_n. However, unlike our previous work (work 2 above), the edge weights cannot be negative. We have proposed a scheme to avoid negative weighted edges. So we mirror theoretical conclusions we obtain are that we can turn a negative edge weight into positive using the nearest neighbourhood interchange (NNI), and that substituting a negative edge weight with zero is better than substituting that with any other non-negative value. The resulting unrooted binary tree of this algorithm yields clusters that match the economic sectoral classification of stocks better compared to SLCA. However, the clustering performance of this work does not happen to be significantly different from some much simpler previously used hierarchical clustering algorithms.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

1.1 Context

'Complex systems' is the term referring to the study of systems with a significant number of components in which we want to find out how the relationships between those components affect the behaviour of the system. The study of complex systems includes concepts from various disciplines such as mathematics, statistics, and computer science. One type of complex system is a complex network which consists of a large number of vertices and the relationships between them [Albert and Barabási, 2002]. There are many examples of such networks such as the world-wide web [Albert et al., 1999; Barabási et al., 2000], citation networks [Redner, 1998; Small, 1973], social networks [Galaskiewicz and Wasserman, 1993; Wasserman and Faust, 1994; Watts et al., 2002; Newman et al., 2002], and financial networks [Boss et al., 2004; Soramäki et al., 2007]. The focus of this thesis is financial networks and more specifically, the stock-correlation network.

Stock-correlation network refers to a network in which some pairs of stocks are connected to each other through what is referred to as *edge* or *link*. A weight is often assigned to each edge which specifies how "similar" the two stocks are that are connected through that edge. In this case, the heavier an edge is, the more similar the two stocks that are connected by it are. In contrast, the edge weights can denote dissimilarity—or equivalently distance. In this case, the lighter an edge is,

the more similar the two stocks on its two ends are. Similarity between two stocks in the context of stock-correlation networks refers to their similarity in price action, that is the drop or increase in their price in different time frames over a long period of time. Often the pairwise similarity values between stocks are stored in a matrix such that the entry in row i and column j denotes the similarity between the stocks that they—the row and the column—represent. This is referred to as the similarity matrix. However, the matrix can store the distance or dissimilarity between stocks, in which case it is called the distance matrix.

Stock-correlation networks have found applications in different areas including portfolio optimisation and studying the dynamics of the stock market—especially during financial crises. For the remainder of this chapter, we make an overview of how these networks are generated, and what motivated this thesis. We try to keep technical terminology in this chapter to a minimum. However, any term used is going to be defined in the next chapters (most of them in Chapter 2), and the usage of such terms in this chapter is in a manner that it attempts not to hinder grasping the theme and motivation of the thesis.

The steps that are often applied in building stock-correlation networks are:

1. generating the similarity matrix (mentioned above) of stocks
2. converting those similarity values to distance (and obtaining the distance matrix)
3. applying an algorithm to generate a network representing the relationship between stocks.

Sometimes the second step is skipped and the network is generated from the similarity matrix in step one. For this matrix, the most common applied measure to account for similarity is Pearson correlation coefficient. After getting the distance (or similarity) matrix, we can consider this matrix as the adjacency matrix of a complete weighted graph. In this thesis, we denote this complete weighted graph by K_n with a slight abuse of notation. The stock-correlation network is obtained after filtering the edges using an algorithm (step 3 above) on K_n. Rephrased, stock-correlation network refers to a subgraph of K_n. The overall theme of this thesis is basically trying to come up with algorithms that we want to apply in step 3 above to extract this subgraph.

The most common algorithms used to generate the stock-correlation network—or filtration algorithms—are minimum spanning tree (MST), asset graph (AG), planar maximally filtered graph (PMFG), and the threshold method. (These algo-

rithms are explained in detail in Chapter 2.) In all these algorithms. if the edge weights in the complete weighted graph represent distances, they are sorted from the lightest to heaviest. In contrast, if the edge weights denote similarity, they are sorted form the heaviest to the lightest. In the case of MST and PMFG, the network building procedure from the sorted edge list is straightforward: beginning from an empty network, from the top of the list, an edge is added to the network as long as the network remains a tree in MST, or planar in PMFG. In AG, the top N edges are picked where N can be any number, but usually for the sake of comparing with MST, it is $N = n - 1$ where n denotes the number of stocks. Lastly, in the threshold method, only the top edges with weights above a similarity threshold value—or below a distance threshold—are filtered to be included in the network.

A question that arises is what constitutes a "good" filtration algorithm? How should we evaluate the output network of the above-mentioned filtration algorithms? What are the positive aspects of a good stock-correlation network generating algorithm? There is no definite answer in the literature of the area to these questions. However, several criteria have been mentioned such as sparsity, scale-free behaviour, homogeneity of cliques, survival ratio, good clustering, and robustness. Among these criteria, good clustering seems to have attracted the most attention. The manner in which the clustering output of the network is commonly assessed is through its agreement with the economic sectoral classification of stocks.

The main contribution of this thesis is devising new methods and algorithms for building stock-correlation networks to address two aspects: clustering and structural limitation. The overall theme of this thesis is coming up with algorithms of building a stock-correlation network to get a better clustering output compared to previously proposed algorithms. Besides that, we try to avoid the structural limitations of the networks previously proposed in this area and use more information from the complete weighted graph of similarity (or distance) between stocks to build the network.

1.2 Motivation

In the above, we pointed out that the focus of this thesis is proposing stock-correlation network generating algorithms to address the following:

1. structural limitations of the previously proposed algorithms
2. utilise more information from the complete weighted graph to build the network

3. obtain a better clustering of stocks.

Below, we give a brief description of these aspects, and the motivation to devise some methods and algorithms to address them.

1.2.1 Structural Limitations of Previous Works

The most popular algorithms in the literature are arguably MST and PMFG. One of the major features of MST is derived from its correspondence with a well-known hierarchical clustering algorithm (discussed in detail later). However, in this algorithm, a lot of edges from K_n are not included in the network only for the sake of obtaining a tree structure. Same goes for PMFG in the sense that a lot of edges are excluded from the network merely to maintain the planar (defined in Section 2.2.6) structure. One of the appeals of PMFG is that it always contains the MST of its corresponding complete weighted graph. Hence, it provides more information than MST, yet at the same time, contains its underlying structure. However, it appears that the planar structure of such a network does not serve any purpose in the analysis of stock-correlation networks. Consequently, it can be a major limitation that we do not include a lot of edges from K_n in the network just in order to retain its planar structure. In Chapter 3, we propose the proportional degree (PD) algorithm to tackle this structural limitation—especially that of PMFG.

1.2.2 Utilising More Information from the Complete Weighted Graph

MST is apparently the most widely used algorithm for generating stock-correlation networks. As mentioned above, the structural limitation of MST is part of the reason it misses some information provided by edges in K_n. However, besides its tree structure, there is another argument that can be made as to MST not utilising some information to decide on inclusion of an edge in the network. In short, in the process of building MST, if two parts of the graph are disconnected, the lightest possible edge from K_n that can connect them is picked to do so. Once this is done, all other information on the weights of other edges connecting them is lost. By the same token, if the weights in K_n denote similarity, the heaviest one is picked. In Chapter 4, we propose a new algorithm of building a tree in such a manner that more information from K_n is used as to including edges in the tree and specifying their corresponding weight. To this end, we aim to find the tree that matches K_n such that the distance between any two vertices in the tree is close to their distance

in K_n. It needs to be emphasised that the theoretical interest of this algorithm is for approximating any complete weighted graph by a tree. We applied this tree to generate stock-correlation networks, but the problem that it addresses goes beyond the scope of stock-correlation networks, and it is aimed at a broader context.

1.2.3 Better Clustering of Stocks

As discussed earlier, MST is associated with a hierarchical clustering of stocks. To be more precise, MST corresponds in a certain way to a rooted tree that represents a hierarchical structure which is the output of a specific clustering method called single linkage cluster analysis (SLCA). A leaf is a vertex in a tree whose degree is one. The tree which is the output of SLCA has the following features. 1- The root's degree is two. 2- Every other internal vertex has degree three except the leaves. SLCA has the same shortcomings as MST in terms of the extent of using information from K_n to include edges in its output tree. To address this, we want to find a tree whose all vertices except the leaves have degree three such that the distance between any two leaves is close to their distance in K_n. We call this tree (on which all vertices other than the leaves have degree three) a binary tree with a slight abuse of notation. (The slight abuse of notation comes from the fact that this definition is that of an unrooted binary tree. Also, in computer science, the definition of a binary tree is slightly different.) Having said that, in this work, the edge weights cannot be negative so that the resulting binary tree can be comparable with the output of SLCA (explained in Chapter 5). Moreover, since this binary tree is also applied in the area of phylogenetic trees, and there is no consensus regarding whether negative edge weights should exist in a phylogenetic tree, our method can serve a purpose in that area as well. In the end, the clusters in the binary tree have been compared with those in SLCA based on the economic sectoral classification of stocks.

1.3 Thesis Structure

Below is the outline of the chapters in the remainder of the thesis.

Chapter 2 - Background This chapter introduces the concepts, terminology, methods, and algorithms used throughout the thesis alongside the literature review of stock-correlation networks. This chapter consists of two sections. The first section defines the concepts, methods and algorithms used in the later chapters. In

the second section, we provide a comprehensive review of the literature of stock-correlation networks and the pros and cons of some algorithms proposed to build such networks—especially MST and PMFG.

Chapter 3 - Proportional Degree Stock-correlation Network In this chapter, we put forward our algorithm called proportional degree (PD) to generate stock-correlation networks. We give a formal definition of homogeneity of cliques and show that PD demonstrates a higher homogeneity of cliques compared to PMFG. We use Louvain community detection to partition the PD and PMFG networks and see how well each one of them matches the economic sectoral classification of stocks. We do so using the adjusted rand index (ARI) which is a measure of similarity between two partitions. As to clustering, we also use NSC to partition K_n, PMFG, and PD. Then we evaluate how well the partitions of PD and PMFG match that of K_n for different number of clusters using ARI, and demonstrate that for a reasonable number of clusters, PD outperforms PMFG in terms of matching the clusters of K_n. Lastly, we show that PD is more robust than PMFG, meaning, in the absence of some vertices (or edges) the extent to which the clustering performance of PD diminishes is less than that of PMFG.

Chapter 4 - On Finding the Optimal Tree of a Complete Weighted Graph In this chapter, we come up with a new tree structure called the residual sum of squares optimal tree (RSSOT) that uses more information from K_n than MST to generate the tree. One hopes for getting better clusters since this tree uses more information from K_n. We show how to get the edge weights for a given tree based on the tree structure and the RSS criterion. We also propose a very efficient neighbourhood search method to investigate new tree structures and see whether they yield a tree with smaller RSS. Lastly, we propose two metaheuristics: SA and ILS to find the best tree, and we demonstrate which one has a better performance depending on the distance values distribution of K_n, namely, when the dispersion of the distance values in K_n is small, ILS exhibits a significantly better performance than SA. Otherwise, we found no apparent difference between the performances of these two metaheuristics for this problem. The resulting trees of this work have a star-like structure which is not a useful structure to be used as a stock-correlation network.

Chapter 5 - Semi-Labelled Binary Tree Optimisation Subject to Non-Negativity In this chapter, we propose a scheme to find a binary tree (as defined above) where

the path length between any two leaves best estimates their distance in the complete weighted graph K_n. In our running application, the leaves denote stocks, and the internal vertices are unlabelled—as with SLCA. However, in this tree, the edge weights are subject to non-negativity—to make the algorithm comparable to SLCA. As mentioned above, MST is associated with the output tree of SLCA which gives a hierarchical clustering of stocks. We want to compare the clustering performance of our tree with SLCA. In this work, we first demonstrate how to come up with the system of linear equations to find the edge weights—without a non-negativity constraint—of a given binary tree based on the RSS optimality criterion. We also show how we can make use of Cholesky decomposition to find these edge weights. We then demonstrate the nearest neighbourhood interchange (NNI) method which lets us investigate different tree structures. We prove some results that help guide an algorithm attempting to make all edge wegihts non-negative. Based on these proofs, we have devised a scheme based on the ILS framework to find the best tree. We have demonstrated that our tree has a better clustering performance than the output of SLCA as to matching the economic sectoral classification of stocks. However, when we examined other clustering methods, we found that our tree does not have a significant difference in terms of clustering from some simpler methods such as the average linkage cluster analysis (ALCA) and complete linkage cluster analysis (CLCA)—both of which are slightly different to SLCA in terms of their procedure.

Chapter 6 - Concluding Remarks In this chapter, we provide a summary of the research we have done in this area and our key results. We also propose some potential topics for prospective researchers in this area.

1.4 Publications from This Research

Most results from Chapters 3, 4 and 5 of this thesis are included in Hosseini et al. [2021b], Hosseini et al. [2020] and Hosseini and Wormald [2021] respectively. Also, a more detailed version of Hosseini et al. [2020] will appear in Hosseini et al. [2021a].

CHAPTER 2

Background

2.1 Introduction

In order to understand the stock-correlation networks better, we define some related concepts and terminology, and then we review the previous research done in this area.

2.2 Graphs

Graphs are structures that are made up of objects called *vertices* or *nodes* where some pairs of these objects are connected through what are called *edges* or *links*. A common way to denote graphs is by $G(V, E)$ where V denotes the set of vertices and E the set of edges connecting those vertices. The size of a graph refers to the number of vertices in that graph. Hence, the vertices of a graph of size n can be denoted by $V = \{v_1, \ldots, v_n\}$. In such a graph, if every pair of vertices is connected by an edge, this graph is called a complete graph of size n that is usually denoted by K_n. Accordingly, there are $\binom{n}{2} = \frac{n(n-1)}{2}$ edges in this graph. The degree of a vertex v_i in a graph is the number of vertices that are connected to v_i by exactly one edge. In other words, the number of vertices adjacent to v_i determines the degree of v_i. For example, in a complete graph with n vertices, the degree of every vertex is $n - 1$.

If a graph is weighted, it means a value is associated with each of its edges. Wherever in this thesis we mention graph, we mean an unweighted graph unless specified otherwise. What this value—edge weight—denotes depends on the context. For example, if the vertices denote cities, the weight of the edges connecting those cities can be the length of the roads connecting them. In another example, if the graph is modelling a social network, the edge weights can denote the number of emails exchanged between members of that social network. We denote a weighted graph by $G(V, E, w)$ where w denotes a function assigning the weights corresponding to the edges. We need to point out that with a slight abuse of notation, we denote the complete weighted graph by K_n in this thesis.

Lastly, a graph can be directed or undirected. When it is not mentioned, it means that it is undirected. In a directed graph, a direction is associated with each edge. Thus, in such a graph, $e_{ij} \neq e_{ji}$ where e_{ij} denotes the edge connecting vertices v_i and v_j. For example, in the context of a social network, a weighted directed graph can be defined such that w_{ij} denotes the number of emails sent from member denoted by v_i to member v_j.

2.2.1 Adjacency Matrix

The adjacency matrix $M = \left(m_{ij}\right)_{n \times n}$ of a graph $G(V, E, w)$ of size n is a matrix that corresponds to the structure of G such that m_{ij} is the weight of the edge joining v_i and v_j. In an unweighted graph, each edge weight is 1 by default. We have the following properties of an adjacency matrix.

1. M is a binary matrix if G is unweighted.
2. $m_{ij} = 0$ if there is no edge joining v_i and v_j.
3. M is symmetric unless G is directed.

2.2.2 Subgraph

A subgraph is a subset of the vertices of a graph and the edges joining those vertices. Thus, for the graph $G(V, E)$, the graph $G'(V', E')$ is considered a subgraph of G iff $V' \subseteq V$ and $E' \subseteq E$. As such, $\forall e_{ij} \in E'$, $v_i \in V'$ and $v_j \in V'$.

2.2.3 Connected Component

In graph theory, a walk is a sequence of consecutive vertices and edges. That is, if we traverse a graph, we get a walk. If no vertex is repeated on a walk, such a

walk is called a path. (In the context of directed graphs, paths are also directed.) A connected component C of the graph $G(V, E)$ is a maximal subgraph of G where there is at least one path between any two vertices belonging to it.

2.2.4 Clique

A clique C of size m—or an m-clique—in the graph $G(V, E)$ is a subset of vertices such that there is an edge between any two of those vertices. In other words, $\forall v_i, v_j \in C \implies e_{ij} \in E$. Such a clique is maximal if it is not part of a clique with a larger size.

2.2.5 Graph Centre

In the context of graph theory, the distance between two vertices $d(v_i, v_j)$ is the length of the shortest path between them. For an unweighted graph, the length of a path is the number of edges that it contains, and for a weighted graph, it is the total sum of the weights of these edges. The *centre* of a graph $G(V, E)$ is the set of vertices $\mathrm{U} = \underset{v_i}{\operatorname{argmin}} \max_{v_j \in V} d(v_i, v_j)$ In other words, a vertex v_i is a graph centre if its maximum distance from all other vertices is minimal.

2.2.6 Planar Maximally Filtered Graph (PMFG)

A planar graph is one that can be drawn on a plane without any two edges crossing—other than meeting at their end points. In other words, a planar graph can be embedded onto a surface with genus $g = 0$ (a sphere) without any two edges crossing where the genus of a space is the number of "holes" that it has. That is why a sphere has genus $g = 0$.

Figure 2.1: From left to right, surface with genus 0, 1, and 2 (figure taken from Warne [2013, Figure 2.4])

A planar maximally filtered graph (PMFG) is a planar graph where it would not be planar if we added more edges to the graph. For n vertices, if $n \geq 3$, this

graph has the following properties.

1. It has exactly $3n - 6$ edges (see Nishizeki and Chiba [1988, Theorem 1.2]) and Corollary 1.1 that follows from it).
2. It has at most $3n - 8$ 3-cliques and at most $n - 3$ 4-cliques (see Wood [2007, Theorem 5] for an upper bound on the number of cliques in any graph and then Corollary 2 and Proposition 5 in the same paper).
3. It cannot have cliques with size larger than 4 since such a clique cannot be embedded on a plane.

The third property above is based on Kuratowski's Theorem (refer to Even [2011, Section 7.1]).

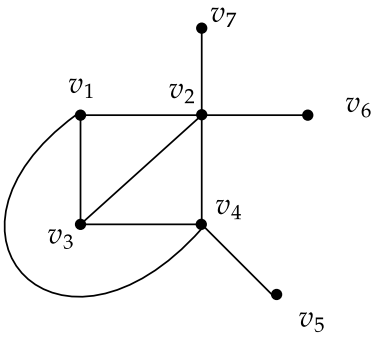


Figure 2.2: An example of a (not maximally) planar graph where there are four 3-cliques: $\{v_1, v_2, v_3\}, \{v_1, v_2, v_4\}, \{v_1, v_3, v_4\}, \{v_2, v_3, v_4\}$, and one 4-clique which is maximal: $\{v_1, v_2, v_3, v_4\}$.

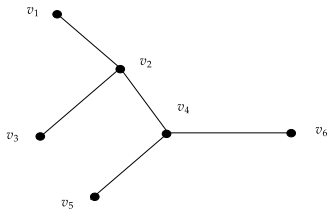
2.2.7 Tree

A tree is an undirected graph which has exactly one path between any two vertices. In a tree, the vertices which have degree 1 are called *leaves* or the *leaf vertices*. All the other vertices which have degree at least 2 are called *internal vertices*. A *subtree* is a tree that is a subgraph of another tree. In a tree, if the degree of every internal vertex is 3, such a tree is an *unrooted binary tree*. With a slight abuse of notation, in this thesis, we call the unrooted binary tree, binary tree. If there is only one internal vertex in a tree, such a tree is called a *star*. Before we mention the properties of a tree, we need to define a *cycle*.

If we traverse a graph such that no vertex is repeated other than the first vertex,

meaning only the end vertex and the start vertex are the same, we get a cycle. Some properties of a tree $T(V, E)$ of size n are as follows.

1. A tree of size n has exactly $n - 1$ edges.
2. A tree is *acyclic*—does not have any cycle.
3. Adding an edge to a tree creates a cycle.
4. Removing a single edge from a tree creates two connected components.
5. A tree has exactly one or two centres. If it has one centre, it is called a *centred tree*. Otherwise, the two centres are connected by an edge.



(a) Binary tree

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

(b) Adjacency matrix

Figure 2.3: An example of a binary tree and its adjacency matrix. In this tree, vertices v_1, v_3, v_5, v_6 are leaves.

Sometimes, one of the vertices of a tree is considered the *root*. Such a tree is a rooted tree. Let the vertex v_0 be the root of the rooted tree T, and let the vertex v_i be on the path from v_0 to v_j. Then v_i is an *ancestor* of v_j and v_j is a *descendent* of v_i. Also, if v_i and v_j are adjacent, v_i is the *parent* of v_j, and v_j is a *child* of v_i. The edges of a rooted tree can be assigned a direction towards the root or from the root towards the leaves. In such cases, the tree is a directed rooted tree. A directed tree is a directed graph that would be a tree if its edges were undirected.

Minimum Spanning Tree (MST)

A *minimum spanning tree* (MST) of the connected weighted graph $G(V, E, w)$ is a subgraph $T(V, E', w')$ which is a tree whose total sum of weights is minimum among all the trees that are subgraphs of G. Thus, it is a tree with the minimum total weight that can span G. There are multiple algorithms to find MST. Two classic and extensively used ones are Kruskal's [Kruskal, 1956] and Prim's [Prim, 1957] algorithms.

Algorithm 1 Kruskal's algorithm

Input:
  $G(V, E, w)$
Output:
  T : the MST

$E' \leftarrow \emptyset$
$w' \leftarrow$ weights corresponding to edges in E'
$E^{sorted} \leftarrow$ list of sorted $e_{ij} \in E$ in order of their corresponding weights w
for e_{ij} in E^{sorted} **do**
    $E' \leftarrow E' \cup \{e_{ij}\}$
    if $G(V, E', w')$ has a cycle **then**
        $E' \leftarrow E' \setminus \{e_{ij}\}$
    if $|E'| = n - 1$ **then**
        break
$T \leftarrow G(V, E', w')$

Kruskal's algorithm is laid out in Algorithm 1. In this algorithm, to find an MST of the connected edge weighted graph $G(V, E, w)$, we sort its edges by their weights in a list from the lightest to the heaviest. Starting from the top of the list, we add the first two edges in the list to an empty graph. We add the next edge on the list to this graph if it does not create a cycle. Otherwise, we skip it and go to the next edge on the list. We continue this process until there are $n - 1$ edges in this graph.

Breadth-first Search (BFS)

Breadth-first search (BFS) is an algorithm for traversing a graph $G(V, E)$. It can be used for traversing a connected or disconnected graph. However, here, we outline how this algorithm traverses a connected graph.

We start from a vertex in the graph which we call the start vertex v_s. In a rooted tree, the start vertex is the root. We define the sequence A such that it includes only v_s. We also define exploring a vertex $v_i \in V$ as follows. We take the neighbours of v_i and add one of them that is not in the sequence A to the end of A. We call a vertex whose all neighbours are in A fully explored. At each step, we explore the first not fully explored vertex in A until all vertices in G are in A. The sequence of vertices in A represents the sequence of vertices traversed.

Depth-first Search (DFS)

As with BFS, *depth-first search* (DFS) is an algorithm for traversing a graph $G(V, E)$. We outline here how this algorithm traverses a connected graph.

The explanation of this algorithm goes the same as that of BFS above. The only difference is the definition of exploring a vertex v_i. In this algorithm, to explore v_i, we take its neighbours and add one of them that is not in the sequence A to the beginning of A rather than the end of A as in BFS.

To make the procedures of BFS and DFS more clear, we apply these two algorithms to traverse the tree in Figure 2.4. We take v_1 as the root. One of the possible sequences of vertices that this tree can be traversed via BFS is $v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}$. Similarly, one of the possible sequences this tree can be traversed via DFS is $v_1, v_2, v_5, v_9, v_6, v_3, v_7, v_{10}, v_4, v_8$.

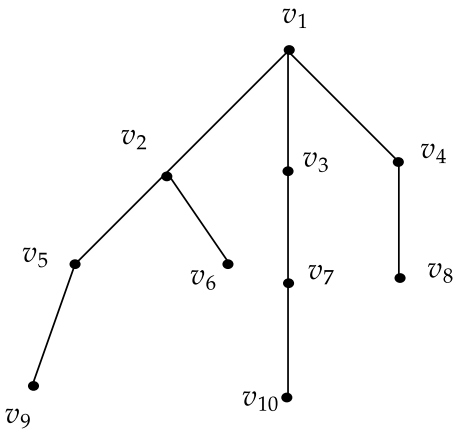


Figure 2.4: An example tree for BFS and DFS to be implemented on

2.3 Metaheuristics

For computable problems, there are procedures that give us the answer in finite time. We call such procedures *algorithm* [Chopard and Tomassini, 2018]. In this context, an instance is a special case of the problem. For example, the set $\{1, 3\}$ is an instance of the set of integer numbers whose summation is 4. In the literature of computer science, computable problems have been classified into the following classes:

1. The time required to *find* the correct answer to the problem is at most as large

as a polynomial of the size of the problem instances.

2. The time required to *verify* the validity of a correct answer is bounded by a polynomial of the size of the problem instances.

The former and the latter classes are called *polynomial* (*P*), and *non-deterministic polynomial* (*NP*) respectively. It can be seen that a P problem is also an NP problem, that is, $P \subseteq NP$ since if the correct solution can be found in polynomial time, it can also be verified in polynomial time.

P problems are usually regarded as "easy" to solve. This is because there are algorithms that can find the correct solution to them such that the time these algorithms require is a polynomial (often of low degree) of the size of the problem. Problems such as finding the largest number in a set, finding the minimum spanning tree of a graph, and finding the row-echelon form of a matrix belong to this class.

However, for NP problems, it is totally different. The time required by the algorithm that looks for the correct solution to these problems can grow exponentially with respect to the size of the problem. This quickly renders exploring all instances of the problem infeasible as the size of the problem becomes larger. There are many problems in this class. One well-known classic example is the knapsack problem. See Garey and Johnson [1979] for a list of some of these problems.

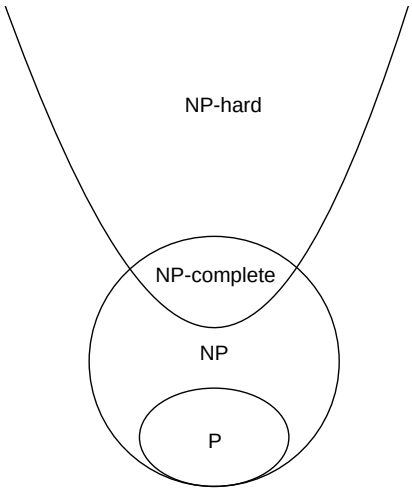


Figure 2.5: Diagram illustrating the relationship between P, NP, NP-complete and NP-hard set of problems

There are also problems that are at least as hard as NP. It means that the time that it takes to find their correct solution grows at least as rapidly that of NP, however, a correct solution to them cannot necessarily be verified in a polynomial

time of the problem size. These problems are called *NP-hard*, and those NP-hard problems which are also in NP are called *NP-complete*. As such, NP-complete is the intersection of NP and NP-hard. Figure 2.5 illustrates the relationship between P, NP, NP-complete and NP-hard problems as explained above.

Now we come to the definition of metaheuristics. Based on the above, it can be seen that exploring the whole search space of an NP-hard problem with a quite large size is most probably infeasible. According to Chopard and Tomassini [2018], metaheuristics are approximation frameworks that achieve acceptable solutions within an acceptable computation time but with no guarantee on the quality of the solution. These frameworks explore the search space using *intensification* and *diversification*. The former and latter refer to exploring the neighbourhood of the current solution and regions in the search space that we hope have not already been explored, respectively. Below, we are going to outline two of these metaheuristics used in this thesis.

2.3.1 Simulated Annealing (SA)

Let $\mathbf{S}$ be the search space of a particular problem. For any admissible solution $x \in \mathbf{S}$ to this problem, let us denote the neighbourhood of x by $N(x)$. Also, let f be the objective function of this problem. We want to find $x^* \in \mathbf{S}$ such that $f(x^*) \leq f(x)$, $\forall x \in \mathbf{S}$. If we investigate $f(x)$ and then $f(x')$ such that $x' \in N(x)$, we call this a move from solution x to x'.

Based on the above, we know that it is most probably wishful thinking to expect to find x^* if the problem discussed is NP-hard. However, we are maybe willing to just find an acceptable solution that we deem "close" enough to the global minimum in this problem. SA tries to achieve this as below.

In simulated annealing (SA), the idea is to allow moves to solutions with objective function values that are worse than the current objective function value to escape from local minima (see Blum and Raidl [2016, Section 1.2.4]). Below, t denotes time steps $t = 0, 1, 2, \ldots$, and $P(.,.)$ specifies a probability that is a function of two variables: t and the difference between the objective function value of the solution x and that of its neighbour x'. Defining x_t as the solution at time t, and starting from $t = 0$, we make the transition from $x_t \leftarrow x$ to $x_{t+1} \leftarrow x'$ where $x' \in N(x)$ in two cases:

1. $f(x') < f(x)$
2. $P(f(x') - f(x), t) > \text{random}(0, 1) \quad \text{if } f(x') > f(x)$.

Otherwise $x_{t+1} \leftarrow x$. In the above, random$(0,1)$ denotes a number picked uniformly at random in the interval $(0,1)$. Case 2 specifies that we accept a solution with a higher objective function value than that of its neighbour with a certain probability. In other words, the larger the value of $f(x') - f(x)$ or that of t, the less probable the transition from $x_t \leftarrow x$ to $x_{t+1} \leftarrow x'$ is. It should be noted that case 2 ensures the method does not get stuck in a local minimum—something that would happen if we had only case 1, which is equivalent to a descent-only algorithm. We continue this procedure for a specific amount of time and pick the solution with the lowest objective function value that we have found.

2.3.2 Iterated Local Search (ILS)

"A heuristic is a method of exploration that exploits some specific aspects of the problem at hand and only applies to it" [Chopard and Tomassini, 2018]. According to Lourenço et al. [2019, p. 580–583], the essence of iterated local search (ILS) is iteratively building a sequence of solutions generated by a heuristic, leading to far better solutions than if one were to use repeated random trials of that heuristic .

As with SA in the above (Section 2.3.1), we want to minimise f, but this time, we make the transition from $x_t \leftarrow x$ to $x_{t+1} \leftarrow x'$ only if $f(x') < f(x)$—so far, it is a descent-only algorithm. However, in contrast to a descent-only algorithm, when we get stuck in a local minimum, we restart the algorithm—by modification of the current local minimum—to a new initial system state. Basically, ILS consists of the following two steps:

1. Modification of the current local minimum in an attempt to kick it far enough from its current basin
2. Descent to get to a new local minimum.

As a final point, we need to point out that it can be perceived from the context of SA and ILS that they can also be applied to maximisation problems.

2.4 Clusters

One of the most extensively investigated features of complex networks is *community structure* or *clustering*. Intuitively, clustering can be defined as grouping objects such that objects in the same group are more similar to one another than to those in other groups. However, Fortunato [2010] mentions that quantitatively, there is no universal consensus on the definition of community in graph clustering. In

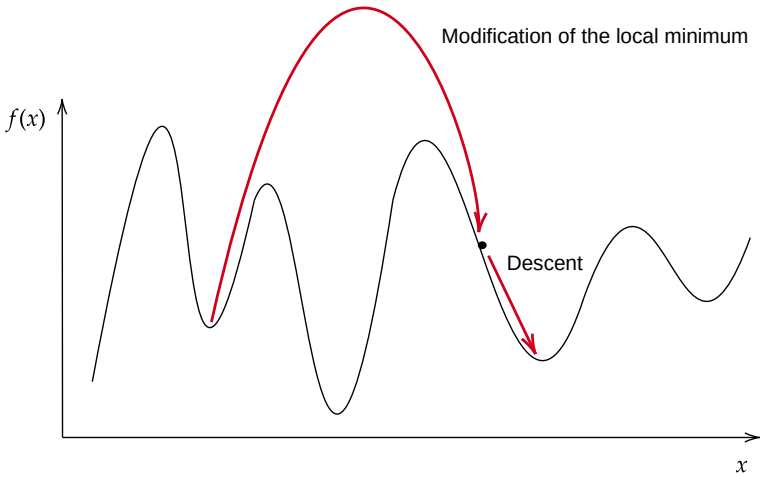


Figure 2.6: An illustration of the two steps of ILS

this context (graphs), clusters may be groups of vertices in which the density of edges inside those groups is considerably larger than the average edge density of the graph (see [Fortunato, 2010, Section 3.2.1]). If each vertex of a graph only belongs to one cluster (no overlapping clusters), such a group of clusters of the graph determines a *partition* of the vertices.

Researchers have proposed different ways for categorising clustering approaches. Fraley and Raftery [1998] divided clustering approaches into hierarchical and partitioning techniques. In the hierarchical approaches, as the term suggests, vertices are clustered in hierarchies: a sequence of partitions each corresponding to a different number of clusters. On the contrary, in partitioning approaches, there is no hierarchical structure—also no overlapping clusters as the name suggests. In hierarchical clustering, if at the beginning all vertices belong to their own unique clusters, and they are merged until a cluster made up of all vertices is formed, such a method of hierarchical clustering is referred to as *agglomerative* clustering. In contrast, if we start from a cluster consisting of all vertices and finish with all vertices belonging to their own unique cluster, this method of hierarchical clustering is called *divisive* hierarchical clustering (See Fraley and Raftery [1998, Section 2.1]). For the remainder of the thesis, wherever we mention hierarchical clustering, we refer to the agglomerative hierarchical clustering .

Saxena et al. [2017] categorise clustering approaches into the following variants: graph theoretic, spectral, model-based, mixed density-based, grid-based, evolutionary approaches based, search-based, collaborative fuzzy, multi-objective, and overlapping community detection. Below, we are going to elaborate on those clustering approaches that we utilised in our works. However, before that, we define some measures that quantify similarity.

2.4.1 Similarity Measures

Similarity is one of the most important concepts in studying clustering. If we want similar objects—or vertices in the context of graphs and networks—to be in the same cluster, we first have to define some measure of similarity. Similarity can also be quantified as distance such that the smaller the distance between two objects—the closer they are—the more similar they are. In some papers and resources, distance is referred to as dissimilarity. Let us denote the distance between two vertices v_i and v_j by d_{ij}. The distance measure is metric if it satisfies the three axioms of a metric space:

1. $d_{ii} = 0 \qquad \forall i$
2. $d_{ij} = d_{ji} \qquad \forall i, j$
3. $d_{ij} + d_{jk} \geq d_{ik} \qquad \forall i, j, k.$

A commonly used family of distance measures is the Minkowski metric (see Sneath et al. [1973, p. 125]). Here, the distance between two objects x_i and x_j is defined as

$$d_{ij} = \|x_i - x_j\|_p = \left(\sum_{k=1}^{q} |x_{ik} - x_{jk}|^p \right)^{\frac{1}{p}} \tag{2.1}$$

in which x_{ik} is the value of the k-th variable for object x_i. We call $\|\boldsymbol{x}\|_p$ the p-norm of vector $\boldsymbol{x}$. The most common value for p in equation (2.1) is 2 which gives the 2-norm distance—or equivalently the Euclidean distance.

Pearson Correlation Coefficient

The *Pearson correlation coefficient* is a measure of linear correlation between two random variables X and Y. This measure is in the interval $[-1, 1]$ where -1, 0 and 1 show respectively perfect anti-correlation, no correlation and perfect correlation.

This measure for the two random variables X and Y is defined as

$$\rho_{XY} = \frac{\mathbf{E}\left[(X - \mathbf{E}\left[X\right])\left(Y - \mathbf{E}\left[Y\right]\right)\right]}{\sqrt{\mathbf{E}\left[\left(X - \mathbf{E}\left[X\right]\right)^2\right]\mathbf{E}\left[\left(Y - \mathbf{E}\left[Y\right]\right)^2\right]}} = \frac{\sigma_{XY}}{\sigma_X \sigma_Y} \tag{2.2}$$

where σ_{XY} is the covariance of the variables X and Y, and σ_X and σ_Y are the standard deviations of X and Y respectively. We will demonstrate in the next chapters that if needed, this measure can be converted into a metric distance. For the rest of the thesis, wherever we mention correlation coefficient we mean the Pearson correlation coefficient.

Mutual Information

Mutual information measures the level of independence between two random variables [Cover and Thomas, 2012] such that a value of 0 shows statistical independence of the random variables. This measure is derived from Shannon's information entropy [Shannon, 2001], a measure that quantifies the uncertainty of a random variable. To define mutual information, first we need to define the *entropy* and *joint entropy* of two discrete random variables X and Y by

$$H(X) = -\sum_i p(x_i) \log_2 p(x_i) \tag{2.3}$$

$$H(X,Y) = -\sum_i \sum_j p(x_i, y_j) \log_2 p(x_i, y_j) \tag{2.4}$$

where $p(x_i)$ and $p(x_i, y_j)$ are the probability distribution and joint probability distribution of X and (X, Y) respectively. Then the mutual information between X and Y can be formulated as

$$I\left(X,Y\right) = H(X) + H(Y) - H(X,Y). \tag{2.5}$$

Unlike correlation coefficient, mutual information is not bounded up by 1. Since large values of mutual information could be hard to interpret without skewing the results, it is useful to use the normalised mutual information (NMI) which brings the values down to the bounded interval $[0, 1]$. One way to normalise mutual information proposed by Kvalseth [1987] is the formula

$$NMI(X,Y) = \frac{2I(X,Y)}{H(X) + H(Y)}. \tag{2.6}$$

It should be said that as done by Kraskov et al. [2005, equation (9)] the mutual information can be converted into a metric by

$$d_{XY} = 1 - \frac{I(X,Y)}{H(X,Y)} \tag{2.7}$$

which gives a value in the interval $[0,1]$. See Li et al. [2001] for a proof that this measure is a metric. Below, we are going to elaborate on the clustering techniques used throughout this thesis—all of which utilise distance or similarity measures.

2.4.2 Hierarchical Clustering

As mentioned before, hierarchical clustering methods group objects into clusters in a hierarchy. We are going to explain three of these methods: *single linkage cluster analysis (SLCA)*, *average linkage cluster analysis (ALCA)*, and *complete linkage cluster analysis (CLCA)*. These three methods apply the same procedure except that they differ in how they determine the distance between clusters in each step of their procedure. We will define SLCA first and show how to perform the other two clustering methods by a simple modification to SLCA.

Let the distance matrix $D = \left(d_{ij}\right)_{n\times n}$ denote the distances between all pairs of n objects. This distance matrix can also be thought as the edge weights of a complete weighted graph K_n in which case the objects can be deemed to be the vertices of that graph. Then the pseudocode of SLCA can be outlined as follows. The output of this algorithm is a hierarchical tree (HT), that is a dendrogram: it illustrates the hierarchy as defined in paragraph 2 of Section 2.4.

Algorithm 2 SLCA algorithm

1: **Input:**
2: $D = \left(d_{ij}\right)_{n\times n}$: the distance matrix
3: **Output:**
4: Hierarchical tree
5:
6: $c_i \leftarrow \{v_i\}$: every vertex belongs to their own cluster
7: **while** number of clusters > 1 **do**
8: Find the closest clusters c_x and c_y such that $d_{c_x,c_y} = \min d_{ij}$ where $v_i \in c_x, v_j \in c_y$
9: $c_{xy} \leftarrow (c_x, c_y)$: Merge clusters c_x and c_y into one cluster c_{xy}
10: Delete the rows and columns corresponding to c_x and c_y from D
11: $d_{c_{xy},c_k} \leftarrow \min\{d_{c_x,c_k}, d_{c_y,c_k}\}$: Distance between the new cluster and the other clusters c_k
12: Add a new row and column d_{c_{xy},c_k} in D corresponding to c_{xy}

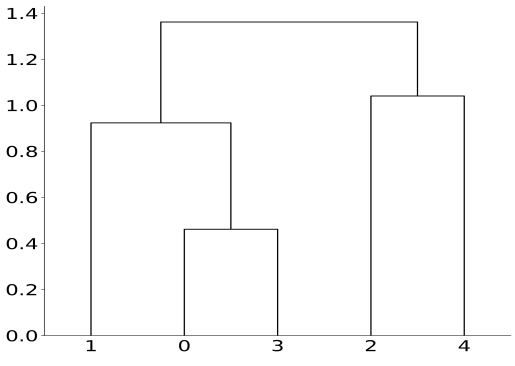
As can be seen in Algorithm 2 (see Everitt et al. [2011, Section 4.2.1] for an illustrative example of this algorithm), at first every vertex belongs to their own cluster. Then the two closest clusters join together to form a bigger cluster. Thus, at each step of this algorithm, we have one less cluster compared to the previous step, and accordingly, for n vertices, we have $n-1$ steps. In order to determine the closest pair of clusters, we first have to specify the distance between clusters at each step. As shown in line 8 of the algorithm, the distance between two clusters c_x and c_y is the minimum distance between all pairs of vertices v_i and v_j such that $v_i \in c_x$ and $v_j \in c_y$. We continue this procedure until the last two clusters join together at step $n-1$. Figure 2.7 is an example of a distance matrix D and its HT obtained via SLCA.

We need to point out that as mentioned by Carlsson and Mémoli [2010], one of the drawbacks of SLCA is what is known as the *chaining phenomenon*. Basically, this means that two clusters are forced to join together because of a single pair of vertices in them being close to one another although all the other pairs of vertices are distant from each other. It can cause a long chain of clusters that are possibly resulting from noise in data. It can be seen that this drawback is related to the point that we made in Section 1.2.2 about how we want to use more information from K_n compared to MST to build a stock-correlation network.

Now we can elaborate on ALCA and CLCA. The only difference between these two algorithms and SLCA is how to determine the distance between clusters in each step—line 11 of Algorithm 2. In the former, the distance between two clusters

	0	1	2	3	4
0	0	0.92	1.68	0.46	1.80
1	0.92	0	1.36	0.93	1.46
2	1.68	1.36	0	1.69	1.04
3	0.46	0.93	1.69	0	1.78
4	1.80	1.46	1.04	1.78	0

(a) Distance matrix



(b) HT by SLCA

Figure 2.7: An example of a distance matrix D and its corresponding HT obtained via SLCA

c_x and c_y is the average distance between all pairs of vertices v_i and v_j such that $v_i \in c_x$ and $v_j \in c_y$. In the latter, the distance between c_x and c_y is the opposite of SLCA, that is, $d_{c_x,c_y} = \max d_{ij}$ where $v_i \in c_x$, $v_j \in c_y$. In this way, these two algorithms can address the chaining phenomenon of SLCA.

(a) SCLA

(b) CLCA

Figure 2.8: An illustration of how SLCA and CLCA determine the distance between each two clusters C_x and C_y in each step of their procedure

2.4.3 Louvain Community Detection

Louvain community detection was first proposed by Blondel et al. [2008] as a partitioning algorithm for networks. This greedy algorithm tries to optimise a function called *modularity* which was first put forward by Newman [2006]. This function assigns a value between -1 and 1 to a partition of a graph as a measure of the density of links inside communities as compared to links between communities. To define this function, let the edge weight s_{ij} in the graph denote the similarity

between vertices v_i and v_j (note that edge weights are not distance). Also, let S be the sum of all similarities (edge weights). We define c_i as the community of vertex v_i, and δ as a simple delta function. Lastly, SW_i and SW_j denote the total sum of the weights of edges adjacent to v_i and v_j respectively. The modularity function is given as below.

$$Q = \frac{1}{2S} \sum_{ij} \left[s_{ij} - \frac{SW_i SW_j}{2S} \right] \delta(c_i, c_j) \tag{2.8}$$

In this algorithm, in the first step, each vertex is in its own community, that is, all the c_i's are distinct. The effect on modularity caused by changing the community of a vertex v_i to that of each of its neighbours in turn is checked. Then the community of vertex v_i is reassigned to the community of the neighbour vertex that leads to the largest increase in modularity. In the case of no increase in modularity, v_i keeps its own community label. This process is applied to all vertices and repeated until the community reassignment of none of the vertices leads to an increase in Q. In the second step, all the vertices belonging to the same community are considered as a single vertex, and the edges between them are denoted by self loops on the new vertex. Also, all edges from vertices of the same community in the previous step to vertices in another community are denoted by a single weighted edge between those communities. These two steps are repeated iteratively until there is no change in the community assignment of the vertices in step one.

That being said, depending on the order of vertices evaluated by this algorithm, we can get different resulting partitions corresponding to different modularity function values. Accordingly, this algorithm does not necessarily yield the global maximum modularity. It is also worth mentioning that finding the exact maximum modularity is an NP-hard problem (see Brandes et al. [2007] and in particular Theorems 4.4 and 4.8 for proof), and one does not hope for an algorithm to solve it.

2.4.4 Normalised Spectral Clustering (NSC)

Dimensionality reduction refers to techniques that find applications in different areas including clustering. *Spectral clustering* refers to some clustering techniques that make use of the eigenvalues and eigenvectors of the similarity matrix to perform dimensionality reduction and then clustering. See Von Luxburg [2007] for an overview of these techniques. *Normalised spectral clustering* (NSC) is a group of spectral clustering techniques. Here, we describe an NSC algorithm proposed by Shi and Malik [2000]. For the remainder of the thesis wherever we mention

NSC, we are referring to this algorithm. Before outlining this algorithm, we have to define a classic and popular clustering technique called *k-means*.

The k-means clustering problem was first proposed by MacQueen et al. [1967]. Let us say we want to partition N objects $x_1, \ldots, x_N$ into k clusters $C_1, \ldots, C_k$. The objective of k-means is to find the cluster centres $m(C_1), \ldots, m(C_k)$ such that the sum of 2-norm or Euclidean distances (as defined in Section 2.4.1) squared between each object x_i and its closest cluster centre $m(C_j)$ is minimised. That is, we want to find the cluster centres as

$$\underset{m(C_1),\ldots,m(C_k)}{\operatorname{argmin}} \sum_{i=1}^{N} \min_{j\in\{1,\ldots,k\}} \left(\frac{1}{2} \left\| x_i - m(C_j) \right\|_2^2 \right). \tag{2.9}$$

This is an NP-hard problem as proved in Aloise et al. [2009]. Lloyd [1982] proposed an algorithm for this problem for the first time, and it is widely known as "the k-means algorithm" or Lloyd's algorithm, and the k-means clustering technique usually refers to clustering using this algorithm. To explain this algorithm, let $m^t(C_1), \ldots, m^t(C_k)$ denote the cluster centres at iteration t. The pseudocode of this algorithm is as follows.

Algorithm 3 k-means algorithm

Input:
  $(x_1, \ldots, x_N)$: N data points
  k: number of clusters
Output:
  $C_1, \ldots, C_k$: clusters

$t = 0$
$m^t(C_1), \ldots, m^t(C_k)$: initial cluster centres
while True **do**
    Step 1: Assign each x_i for $i = 1, \ldots, N$ to cluster C_j such that $m^t(C_j)$ is nearest to x_i using the Euclidean distance
    Step 2: Compute $m^{t+1}(C_j)$ for $j = 1, \ldots, k$ as the mean of all points assigned to the cluster C_j
    if $m^t(C_j) = m^{t+1}(C_j) \quad \forall j = 1, \ldots, k$ **then**
        break
    else
        $t \leftarrow t + 1$

Now we come to the description of NSC. This algorithm takes the similarity matrix and the number of clusters k as its inputs and partitions the data set as

below.

Algorithm 4 NSC algorithm

Input:
  $W = (w_{ij})_{i,j=1,2,\ldots,n}$: similarity matrix
  k: number of clusters
Output:
  $\{C_1, \ldots, C_k\}$: clusters

D : digonal matrix with $d_i = \sum_{j=1}^{n} w_{ij}$, $i = 1, 2, \ldots, n$
$L = D - W$: Laplacian matrix
$\boldsymbol{\nu}_1, \boldsymbol{\nu}_2, \ldots, \boldsymbol{\nu}_k \leftarrow$ eigenvectors of the k smallest eigenvalues of the eigenproblem $L\boldsymbol{\nu} = \lambda D\boldsymbol{\nu}$
$V \in \mathbb{R}^{n \times k} \leftarrow$ matrix with $\boldsymbol{\nu}_1, \boldsymbol{\nu}_2, \ldots, \boldsymbol{\nu}_k$ as columns
$\boldsymbol{y}_i \in \mathbb{R}^k \leftarrow$ corresponding vector to the i-th row of V for $i = 1, 2, \ldots, n$
$C_1, C_2, \ldots, C_k \leftarrow$ clusters of $\boldsymbol{y}_i \in \mathbb{R}^k$, $i = 1, 2, \ldots, n$ by k-means algorithm

But what is a good choice of k? One tool to answer this question is the eigengap heuristic [Von Luxburg, 2007]. Denoting sorted (from small to large) eigenvalues of Laplacian L of the similarity matrix as $\lambda_1, \lambda_2, \ldots, \lambda_n$, eigengap heuristic states that the network should be divided into k clusters such that λ_{k+1} is significantly larger than $\lambda_1, \ldots, \lambda_k$. In other words, if the largest gap is between λ_k and λ_{k+1} for $k = 1, 2, \ldots, n-1$ in the sorted eigenvalues of the Laplacian of the similarity matrix, we divide the network into k clusters.

2.4.5 Adjusted Rand Index (ARI)

The *Rand index* [Rand, 1971] is a measure in statistics that quantifies the similarity between two partitions of a data set. The *adjusted rand index (ARI)* is another version of the Rand index that is corrected for chance. Given two partitions A and B of a set S containing n elements, the ARI of $A = \{A_1, A_2, \ldots, A_r\}$ and $B = \{B_1, B_2, \ldots, B_s\}$ is as given by

$$ARI = \frac{\sum_{ij} \binom{n_{ij}}{2} - \frac{\left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right]}{\binom{n}{2}}}{\frac{1}{2}\left[\sum_i \binom{a_i}{2} + \sum_i \binom{a_i}{2}\right] - \frac{\left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right]}{\binom{n}{2}}} \tag{2.10}$$

where $n_{ij} = |A_i \cap B_j|$, $a_i = \sum_{j=1}^{s} n_{ij}$, and $b_j = \sum_{i=1}^{r} n_{ij}$. An ARI of 1 shows identical clusters, 0 shows random assignment to clusters, and negative values show that the similarity between the two partitions is less than expected from random assignment.

2.5 Positive Definite Matrix

A *positive definite* matrix is a special type of symmetric matrix all of whose eigenvalues are positive. If all eigenvalues are non-negative, it is *positive semi-definite*. There are other ways to prove that a matrix is positive definite other than showing all its eigenvalues are positive. Before that, we need to prove the lemmas below.

Lemma 2.5.1. *The eigenvectors of the symmetric matrix $A_{n\times n}$ are mutually orthogonal.*

Proof. Let $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ be two eigenvectors of A respectively corresponding to distinct eigenvalues λ_1 and λ_2.

$$\begin{aligned}\lambda_1\langle \boldsymbol{v}_1, \boldsymbol{v}_2\rangle &= \langle \lambda_1\boldsymbol{v}_1, \boldsymbol{v}_2\rangle = \langle A\boldsymbol{v}_1, \boldsymbol{v}_2\rangle = \langle \boldsymbol{v}_1, A^\intercal \boldsymbol{v}_2\rangle \\ &= \langle \boldsymbol{v}_1, A\boldsymbol{v}_2\rangle = \langle \boldsymbol{v}_1, \lambda_2\boldsymbol{v}_2\rangle = \lambda_2\langle \boldsymbol{v}_1, \boldsymbol{v}_2\rangle\end{aligned}$$

Thus, $(\lambda_1 - \lambda_2)\langle \boldsymbol{v}_1, \boldsymbol{v}_2\rangle = 0$. Since λ_1 and λ_2 are distinct, $\lambda_1 - \lambda_2 \neq 0 \implies \langle \boldsymbol{v}_1, \boldsymbol{v}_2\rangle = 0 \implies \boldsymbol{v}_1 \perp \boldsymbol{v}_2$. □

Lemma 2.5.2. *If A is positive definite, and $\boldsymbol{v}$ is an eigenvector of A, $\boldsymbol{v}^\intercal A\boldsymbol{v} > 0$.*

Proof. Let λ be the eigenvalue corresponding to $\boldsymbol{v}$. Since A is positive definite, $\lambda > 0$. Then we have $A\boldsymbol{v} = \lambda\boldsymbol{v} \implies \boldsymbol{v}^\intercal A\boldsymbol{v} = \lambda\boldsymbol{v}^\intercal\boldsymbol{v} > 0$. □

Every real symmetric $n \times n$ matrix has n eigenvectors, and based on Lemma 2.5.1, we can infer that the eigenvectors of a real symmetric matrix span $\mathbb{R}^n$. Accordingly, the eigenvector $\boldsymbol{v}$ in Lemma 2.5.2 can be substituted by any other vector $\boldsymbol{x} \in \mathbb{R}^n$, and the ineqaulity $\boldsymbol{x}^\intercal A\boldsymbol{x} > 0$ still holds. This is another definition of a positive definite matrix, that is, the symmetric matrix A is positive definite if $\boldsymbol{x}^\intercal A\boldsymbol{x} > 0$ for all $x \neq \mathbf{0}$.

Finally, we can use the following lemma to show a matrix is positive definite.

Lemma 2.5.3. *A is positive definite if $A = B^\intercal B$ where B is a matrix whose columns are independent.*

Proof. Since the columns of B are independent, $B\boldsymbol{x} = 0$ implies that $\boldsymbol{x} = 0$. So $\boldsymbol{x}^\intercal A\boldsymbol{x} = (B\boldsymbol{x})^\intercal (B\boldsymbol{x}) > 0$ for any $\boldsymbol{x} \neq 0$. □

2.6 Cholesky Decomposition

Cholesky decomposition is a decomposition of a positive definite matrix. This decomposition has several applications, one of the most important of which is solving systems of linear equations. In this context, it is more efficient than the well-known LU decomposition (see the last paragraph of Press et al. [2007, p. 100]) for a brief explanation on why). The Cholesky decomposition of a positive definite matrix A is in the form $A = LL^\intercal$ where L is a unique lower triangular matrix whose entries are calculated by equations (2.11) and (2.12). As such, to solve the equation $A\mathbf{w} = \mathbf{d}$, we can solve $Ly = \mathbf{d}$, and then $L^\intercal \mathbf{w} = y$ to find $\mathbf{w}$.

$$L_{ii} = \sqrt{A_{ii} - \sum_{k=1}^{i-1} L_{ik}^2} \tag{2.11}$$

$$L_{ij} = \frac{1}{L_{jj}} \left(A_{ij} - \sum_{k=1}^{j-1} L_{ik} L_{jk} \right) \tag{2.12}$$

2.7 Stock-correlation Network

One kind of financial network is a stock-correlation network. In such a network, vertices denote stocks and the weight of an edge between two stocks shows the similarity between them. Similarity could be, for example, the influence the stocks have over the price of each other. Correlation coefficient is one of the most commonly used measures to account for similarity in stock networks. (We will explain in the next chapters what random variables we use in order to apply different similarity measures in this area: stock-correlation networks.) That being said, including all the cross correlations between stocks would create a complete weighted graph that reflects the complexity through a densely interwoven structure. Because of this, several algorithms have been proposed to filter the complete weighted graph into a simple subgraph to use as a representation of the original network. Some of these algorithms are minimum spanning tree (MST) [Mantegna, 1999; Bonanno et al., 2003; Tabak et al., 2010; Fiedor, 2014; Guo et al., 2018b], asset graph [Onnela et al., 2003b], planar maximally filtered graph (PMFG) [Tumminello et al., 2005,

2007b; Song et al., 2011a; Fiedor, 2014; Wang and Xie, 2015; Wang et al., 2017], and correlation threshold method [Boginski et al., 2005; Huang et al., 2009; Chi et al., 2010; Namaki et al., 2011]. See Birch et al. [2016] for an advantages-and-limitations comparison of MST, AG, and PMFG on a dataset. The question is, what exactly makes a filtering algorithm better than the others? There is no unique answer, but to get a perspective, let us take a quick look at the reported positive aspects of the above-mentioned methods.

Mantegna [1999] attributed the advantage of MST to the fact that it provided a hierarchical clustering of stocks. Onnela et al. [2003b] demonstrated the advantage of AG by observing that it had a higher survival ratio (ratio of common edges existing in two consecutive time periods) compared with MST. Yet they also mentioned that unlike MST, there was not an evident scale-free behaviour indicating that the degree distribution is power law for AG. (The power law distribution of the degrees of vertices is discussed since this property has been previously demonstrated in a lot of complex networks [Albert and Barabási, 2002; Dorogovtsev and Mendes, 2002, 2001; Szabó et al., 2003; Marsili, 2002; Yang et al., 2003]). Overall, they found AG better in terms of being less fragile in the presence of a market crisis, and that it incorporated more information from the original complete weighted graph compared to MST, for it did not have the structural limitations of MST. Tumminello et al. [2005] attributed the usefulness of PMFG to the fact that the network produced always contains the one produced by MST, and that it contains cliques in it with stocks in those cliques mostly belonging to the same economic sectors. Boginski et al. [2005] mentioned that the correlation threshold method was useful since for a large enough minimum threshold on the value of correlation coefficient, their network had a scale-free behaviour, and they could classify financial instruments through the analysis of cliques and independent sets (discussed later).

Others have also discussed advantages of the above algorithms. Huang et al. [2009] argued that the correlation threshold method displayed robustness against random vertex failures and a high average clustering coefficient. One of the points that Wang et al. [2017] made is that PMFG is useful because it provided a good clustering of the stocks according to the economic sectoral benchmark clustering.

In summary, the positive aspects of filtering algorithms considered so far in the literature are sparsity, scale-free behaviour, homogeneity of cliques, survival ratio, good clustering, and robustness. Of these, the clustering behaviour seems to attract the most attention. We next review the above mentioned methods and explore their properties especially in terms of clustering.

2.7.1 MST Stock-correlation Network

The first paper in the stock-correlation networks area was published by Mantegna [1999]. In their work, they took the pairwise correlation coefficients of the *log*-returns of stocks (explained in the next chapter) and converted those values to distances. Then they built an MST network based on the distance matrix D derived from the distances between stocks. In the area of stock-correlation networks, this is referred to as the MST algorithm—although in graph theory MST is not the name of an algorithm. They showed that the hierarchical clustering (SLCA) of stocks associated with MST agrees reasonably well with the economic sectoral classification of stocks.

The question that comes to mind is how is SLCA associated with MST? Let $T(V, E)$ be the MST corresponding to D. Also, let $d_{ij}^{<}$ be the length of the longest edge on the path in T with the start vertex of v_i and the end vertex of v_j. The value $d_{ij}^{<}$ is the *ultrametric* distance between v_i and v_j. The ultrametric space has the same three properties of a metric space plus the ultrametric inequality: $d_{ik}^{<} \leq \max\{d_{ij}^{<}, d_{jk}^{<}\}$. If we get the SLCA of $D^{<} = \left(d_{ij}^{<}\right)_{n\times n}$, it is going to be the same as that of D. Thus, we can get the SLCA of D from the MST of D without knowing D, and this is how SLCA is associated with MST. For the remainder of the thesis, wherever we mention the hierarchical structure in an MST, we mean the hierarchical structure that is associated with the SLCA of that MST. See Mantegna and Stanley [2000] for a detailed discussion of ultrametricity and analysing complexity in finance.

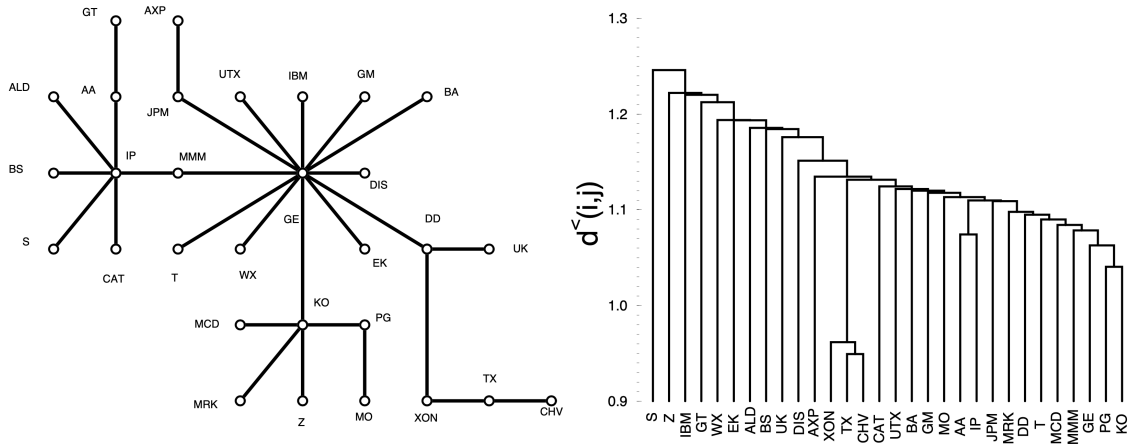


Figure 2.9: MST and its corresponding SLCA obtained from the 30 stocks used to calculate the Dow Jones Industrial Average. We can see certain stocks in the same sub-sectors clustered together: XON, TX and CHV are oil companies, AA and IP are raw materials companies, and PG and KO are consumer non-durables companies (figure taken from Mantegna [1999, Fig. 1])

The hierarchical structure of stocks in MST have been the focus of a lot of past research in the stock-correlation networks area [Bonanno et al., 2003; Brida and Risso, 2010; Coletti, 2016; Garas and Argyrakis, 2007; Nobi et al., 2015; Tabak et al., 2010; Wang and Xie, 2015]. Brida and Risso [2010] discussed that in their MST—as it can be seen in Figure 2.10—stocks of companies with strong interrelationships or similar production activities cluster together which can be seen in the leaves of the HT (described in Section 2.4.2) that corresponds to their MST. Coletti [2016] discussed how well the clusters they identified in MST (and equivalently SLCA) match the economic sectors of the stocks in them. They proposed using extra information such as the volume traded to obtain the correlation between stocks and a better clustering of them in MST. Tabak et al. [2010] pointed out that stocks belonging to the same sector tend to cluster together in MST. They further discussed the relative importance of different sectors in the stock-correlation network. Lastly, Wang and Xie [2015] used MST to analyse international real-estate securities. They demonstrated that in the HT corresponding to MST, securities are clustered based their region.

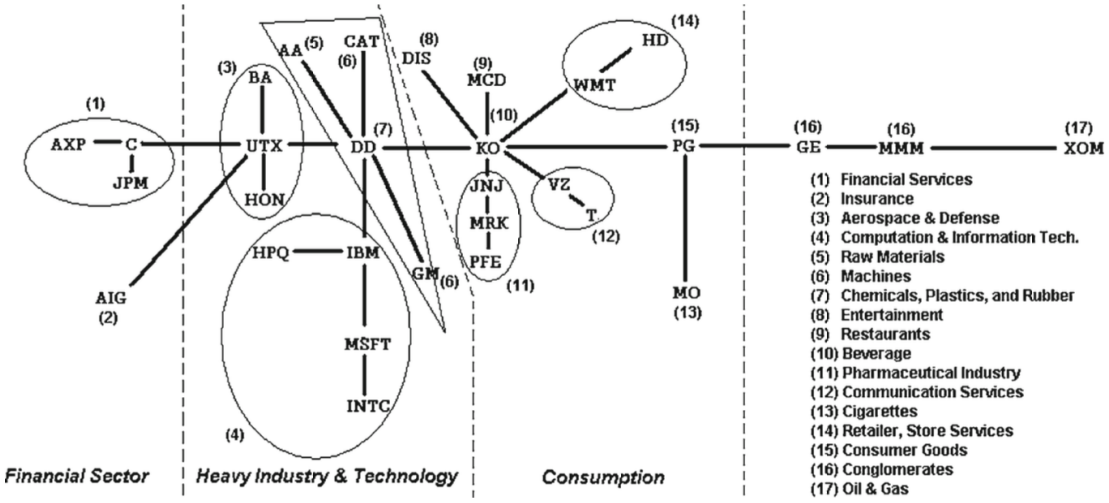


Figure 2.10: The MST stock-correlation network represented in Brida and Risso [2010, Fig. 1].

Besides the above, researchers have also utilised MST to evaluate risk and optimise portfolio allocation [Kaya, 2013; Onnela et al., 2002, 2003c]. Kaya [2013] showed that stocks that are located towards the centre of the MST network tend to have higher returns. They also offered an investment strategy that utilises network centrality information for portfolio allocation. Markowitz [1952] proposed a portfolio optimisation approach (see Rubinstein [2002] for a brief introduction to this approach). Onnela et al. [2002, 2003c] demonstrated that the assets of the

optimal *Markowitz* portfolio lie practically at all times on the outskirts of the MST stock-correlation network.

The other application of MST stock-correlation networks has been studying the market's dynamics, especially during financial crises [Han et al., 2019; Majapa and Gossel, 2016; Onnela et al., 2003a, 2002, 2003c; Song et al., 2011a; Nobi et al., 2015; Zhang et al., 2011]. For example, Nobi et al. [2015] mentioned that the hierarchical structure of stocks increases during financial crises, and Zhang et al. [2011] claimed that a chain-like MST structure is associated with a high-volatility economic crisis.

Overall, the use of the MST algorithm in this area has been extensive. In addition to the above, researchers have studied the power law distribution of degrees in the MST stock-correlation network [Bonanno et al., 2003; Onnela et al., 2003c; Wang and Xie, 2015]. (As mentioned in the second paragraph of Section 2.7, the power law distribution of degrees is one of the properties of many complex networks). They have also used variants of the MST network such as the forest (not the same forest as the term *forest* in graph theory) of all possible MSTs [Gan and Djauhari, 2015], and a combination of spectral analysis and maximal spanning tree [Heimo et al., 2007] to study the stock market. It should be said that the maximal spanning tree is based on the same idea as MST where edge weights denote similarity instead of distance. The procedure to build such trees is the same as the minimum spanning tree, but instead of sorting edge weights from small to large, they are sorted in reverse since similarity is essentially the inverse of distance.

Another type of spanning tree called the *average linkage minimum spanning tree* (ALMST) has been proposed by Tumminello et al. [2007a] to build a stock-correlation network. ALMST is associated with ALCA (described in the last paragraph of Section 2.4.2). In order to explain ALMST, let us first outline another algorithm to build MST. Let D be the distance matrix of n stocks. We start with the empty graph G—which means it has no edges—consisting of n vertices denoting the stocks. The algorithm for generating MST goes as follows.

Algorithm 5 MST algorithm

1: Set Q as the matrix of elements q_{ij} such that $Q = D$.
2: Select the minimum distance q_{hk} between elements belonging to different connected components S_h and S_k in G.
3: Find elements u, p such that $d_{up} = \min\{d_{ij}, \forall v_i \in S_h \text{ and } \forall v_j \in S_k\}$.
4: Add to G the edge between vertices v_u and v_p with weight d_{up}. Once the edge is added to G, v_u and v_p will belong to the same connected component $S = S_h \bigcup S_k$.
5: $q_{ij} = \begin{cases} q_{hk} & \text{if } v_i \in S_h \text{ and } v_j \in S_k \\ \min\{q_{pt} : v_p \in S, v_t \in S_j, \text{ and } S_j \neq S\} & \text{if } v_i \in S \text{ and } v_j \in S_j \\ q_{ij} & \text{otherwise} \end{cases}$
6: If G is still disconnected, go to step 2; otherwise, stop.

This algorithm gives MST—that is G—which is also as discussed in the beginning of this section associated with SLCA, and $d_{ij}^{<} = \sqrt{2(1 - q_{ij})}$ gives the ultrametric distance between v_i and v_j in this MST. If we substitute the entries of Q in line 5 in the algorithm above by

$$q_{ij} = \begin{cases} q_{hk} & \text{if } v_i \in S_h \text{ and } v_j \in S_k \\ \text{mean}\{q_{pt} : v_p \in S, v_t \in S_j, \text{ and } S_j \neq S\} & \text{if } v_i \in S \text{ and } v_j \in S_j \\ q_{ij} & \text{otherwise} \end{cases}$$

, then we are performing ALCA, and the graph G we obtain is ALMST.

Tumminello et al. [2007a] also used a resampling technique called *bootstrap* in the following manner. Let $M_{t \times n}$ denote the matrix with columns denoting stocks, rows denoting time steps, and entries denoting the stock prices. We randomly sample t rows from M with repetition, form the distance matrix D, and based on that, we build MST and ALMST. We repeat this procedure a large number of times. The fraction of times that the edge e_{ij} appears in MST and ALMST denotes its *reliability* in each one of them. They show that the reliability of edges in ALMST is lower than that of MST. However, ALMST gives a better clustering of stocks with respect to economic sectors and sub-sectors compared to MST.

2.7.2 PMFG Stock-correlation Network

The PMFG stock-correlation network was first proposed by Tumminello et al. [2005]. They showed that a significant proportion of the 4-cliques and 3-cliques of the network consist of stocks all belonging to the same economic sector. They re-

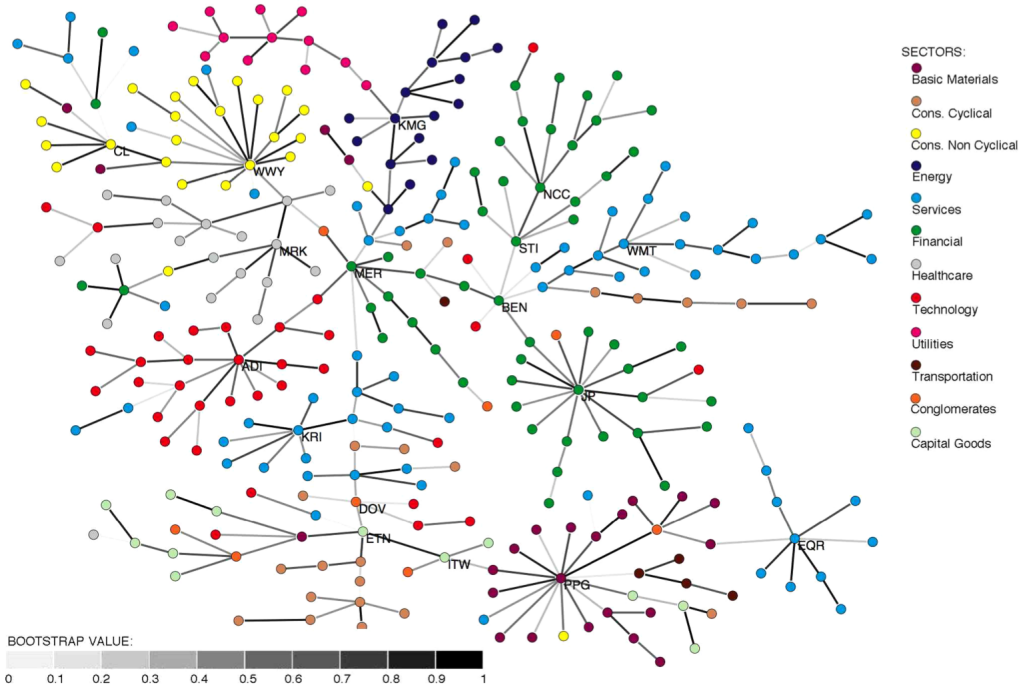


Figure 2.11: An ALMST where the reliability of its edges is denoted by their thickness (figure taken from Tumminello et al. [2007a, Fig. 1])

ferred to it as high homogeneity of cliques in PMFG. As mentioned in Sections 2.2.6 and 2.2.7, for $n \geq 3$ vertices (stocks in this context), PMFG and MST have $3n - 6$ and $n - 1$ edges respectively. Since PMFG has more edges than MST, it contains more details than MST and ALMST regarding the sample correlation matrix [Tumminello et al., 2010]. Yet it has been argued in other papers [Birch et al., 2016] that a downside of PMFG is not directly yielding the hierarchical clustering of stocks like MST. The PMFG construction algorithm used by Tumminello et al. [2005] and others in the area of stock-correlation network is a straight forward algorithm as below.

Algorithm 6 PMFG algorithm

Input:
  V : set of stocks
  D : distance matrix
Output:
  $G(V, E)$: PMFG network

$E \leftarrow \emptyset$
$G(V, E) \leftarrow$ network of stocks V with no edges
$d^{sorted} \leftarrow$ list of (i, j, d_{ij}) $(i, j \in V\,,\ i \neq j)$, sorted in ascending order
for (i, j, s_{ij}) in d^{sorted} **do**
    $E \leftarrow E \cup \{e_{ij}\}$
    if G is not planar **then**
        $E \leftarrow E - \{e_{ij}\}$
        if $|E| \geq 3|V| - 6$ **then**
            Break

Tumminello et al. [2005] also proved that MST is a subgraph of PMFG, and they referred to it as one of the positive aspects of PMFG. Their proof is as follows. Let d^{sorted} denote the list of pairwise distances between vertices sorted from small to large. We also have to define *bridge*: An edge is a bridge if and only if there is no path between two vertices in the graph in the absence of that edge. Tumminello et al. [2005] used the proposition that adding a bridge to a graph does not change its genus which they inferred from Miller [1987]. This is the basis of the proof. Let G_m and T_m denote the PMFG and MST at step m of their construction, that is, we are on the *m-th* element on the list d^{sorted} in Kruskal's algorithm (Algorithm 1) for MST and Algorithm 6 for PMFG. We say $G_m \equiv T_m$ in the following case: If two vertices are connected in one of the graphs by at least one path, they are also connected in the other one by at least one path. We know that $G_2 \equiv T_2$. If $G_m \equiv T_m$, we have the following four cases for G_{m+1} and T_{m+1}:

1. The new edge e_{ij} is a bridge in T_{m+1}. This implies that e_{ij} is a bridge in G_{m+1} also. So e_{ij} is added to both graphs.
2. The new edge e_{ij} is a bridge in G_{m+1}. This implies that e_{ij} is a bridge in T_{m+1} also. So e_{ij} is added to both graphs.
3. The new edge e_{ij} is not a bridge in either T_{m+1} or G_{m+1}, and G_{m+1} is not planar. So e_{ij} is not added to either of the graphs.
4. The new edge e_{ij} is not a bridge in either T_{m+1} or G_{m+1}, and G_{m+1} is planar. So e_{ij} is added to G_{m+1} but not to T_{m+1}.

Hence, $G_{m+1} \equiv T_{m+1}$. We can see by induction that if a new edge is added to MST in its construction, it is also added to PMFG. Otherwise, the new edge is either not added to any of them or only to PMFG.

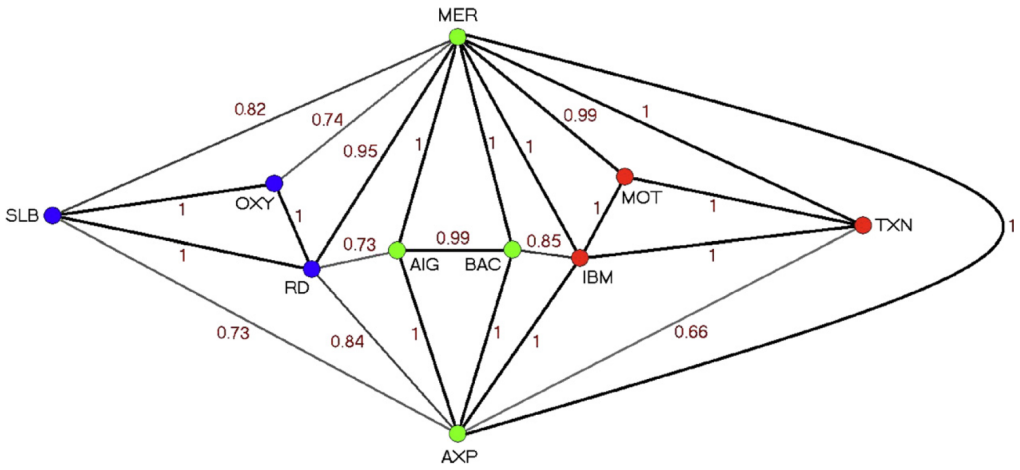


Figure 2.12: An example of a PMFG stock-correlation network. The vertices with the same color belong to the same economic sector, and the thickness of the edges denote their bootstrap reliability as described above for ALMST (figure taken from Tumminello et al. [2010, Fig. 4]).

PMFG has been used extensively in the literature to study the dynamics of the stock market [Buccheri et al., 2013; Song et al., 2011a; Wen et al., 2019], study markets during financial crises [Wang and Xie, 2015] and investigate clusters and their associated economic sectors [Wang et al., 2017]. Almost all these studies investigate the structure of the network with respect to the economic sectors and sub-sectors. Thus, we can perceive the importance of clustering in this area once again. There are also studies that have used PMFG to generate stock-correlation networks [Wang et al., 2018; You et al., 2015; Kenett et al., 2015] but using measures of correlation other than correlation coefficient. The focus of these studies has been comparing the difference that different similarity measures make rather than the stock-correlation network generating algorithms.

2.7.3 The Threshold Method

The threshold method was first proposed by Boginski et al. [2005]. Let the distance or similarity matrix be the adjacency matrix of a complete weighted graph K_n. In this method, the stock-correlation network is a subgraph of K_n such that the weight of every edge in this subgraph is below a specific threshold if the edge weights denote distance, and over a specific threshold if the edge weights denote similarity. Boginski et al. [2005] put a strong emphasis on the power law distribution of

degrees, cliques and independent sets (explained below).

Boginski et al. [2005] observed that for a subgraph of K_n whose edge weights are above a certain threshold for values of correlation coefficient, the stock-correlation network demonstrates a scale-free behaviour (vertex degrees follow a power law distribution as explained in Section 2.7). Without diving into much details, clustering coefficient denotes the probability that two neighbours of a vertex are connected through an edge. Boginski et al. [2005] also observed that such a network has a high clustering coefficient, a property that is evident in many other complex networks.

Lastly, they studied the stock-correlation network with respect to its cliques and largest *independent sets*—defined as follows. An independent set is a set of vertices such that no two of them are adjacent. They used independent sets to find stocks that are less correlated, to diversify a portfolio. By the same token, they studied cliques to find stocks that are highly correlated which means that they have a similar price action. They saw that often the size of the independent sets were small, which translates to finding a "completely diversified" portfolio not being an easy task.

The threshold method has been applied by several other researchers to generate stock-correlation networks. Huang et al. [2009] studied the same aspects of network as those of Boginski et al. [2005] besides the topological stability of the network. They pointed out that the threshold method generated stock-correlation network is robust against random vertex failure. From this, they inferred that it is useful for portfolio management. Chi et al. [2010] used the threshold method to study the stock market dynamics. We can notice in these studies that as with PMFG, there has been a focus on the study of cliques although apparently mostly for portfolio management.

2.7.4 Asset Graph

The asset graph was proposed by Onnela et al. [2003b]. Let us specify the most important edges as the heaviest in terms of weight if the weight denotes similarity, and the lightest if it denotes distance. In the asset graph, we include the top k most important edges in the network. We do so regardless of the structure of the graph in contrast to MST and PMFG. The fact that this graph lacks any structural constraint is considered a desirable aspect of this method by Onnela et al. [2003b]. They set the value of k equal to $n - 1$ (where n is the number of stocks) to compare the asset graph with MST. They argued that MST is forced to "accept" edge weights—

weights denote distance—that are quite long only for the sake of retaining the tree structure. In light of that, the asset graph contains more relevant information compared to its MST counterpart.

Onnela et al. [2003b] also argued that the asset graph has a higher survival ratio than MST where survival ratio is defined as follows. If we build the stock-correlation network for different periods in a time window and denote the network at time periods m and $m+1$ by $G_m(V, E_m)$ and $G_{m+1}(V, E_{m+1})$ respectively, the survival ratio is defined as $\frac{|E_m \bigcap E_{m+1}|}{N}$ where $|E_m| = |E_{m+1}| = N$. Basically, it denotes the fraction of edges that survive from one time period to the other. As discussed before, for MST, $N = |V| - 1$. Onnela et al. [2003b] argued that considering this measure, the asset graph is more robust than MST. Lastly, Onnela et al. [2003b] pointed out that unlike MST, the power law distribution of degrees is not evident in the asset graph.

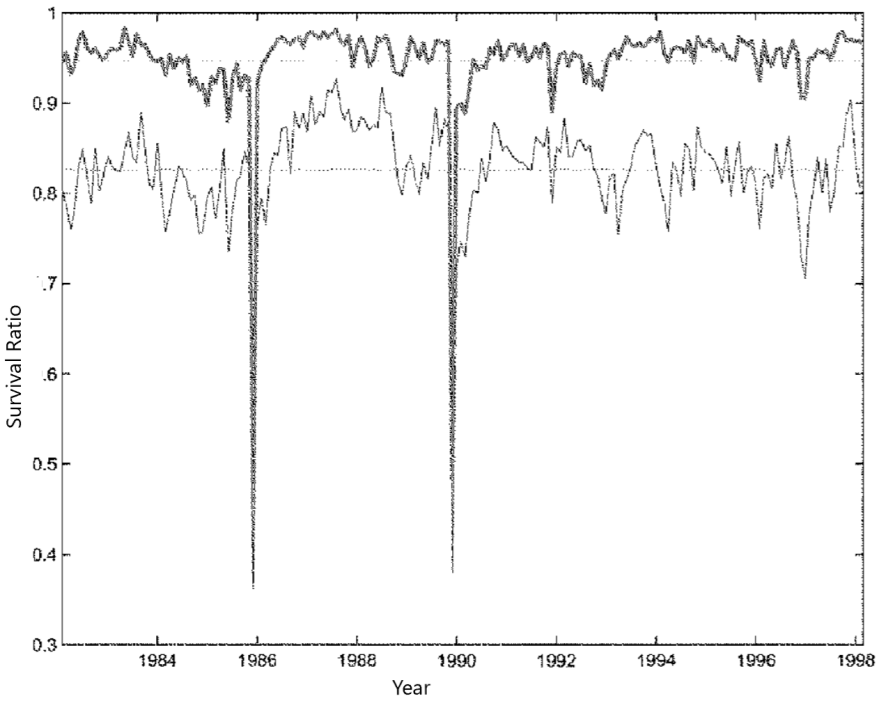


Figure 2.13: Survival ratio of the asset graph versus MST for a stock sample. The thicker curve corresponds to the asset graph and the thinner one corresponds to MST. The window width is 1000 days and the period length is approximately 21 days (figure taken from Onnela et al. [2003b, Fig. 7])

It is worth mentioning that it seems like the asset graph algorithm is the least used among the previously mentioned algorithms—MST, PMFG and the threshold method. Birch et al. [2016] also applied the asset graph algorithm to build a stock-correlation network using $|V| - 1$ edges as with MST. They argued that this

method recognises any misleading edge selections made by MST. As a downside, they mentioned that this method does not give a complete image due to disconnected vertices, and little information is known about these disconnected vertices. In another work, Heimo et al. [2007] used spectral analysis, MST and asset graph to analyse stocks. They argued that asset graph seems to provide more coherent results regarding clusters of stocks compared to spectral analysis.

2.8 Some More Algorithms and Conclusion

We have discussed the most extensively used algorithms for building stock-correlation networks. Other algorithms proposed in this area include, but are not limited to, the directed bubble hierarchical tree (DBHT) [Musmeci et al., 2015; Song et al., 2012, 2011b], p-media problem [Kocheturov et al., 2014], maximum likelihood [Giada and Marsili, 2001, 2002; Guo et al., 2018a], and random matrix theory (RMT) related algorithms [Arai et al., 2015; Chen et al., 2014; Garas and Argyrakis, 2007]. Below, we are going to discuss these algorithms briefly before finishing this chapter. We refer to Marti et al. [2017] and Tsankov [2021] for an overview of papers on methods, algorithms and their applications in financial markets.

DBHT is a method first proposed by Song et al. [2011b] which extracts hierarchies from a PMFG. They showed that PMFG can be viewed as a set of "bubbles" that are themselves maximal planar graphs. The hierarchy comes in form of a tree structure in which adjacent bubbles are joined through 3-cliques. Musmeci et al. [2015] used this algorithm to derive hierarchies from stock markets. They took the industrial sector classification of stocks as a benchmark partition and showed that DBHT can outperform other methods—SLCA, ALCA, CLCA and k-medoids—in retrieving more information with fewer clusters.

Kocheturov et al. [2014] put forward the idea of finding a set S of stocks or *medians*—which are the centres of stars—of a predefined size p that maximise the total "similarity" over all stocks and this set. Thus, their network is a set of connected components that are stars. They denoted this similarity between stock i and set S by the maximum correlation between the *log*-return (explained in the next chapter) of i and all stocks in the set: $\rho(i, S) = \max_{j \in S} \rho_{ij}$. Then they solve the following problem

$$\max_{S} \sum_{i=1}^{n} \rho(i, S) = \max_{S \subset X, |S|=p} \left(\sum_{i=1}^{n} \max_{j \in S} \rho_{ij} \right) \tag{2.13}$$

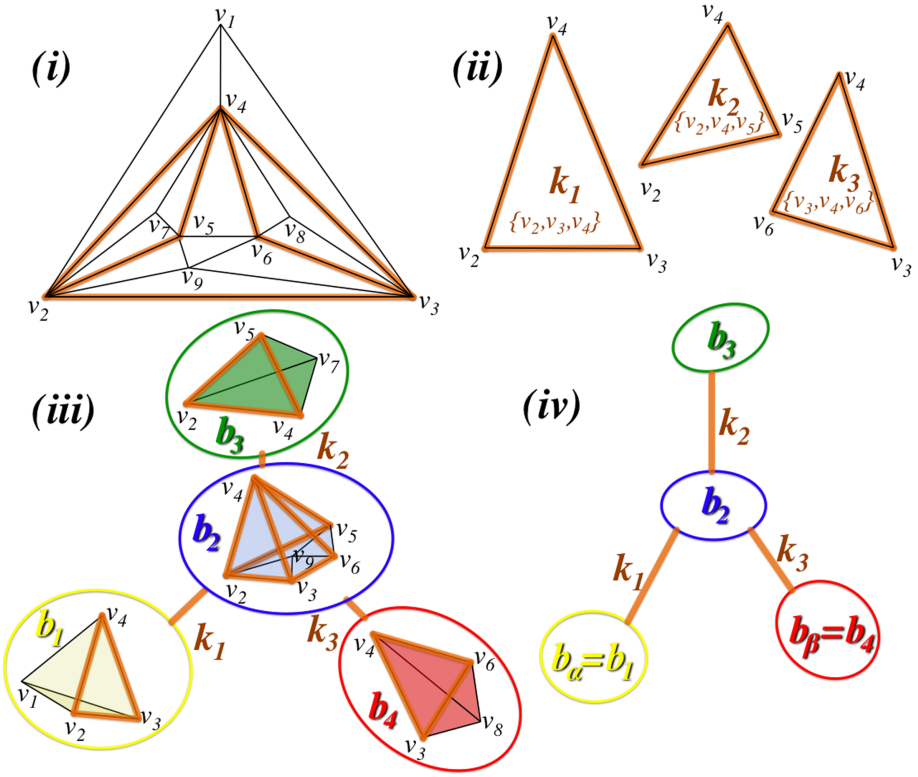


Figure 2.14: Extraction of hierarchies from a PMFG using DBHT [Song et al., 2012]

in which X, n, and p respectively denote the set of all stocks in the market, number of stocks, and the number of clusters (or medians). They argued that the medians are more stable during the market crisis compared to non-crisis periods, and that an increasing stability of the stars could be an indicator of an upcoming crisis.

Lastly, we can see the use of methods using aspects of RMT and maximum likelihood algorithms for stock market analysis. Methods that utilise RMT (sometimes combined with other methods) have been used to detect clusters of stocks [Arai et al., 2015] and investigate the stock market dynamics [Chen et al., 2014]. Take for example, Namaki et al. [2011] who used the threshold method for asset allocation by applying aspects of RMT. Regarding the maximum likelihood algorithm, as with previous algorithms, it has been utilised to detect clusters of stocks Giada and Marsili [2002, 2001], track the behaviour of stock market and study the degree distribution of stocks to see how well it matches power law [Guo et al., 2018a].

We can see that a good clustering has been emphasised in many stock-correlation network studies, and it seems to be the most important aspect that researchers investigate. In light of this, we decided to come up with new algo-

rithms of building stock-correlation networks with the hope of obtaining a better clustering of stocks compared to the previous studies.

CHAPTER 3

Proportional Degree Stock-correlation Network

3.1 Introduction

In this chapter, we propose an algorithm called proportional degree (PD) to build a stock-correlation network based on the normalised mutual information (NMI) similarity matrix. In contrast to the other widely used methods—MST and PMFG—PD uses the relatively (with regards to the total sum of edge weights which denote similarity in the complete weighted graph), rather than the absolutely most important edges (importance as specified in the first paragraph of Section 2.7.4). We show that the PD network with the same size as its PMFG counterpart has a better homogeneity of cliques (homogeneity as mentioned in Section 2.7.2) according to the stock economic sectors. We show that this result still holds even if we use correlation coefficient (defined in Section 2.4.1) rather than NMI—defined in equation (2.6). We also show that the PD network has an overall better clustering compared to the PMFG network in terms of agreement with the normalised spectral clustering (NSC as outlined in Algorithm 4) of the similarity matrix.

In Section 3.2, we show how to extract the pairwise mutual information and NMI between stocks and explain why we use NMI to account for the correlation between stocks. Then we describe the PD algorithm in conjunction with the methods

that we use in order to compare PD and PMFG. In Section 3.3, we present the results of the comparison of the two networks built by the PD and PMFG algorithms. Finally, Section 3.4 provides a recap of this chapter.

3.2 Method

In order to determine the similarity of stocks, we study their similarity in their price action. For n stocks traded in m business days, let P_{it} be the closing price of stock i on day t. The return of stock i on day t is defined by $\frac{P_{it}-P_{i(t-1)}}{P_{i(t-1)}}$. We know that $x \approx \ln(1+x)$ holds if x is small. Since we can assume that the daily return on a stock is small, we can approximate it as follows—which is called *log*-return. The *log*-return of stock i on day t for $t = 2, 3, \ldots, m$ and $i = 1, 2, \ldots, n$ is defined by

$$R_{it} = \ln \frac{P_{it}}{P_{i(t-1)}}. \tag{3.1}$$

(Calculating the *log*-return rather than return is a common practice in finance.) The correlation coefficient of *log*-returns of stocks has been the most commonly used measure to quantify stocks' similarity. However, a few other measures have also been used.

For the rest of the thesis, wherever we mention similarity between stocks, we mean the similarity in their *log*-returns. The correlation coefficient of *log*-returns of two stocks i and j for a sample of m days can be approximated as

$$\rho_{ij} = \frac{\sum_{t=2}^{m}(R_{it} - \overline{R_i})(R_{jt} - \overline{R_j})}{\sqrt{\sum_{t=2}^{m}(R_{it} - \overline{R_i})^2}\sqrt{\sum_{t=2}^{m}(R_{jt} - \overline{R_j})^2}} \tag{3.2}$$

where $\overline{R_i}$ is the average of the *log*-return of stock i over the studied time period. Another measure that has been used in the literature to account for similarity is a variant of the correlation coefficient called the *partial correlation coefficient* (see Kenett et al. [2015] for applications of this measure in financial markets and [Kenett et al., 2010, 2015; Wang et al., 2018; Zhang et al., 2010] for stock-correlation networks generated based on this measure). A partial (or residual) correlation measures the extent to which a given variable Z affects the correlation between another pair of variables X and Y. Thus, the partial correlation value indicates the correlation remaining between X and Y after the correlation between X and Z and between Y and Z have been subtracted. Therefore, the difference between the correlations

and the partial correlations provides a measure of the influence of variable Z on the correlation of X and Y. The partial correlation coefficient between variables X and Y subject to the influence of Z is defined as

$$\rho_{X,Y:Z} = \frac{\rho_{X,Y} - \rho_{X,Z}\rho_{Y,Z}}{\sqrt{\left(1-\rho_{XZ}^2\right)\left(1-\rho_{YZ}^2\right)}}. \tag{3.3}$$

Kenett et al. [2010] used this measure to build two stock-correlation networks: partial correlation threshold network (PCTN) and partial correlation planar maximally filtered graph (PCPG) network. They showed that by using partial correlation, their approach was able to detect the prominent role of financial stocks in controlling the correlation structure of the market. This role was not evident using only the correlation coefficient. Kenett et al. [2015] used this measure to determine the stability of markets and show the influence of stocks on other stocks belonging to different sectors. Lastly, Wang et al. [2018] formed two MST stock-correlation networks based on correlation coefficient and the partial correlation coefficient: MST-Pearson and MST-partial. They showed that correlation between stocks is greatly affected by other markets, and that the outcomes from the MST-partial network are more reasonable and useful than those from MST-Pearson.

The downside of using correlation coefficient is that it only accounts for the linear correlation between variables. In other words, we have:

1. Correlation coefficient $\rho_{XY} = 1$ implies that a linear equation describes the relationship between X and Y perfectly, with all data points lying on a line for which Y increases as X increases.
2. Correlation coefficient $\rho_{XY} = -1$ implies that all data points lie on a line for which Y decreases as X increases.
3. Correlation coefficient $\rho_{XY} = 0$ implies that there is no linear relationship between the variables.

(See Holmes et al. [2017, Section 13.1] for more detail.) For example, if $X \sim \mathcal{N}(0, \sigma^2)$, and $Y = X^2$, $\rho_{XY} = 0$ even though they are obviously not independent.

However, stock returns demonstrate a nonlinear behavior [Hsieh, 1991; Oh and Kim, 2002; McMillan, 2001]. Oh and Kim [2002] used a piecewise nonlinear model to predict the stock price index. McMillan [2001] found a nonlinear relationship between stock returns and interest rates, and showed that their model is more accurate than some previously proposed linear model. Fiedor [2014] argued that it is perplexing to consider only the linear dependency between stock returns to

come up with a corresponding hierarchical clustering of them.

In light of the above, we use the normalised mutual information (NMI) as defined by equation (2.6) to measure the correlation between stocks. We prefer mutual information over correlation coefficient because as argued by Fiedor [2014] and da Silva et al. [1989], the former can detect the relationship between variables that cannot be detected by a linear correlation measure such as the latter. Guo et al. [2018b] argues that this feature of mutual information—detecting nonlinear relationships—is more evident when the stock market exhibits violent fluctuations. Still, we also use correlation coefficient to compare its resulting network with that of NMI in Section 3.3 to make a comparison of PD and PMFG irrespective of the similarity measure used.

Mutual information and its variants have been used in the literature before [Fiedor, 2014, 2015; You et al., 2015; Han et al., 2019; Barbi and Prataviera, 2019; Goh et al., 2018]. You et al. [2015] used the partial mutual information (PMI)—we refer to equation (6) in their paper for definition—to build the Shanghai Stock Exchange network. They argued that PMI is better than correlation coefficient in reconstructing the sector structure of the market more precisely, while retaining the (almost) scale-free property of the networks. It is worth mentioning that they investigated both MST and PMFG and found that the degree distribution is closer to power law in the latter compared to the former. Han et al. [2019] used mutual information along with MST to study the market dynamics of the Chinese stock market. Fiedor [2014] used the mutual information rate (MIR)—see Blanc et al. [2011] for a definition and Gray and Kieffer [1980] for a more rigorous definition of this measure. They used MIR with MST and PMFG to build a stock-correlation network, and they showed that the resulting network based on MIR is significantly different from one based on correlation coefficient. In the following, we demonstrate how to calculate NMI between stocks based on their *log*-returns.

One question we must face is how to construct the probability and joint probability distributions of the stocks in order to find the mutual information between them. We use the same numerical method as proposed by Guo et al. [2018b]. In order to find the probability distribution of the *log*-return of stock i—as given by equation (3.1)—we sort R_{it} values for $t = 2, 3, \ldots, m$ in ascending order and divide the sorted values into q bins. Then we count the number of *log*-returns of stock i for $i = 1, 2, \ldots, n$ in each bin a for $a = 1, 2, \ldots, q$ denoted by f_{ia} and get the approximate probability by $p_{ia} \approx \frac{f_{ia}}{m}$. Similarly, we find the joint probability distribution of the *log*-returns of stocks i and j for $i, j = 1, 2, \ldots, n$ by dividing their sorted

log-returns into $q \times q$ bins. In such case, We denote the number of *log*-returns of i and those of j in bin (a, b) by f_{ijab}, and the approximate joint probability is given by $p_{ijab} \approx \frac{f_{ijab}}{m}$. As a result, we can approximate the entropy of stock i and joint entropy of stocks i and j—see equations (2.3) and (2.4)—by

$$H(S_i) = -\sum_{a=1}^{q} p_{ia} \log_2 p_{ia} \tag{3.4}$$

$$H(S_i, S_j) = -\sum_{a=1}^{q} \sum_{b=1}^{q} p_{ijab} \log_2 p_{ijab}. \tag{3.5}$$

Therefore, the mutual information of stocks i and j can be given by substituting equations (3.4) and (3.5) in equation (2.5), and the NMI is given by

$$NMI(S_i, S_j) = \frac{2I(S_i, S_j)}{H(S_i) + H(S_j)}, \quad i \neq j \tag{3.6}$$

as defined in equation (2.6) which produces a symmetric $n \times n$ matrix with diagonal elements of zero. We consider this matrix to be the similarity matrix of the stocks unless specified otherwise. Now that we have determined how to specify the pairwise NMI between stocks, we can elaborate on our algorithm (PD) for building the stock-correlation network. Below, we are going to explain PD and mention the aspects based on which we compare PD with PMFG.

3.2.1 Proportional Degree (PD) Algorithm

We first determine the degree of each vertex in our output network in a manner such that it is proportional to its weight, where the weight of a vertex (or stock weight) is the sum of its similarity values across all the other vertices. The weight of stock i is defined by

$$SW_i = \sum_{j \neq i} s_{ij} \tag{3.7}$$

where SW_i and s_{ij} respectively denote the weight of stock i and the similarity between stocks i and j.

Consequently, the calculated degree of a vertex d'_i should be more or less given by

$$d'_i = \frac{SW_i}{\sum_{j=1}^{n} SW_j} \times (2M) \tag{3.8}$$

in which M is the total number of edges, so $2M$ is the total sum of the degrees of all vertices. However, the degree of a vertex, being the number of adjacent vertices, is required to be integer. In order to round the calculated degrees d'_i while preserving their total sum, we apply the cascade rounding algorithm as follows.

Table 3.1: Cascade rounding algorithm for finding degrees while preserving the total sum of the calculated degrees

Vertex	Calculated degree	Degree
1	d'_1	$d_1 = \lfloor d'_1 \rceil$
2	d'_2	$d_2 = \lfloor d'_1 + d'_2 \rceil - d_1$
3	d'_3	$d_3 = \lfloor d'_1 + d'_2 + d'_3 \rceil - (d_1 + d_2)$
$\vdots$	$\vdots$	$\vdots$
i	d'_i	$d_i = \lfloor \sum_{j=1}^{i} d'_j \rceil - \sum_{j=1}^{i-1} d_j$
$i+1$	d'_{i+1}	$d_{i+1} = \lfloor \sum_{j=1}^{i+1} d'_j \rceil - \sum_{j=1}^{i} d_j$
$\vdots$	$\vdots$	$\vdots$
n	d'_n	$d_n = \lfloor \sum_{j=1}^{n} d'_j \rceil - \sum_{j=1}^{n-1} d_j$

To use cascade rounding, we first relabel the vertices as 1 to n, from the largest stock weight to smallest. Then we determine the degree of vertex i recursively by subtracting the cumulative sum of the degrees of the $i-1$ vertices before it, from the rounded cumulative sum of the calculated degrees of vertices 1 to i. Thus, $d_1 = \lfloor d'_1 \rceil$ and

$$d_i = \lfloor \sum_{j=1}^{i} d'_j \rceil - \sum_{j=1}^{i-1} d_j, \quad 2 \leq i \leq n \tag{3.9}$$

where d_i is the degree of vertex i and $\lfloor x \rceil$ denotes the nearest integer to x (see Table 3.1 for a demonstration of this algorithm). For the rest of the chapter, wherever we mention degree in association with the PD algorithm, it means the integer or rounded calculated degree. After determining degrees, the PD algorithm builds a network as follows.

Algorithm 7 PD algorithm

Input:
  V : set of stocks
  s_{ij} : similarity between stock i and j given by the NMI (3.6) or correlation coefficient (2.2)
Output:
  $G(V, E)$: proportional degree network

$E \leftarrow \emptyset$
$G(V, E) \leftarrow$ network of stocks V with no edges
$S \leftarrow$ list of (i, j, s_{ij}) $(i, j \in V\ ,\ i \neq j)$, sorted in descending order of s_{ij}
$deg(i)$: number of vertices adjacent to vertex i in network G
for (i, j, s_{ij}) in S **do**
    if $(deg(i) < d_i)$ and $(deg(j) < d_j)$ and $(e_{ij} \notin E)$ **then**
        $E \leftarrow E \cup \{e_{ij}\}$

For the purpose of comparing with the PMFG network, we set the total number of edges in this algorithm to $M = 3n - 6$ to equal the value in PMFG.

3.2.2 Cliques

As mentioned in Section 2.7.2, one of the advantages of PMFG over MST is the additional information linked with the inclusion of 3 and 4-cliques. Also, as pointed out in the same section, one way of analysing the cliques is to investigate how often the stocks in them belong to the same economic sector; in other words, what is the degree of cliques homogeneity with respect to the economic sectors. In the next sections, we compare the cliques homogeneity of PMFG and PD to evaluate which network is better in finding cliques of stocks belonging to the same economic sector.

3.2.3 Clusters

As mentioned in the first paragraph of Section 2.4, if each vertex of a graph only belongs to one cluster (no overlapping clusters), such a division of the graph determines a partition. Partitions of the PMFG stock-correlation networks have been widely studied [Buccheri et al., 2013; Song et al., 2011a; Wang and Xie, 2015; Wang et al., 2017]. As with the analysis of cliques, one of the ways of analysing the clusters is investigating how well they match the economic sector classification of the stocks since we would hope that stocks belonging to the same economic sector are more likely to be in the same cluster. (Chen et al. [2014] showed in their paper that stocks in the same economic sector are more likely to be in the same cluster.)

We evaluate the clusters found by Louvain community detection (defined in Section 2.4.3) in PD and PMFG networks through their similarity to the stocks' economic sectors partition. We also use Louvain community detection and NSC on the similarity matrix of the stocks (complete weighted graph of NMI between stocks as mentioned in the last paragraph of Section 3.2) and compare the resulting partitions with the partitions of the PD and PMFG networks achieved through the same methods. Lastly, to compare any two partitions, we use the adjusted rand index (ARI) as described in Section 2.4.5.

3.3 Results

In this section, we present the results we got on cliques and clusters evaluation of the PD and PMFG networks. First, we briefly outline the data set based on which we have built the networks. Then we do the cliques and clusters analysis, and lastly, we assess the robustness of both networks.

3.3.1 Data Set

We have selected 125 out of 200 stocks in S&P/ASX 200. This index is based on the 200 largest stocks in the Australian Securities Exchange (ASX) which account for around 82% (as at March 2017) of Australia's sharemarket capitalisation (see `http://www.asx200list.com`). The criterion used for selection was that these 125 stocks are the ones that were traded throughout the whole period of the years 2013-2016. Our data set comprises the following: 21 stocks in Consumer Discretionary (CD) sector, 6 in Consumer Staples (CS), 8 in Energy (E), 19 in Financials (F), 10 in Health Care (HC), 16 in Industrials (I), 2 in Information Technology (IT), 26 in Materials (M), 12 in Real Estate (RE), 2 in Telecommunication Services (TS), and 3 in Utilities (U).

We should point out that as you will see, in the analysis that we have done in the rest of the chapter, in parts, we have taken different random samples of the stocks to input into the algorithms. The reason for this is testing the algorithms on more than one input and assessing the consistency of their results.

3.3.2 Stock-correlation Networks

In order to get the NMI between all the stocks to generate the similarity matrix, we chose the bin size of $q = 20$ (referring to equations (3.4) and (3.5)) for the 1013 trad-

ing days in our data since as Guo et al. [2018b] mention, for a large enough q, there is not much difference in the values of mutual information, and they considered a bin size of $q = 10$ for the 734 trading days in their data.

We generated the PD and PMFG networks for the above-mentioned data, the analysis of which is provided below. In the PD network, we have vertices with degrees ranging from 1 to 9 whereas in the PMFG network the degrees range from 3 to 29. Visualisations of both networks are in the forthcoming Figure 3.3.

3.3.3 Cliques

The analysis of networks generated in the above (Section 3.3.2) indicates that there are 87 maximal cliques of size 3 and larger, including 52 maximal cliques of size 3, 23 of size 4, 9 of size 5, and 3 of size 6 in PD. Similarly, there are 122 maximal cliques of size 4 in the PMFG network. To quantify homogeneity, $^{47}/_{87} = 0.54$ of the maximal cliques in the PD network consist of stocks all belonging to the same economic sector whereas this ratio is $^{43}/_{122} = 0.35$ for the PMFG network. We also compared the homogeneity of the maximal cliques with minimum size of 3 in the two networks on different random subsets of the stocks. To this end, we considered different proportions $r = {}^{4}/_{5}, {}^{3}/_{4}, {}^{2}/_{3}, {}^{1}/_{2}$ of all the 125 stocks, and for each r, we took 10 random samples of size $\lfloor rn \rceil$ ($\lfloor x \rceil$ denotes rounding x to the nearest integer) from the stocks. We plotted the results of all samples for each r and each network as shown in Figure 3.1, and we can see that for every r, the PD network has an overall larger homogeneity of maximal cliques compared with PMFG.

In the PD network, there are 12 maximal cliques of size more than 4 as shown in Table 3.2. In this table, the columns show respectively the stocks in the clique, their corresponding sectors, and the clique size.

Contrasting with this, in a PMFG network—as in any other planar graph—we cannot have a maximal clique with size larger than 4 since as mentioned in Section 2.2.6, such a clique cannot be embedded onto a surface with genus $g = 0$ without any two edges crossing. Also, as discussed in the same section, we can have at most $n - 3$ maximal 4-cliques and $3n - 8$ 3-cliques in any planar graph. Accordingly, in another analysis, we compared the homogeneity of the 3-cliques and 4-cliques in the PD network, which are not necessarily maximal, with their counterparts in the PMFG network. There are 236 3-cliques and 101 4-cliques in the PD network, and we observe a homogeneity of $^{178}/_{236} = 0.75$ in 3-cliques and $^{91}/_{101} = 0.90$ in its 4-cliques. The corresponding ratios are $^{152}/_{367} = 0.41$ and $^{43}/_{122} = 0.35$ in the PMFG network. So we can also see that all the maximal cliques in the PMFG network are

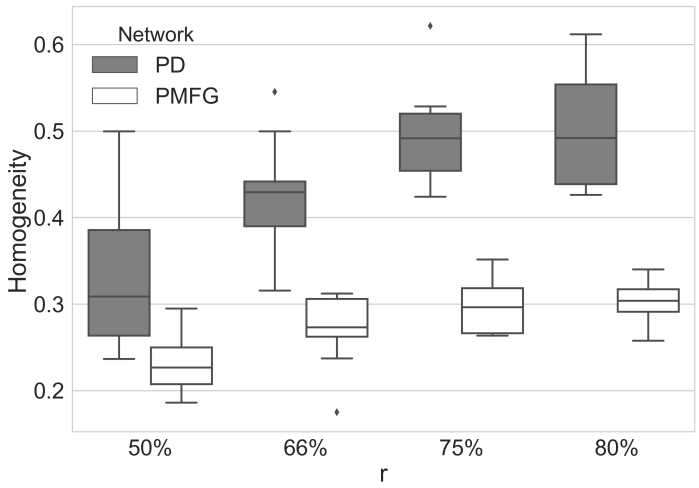


Figure 3.1: Maximal cliques homogeneity comparison of the two networks on different random subsets of proportions r of all stocks

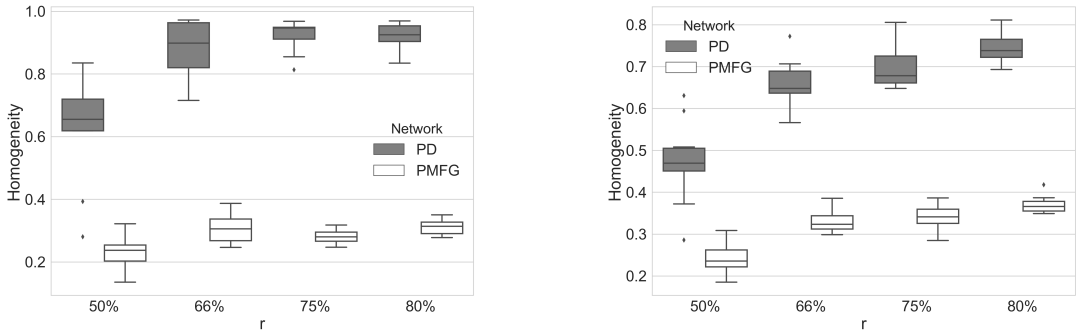
Table 3.2: Maximal cliques with size more than 4 in the PD network

Clique	**Sector**	**Size**
OSH, WPL, WOR, BPT, STO, ORG	E, E, E, E, E, E	6
MGR, CHC, GPT, GMG, SGP	RE, RE, RE, RE, RE	5
MGR, CHC, GPT, GMG, IOF	RE, RE, RE, RE, RE	5
MGR, CHC, GPT, CQR, SGP	RE, RE, RE, RE, RE	5
ILU, RIO, SFR, OZL, WSA	M, M, M, M, M	5
ILU, RIO, SFR, OZL, FMG	M, M, M, M, M	5
ORG, WPL, WOR, BHP, STO	E, E, E, M, E	5
ANZ, NAB, CBA, WBC, MQG	F, F, F, F, F	5
ANZ, NAB, CBA, WBC, BEN, BOQ	F, F, F, F, F, F	6
ANZ, NAB, CBA, WBC, BEN, SUN	F, F, F, F, F, F	6
ANZ, NAB, CBA, WBC, AMP	F, F, F, F, F	5
ANZ, NAB, ASX, BEN, SUN	F, F, F, F, F	5

4-cliques here. As with the previous analysis, we compared the homogeneity of 3-cliques and 4-cliques of the two networks on different random subsets of the stocks, and the result is plotted on Figure 3.2. We can see based on this figure and the cliques analysis so far that the comparison of 3-cliques and 4-cliques homogeneity between the two networks is even more striking than for maximal cliques.

We also evaluated the cliques homogeneity of PD and PMFG built based on

correlation coefficient. Building the correlation-coefficient-based PD network on our stocks, we find the network to have 83 maximal cliques of size 3 and larger with a corresponding homogeneity of $^{48}/_{83} = 0.58$. Also, in this network, there are 120 4-cliques and 243 3-cliques whose corresponding homogeneities are $^{107}/_{120} = 0.89$ and $^{184}/_{243} = 0.76$ respectively. Regarding the correlation-coefficient-based PMFG network built upon our stocks, it has 122 maximal cliques—all of which are of size 4—whose corresponding homogeneity is $^{44}/_{122} = 0.36$. This result agrees with the cliques homogeneity obtained by Tumminello et al. [2005]. The PMFG correlation-coefficient based network has also 367 3-cliques with a corresponding homogeneity of $^{153}/_{367} = 0.42$. It can thus be seen that both the correlation-coefficient-based PD and correlation-coefficient-based PMFG have almost the same homogeneity as their NMI-based counterparts. Moreover, interestingly, these two networks share almost 60% of their edges with their NMI-based counterparts. Thus, it appears that PD gives better information in terms of the cliques homogeneity than PMFG regardless of the similarity measure used. We should note that for the remainder of the chapter wherever we mention the PD or PMFG network, the network is built based on NMI.



(a) 4-cliques

(b) 3-cliques

Figure 3.2: 4-cliques and 3-cliques homogeneity comparison of the two networks on different random subsets of proportions r of all stocks

3.3.4 Clusters

Using the Louvain community detection approach, following Wang and Xie [2015] and Wang et al. [2017], we identified clusters as shown in Tables 3.3 and 3.4. We found seven clusters in PD and six in PMFG. In both tables, "Stocks" refers to the stocks in each cluster. "Sectors" refers to the number of stocks belonging to each economic sector in the corresponding cluster. For example, 17F denotes 17 stocks

belonging to the Financials sector. "Size" refers to the cluster size, "Dominant" refers to the economic sector repeated the most in the cluster, and "Percentage" refers to the proportion of the stocks belonging to the dominant sector in the cluster.

Table 3.3: Clusters captured in the PD network by Louvain community detection

Cluster	Stocks	Sectors	Size	Dominant	Percentage
1	CBA, WBC, ANZ, NAB, AMP, SUN, MQG, IAG, ASX, LLC, BEN, BOQ, PTM, CGF, IFL, HGG, PPT, MFG	17F, 1RE	18	F	94%
2	BHP, WPL, RIO, ORG, FMG, STO, OSH, ORI, WOR, ILU, ALQ, AWC, SVW, BSL, WHC, DOW, MND, SGM, MIN, BPT, OZL, SFR, IGO, WSA	13M, 7E, 4I	24	M	54%
3	TLS, TCL, GMG, SGP, GPT, SYD, MGR, APA, IOF, CQR, BWP, CHC, ABP, SCP, MQA	10RE, 3I, 1U, 1TS	15	RE	67%
4	WOW, WES, CSL, BXB, RHC, SHL, COH, PRY, IVC	5HC, 2CS, 1CD, 1I	9	HC	56%
5	NWS, NCM, RMD, DXS, QAN, DUE, TPM, SKI, ANN, NVT, RRL, MSB, TME, FXJ, MMS, EVN, SIP, GWA, SAI, SRX, GUD, NST	5CD, 5HC, 4I, 4M, 2U, 1RE, 1TS	22	CD	23%
6	QBE, AMC, CCL, AZJ, CTX, CPU, AIO, IPL, FBU, JHX, BLD, TWE, MTS, GNC, ABC, SWM, MYR, SXL, CSR, NUF, PBG, AAD	8M, 5CD, 4CS, 2I, 1E, 1F, 1IT	22	M	36%
7	CWN, TTS, SEK, FLT, HVN, SUL, TAH, ALL, DLX, QUB, JBH, PMV, FXL, IRE, BRG	10CD, 2I, 1F, 1IT, 1M	15	CD	67%

Nonetheless, as pointed out in Section 2.4.3, Louvain community detection yields different partitions depending on the order of vertex evaluation. To mitigate the effect of different partitions corresponding to different orders of vertices on ARI, we applied the Louvain method on 100 random orders of vertices in both networks and took the average of those 100 ARIs for each network in terms of resemblance to the economic sectors' partition of stocks. This produced average ARIs of 0.31 and 0.26 for PD and PMFG networks respectively.

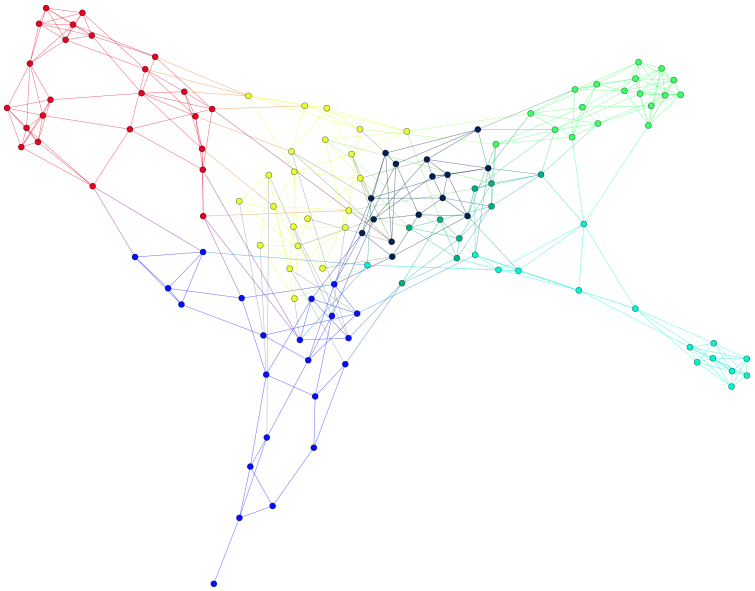
However, the economic sector classification is not the be-all and end-all partition of stocks. For example, every stock labelled as Real Estate in the ASX/S&P 200

Table 3.4: Clusters captured in the PMFG network by Louvain community detection

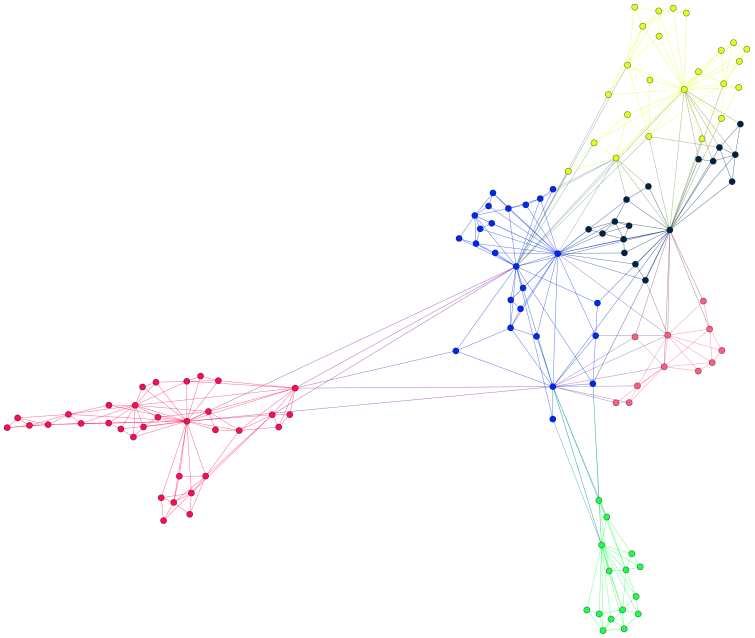
Cluster	Cluster	Sectors	Size	Dominant	Percentage
1	CBA, ANZ, NAB, TLS, WOW, WES, QBE, SUN, IAG, CCL, AZJ, TTS, BEN, MTS, BOQ, GNC, TPM, TAH, CGF, MSB, DLX, QUB, FXL, GWA, AAD	10F, 5CS, 3CD, 3I, 2TS, 1HC, 1M	25	F	40%
2	BHP, WPL, RIO, NCM, ORG, FMG, STO, OSH, ORI, WOR, CTX, AIO, IPL, ILU, ALQ, AWC, SVW, BSL, WHC, DOW, MND, SGM, RRL, MIN, BPT, OZL, FXJ, NUF, EVN, SFR, IGO, WSA, NST	19M, 8E, 5I, 1CD	33	M	58%
3	WBC, LLC, FBU, JHX, QAN, BLD, HVN, SUL, ABC, SWM, MYR, JBH, PMV, SXL, CSR, PBG, GUD	9CD, 5M, 1F, 1I, 1RE	17	CD	53%
4	CSL, RMD, RHC, SHL, DXS, COH, TWE, PRY, ANN, SIP, SRX	9HC, 1CS, 1RE	11	HC	82%
5	NWS, AMP, BXB, MQG, AMC, CWN, ASX, CPU, SEK, FLT, PTM, ALL, NVT, IFL, HGG, PPT, TME, IVC, MMS, MFG, IRE, BRG, MQA, SAI	8CD, 8F, 5I, 2IT, 1M	24	CD	33%
6	TCL, GMG, SGP, GPT, SYD, MGR, APA, DUE, SKI, IOF, CQR, BWP, CHC, ABP, SCP	10RE, 3U, 2I	15	RE	67%

data of 01/10/2018 had been put in the Financials category in the ASX/S&P 200 data of 21/03/2016, which means that the economic sector classification is subject to change, reducing the likelihood that it represents the unique correct partition. Indeed, it could also be argued that there are some significant sub-categories in other economic sectors, which would create more clusters than the number of economic sectors. To create another partition benchmark other than the economic sector classification, we used Louvain community detection on the complete weighted graph of NMI between stocks (similarity matrix of the stocks). This computation produced only four clusters of stocks. Comparing clusters achieved by Louvain community detection in PD and PMFG networks as shown in Tables 3.3 and 3.4 with the new partition benchmark, we got ARIs of 0.40 and 0.36 respectively. Yet not much can be concluded from this comparison since the number of clusters in the benchmark partition is so different from the numbers of clusters in the two

networks.



(a) Different colors referring to different clusters of Table 3.3



(b) Different colors referring to different clusters of Table 3.4

Figure 3.3: Clusters found in the PD (a) and PMFG (b) networks using Louvain community detection. These networks have been visualised using the Python library *NetworkX* [Hagberg et al., 2008] and modified via the software *Gephi* [Bastian et al., 2009].

As mentioned above, since the economic sectoral classification can be misleading, and we do not get the same number of clusters in the two networks—PD and PMFG—using Louvain community detection, we did as follows to draw a more

significant comparison between the clustering behaviour of the two networks. We used NSC on the similarity matrix of the stocks and called the resulting partition C_K. (Here, K refers to the complete weighted graph and should not be mistaken by k which denotes the number of clusters in the NSC algorithm.) Then we applied NSC to the PD and PMFG networks where the corresponding partitions are denoted by C_{PD} and C_{PMFG} respectively. For the similarity matrix to input into NSC, we used the binary adjacency matrix of the networks. In Figure 3.4, the Y-axis denotes the ARI of C_K and C_{PD} versus that of C_K and C_{PMFG}, and the X-axis denotes k. Here, we regard a network to have a good ARI performance if its ARI against C_K is large. It can be seen that for small values of k, there is not much difference in the ARI performance of the networks, for $k = 7, 8$, PMFG has a better ARI performance, and for $k > 8$, PD consistently has a better ARI performance than PMFG. As shown in Figure 3.4, we restrict the number of clusters to being at least 4 because this is the least number of clusters in the application of Louvain to any of the networks or graphs under discussion, and is much smaller than the number of economic sectors.

We implement the heuristic described in the last paragraph of Section 2.4.4 and ignore the gaps between the first, second, and third largest eigenvalues since we ignore 1 and 2 as the number of clusters. We find the largest gaps between the sorted eigenvalues for the similarity matrix are $g(\lambda_4, \lambda_5) = 0.74$, $g(\lambda_{10}, \lambda_{11}) = 0.35$, and $g(\lambda_{11}, \lambda_{12}) = 0.16$. We then note that the PD network has a better ARI performance than the PMFG network for $k = 4, 10, 11$. From another perspective, one of the points we made is that there could be some subsectors lurking in the classification of stocks by the economic sector. As we have 11 economic sectors, from this point of view, the number of clusters would be expected to be $k > 11$, and for these values of k, PD consistently displays a better ARI performance than PMFG. It should be said that although spectral clustering does not perform well on sparse networks all the time (see Krzakala et al. [2013], Amini et al. [2013] and Le et al. [2015] for discussions on this matter), NSC gives a sensible result in our networks as the partitions agree fairly well with the economic sector classification C_e of the stocks as shown in Table 3.5. Also, the result of this table is another indicator that small values of k are not valid, for the ARI of C_{PD} and C_e is smaller than that of C_K and C_e. Besides this, the ARI of C_{PD} and C_e and that of C_{PMFG} and C_e is small for small values of k compared to larger values of k.

As with our analysis of the homogeneity of cliques, to test the validity of our result, we also compared the ARI performance of the two networks on different random subsets of the stocks. To this end, again, we considered different possible

Table 3.5: ARI of $C_{PD}/C_{PMFG}/C_K$ and C_e

k	PD	PMFG	Complete weighted graph
5	0.195	0.0585	0.121
6	0.197	0.0921	0.124
7	0.195	0.069	0.229
8	0.236	0.1665	0.27
9	0.339	0.1557	0.352
10	0.242	0.1015	0.376
11	0.274	0.0833	0.29
12	0.279	0.0799	0.33

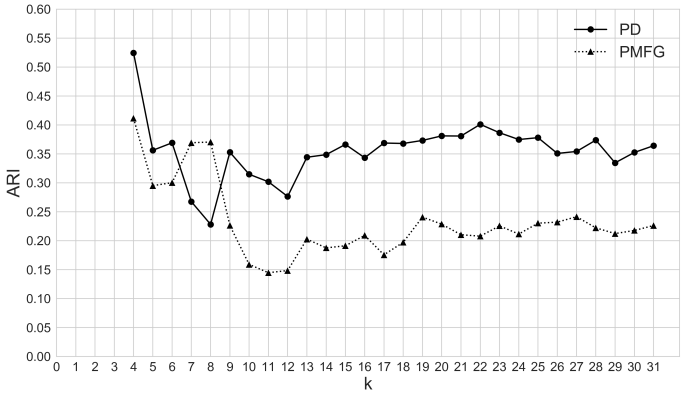
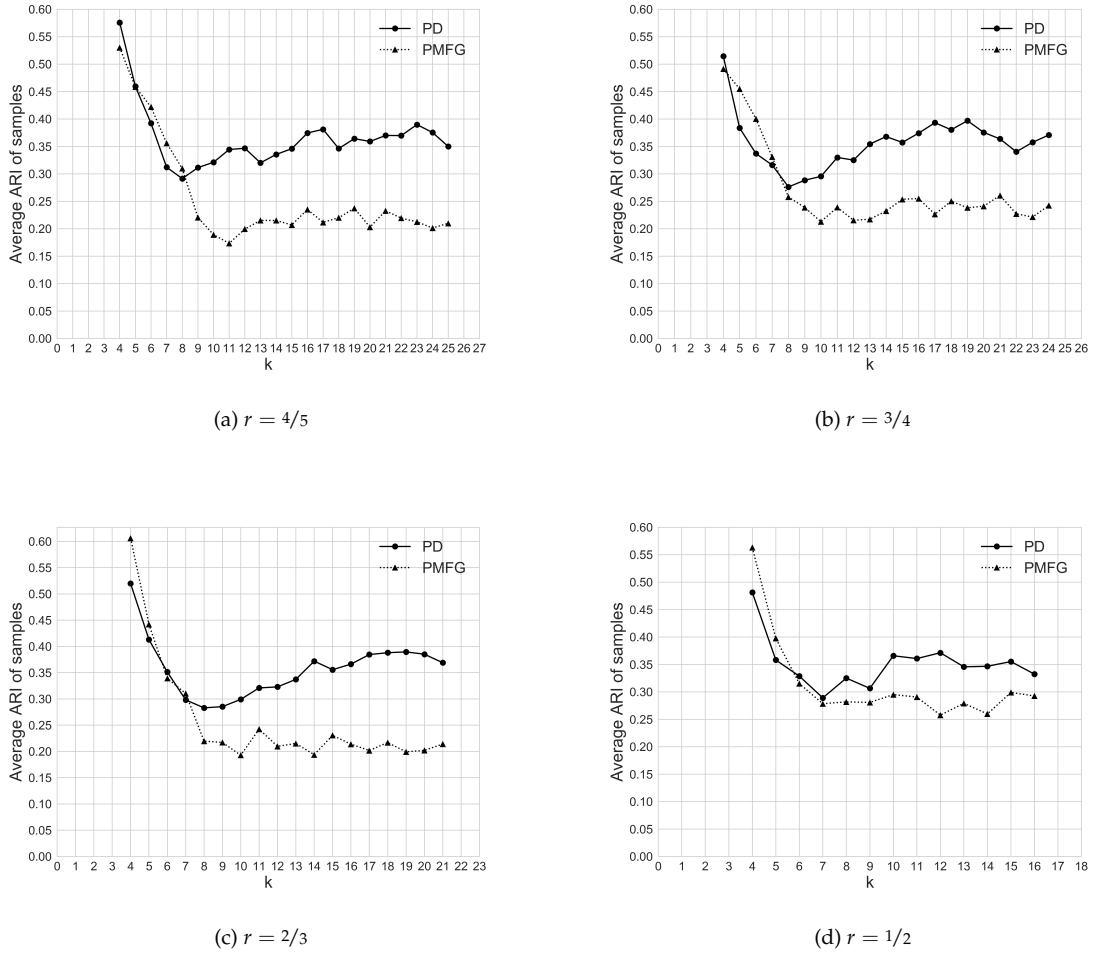


Figure 3.4: ARI performance comparison of C_{PD} versus C_{PMFG}

proportions $r = 4/5, 3/4, 2/3, 1/2$, and for each r, we took 10 random samples of size $\lfloor rn \rceil$ from the stocks. Then we implemented the PD and PMFG algorithms on the samples to generate the two networks and applied NSC on both networks for each sample. Denoting the partitions of PD, PMFG, and the complete similarity matrix of sample i by C_{PD_i}, C_{PMFG_i}, and C_{K_i} respectively, we considered the average ARI of C_{K_i} and C_{PD_i} versus that of C_{K_i} and C_{PMFG_i} for $i = 1, \ldots, 10$ and plotted the results as shown in Figure 3.5. We can see the same pattern for every r; that for a large enough k, PD has consistently a better average ARI performance than PMFG whereas for small values of k, there is virtually no difference in the average ARI performance of the networks. In addition, we can see in Figure 3.5 that as the size of the network shrinks (for smaller values of r), the difference between the average ARI performance of the networks becomes smaller. In other words, forcing into a planar network has less effect on clustering of the stocks in smaller networks. One reason could be that there is less use of larger cliques in smaller networks; thus,

the restriction of PMFG to maximal clique size of 4 becomes less important.



(a) $r = 4/5$

(b) $r = 3/4$

(c) $r = 2/3$

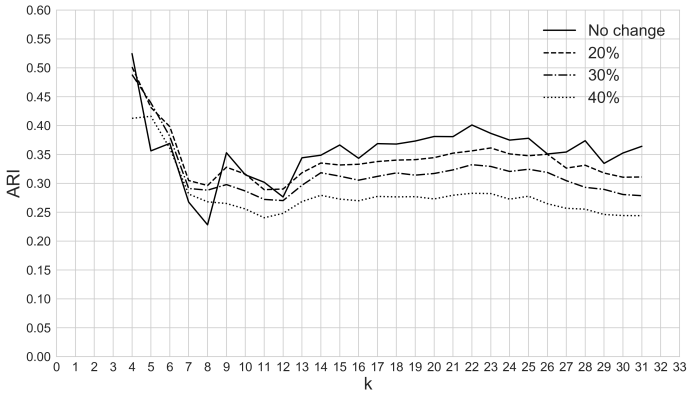
(d) $r = 1/2$

Figure 3.5: Average ARI performance of the PD and PMFG networks for different proportions r of stocks

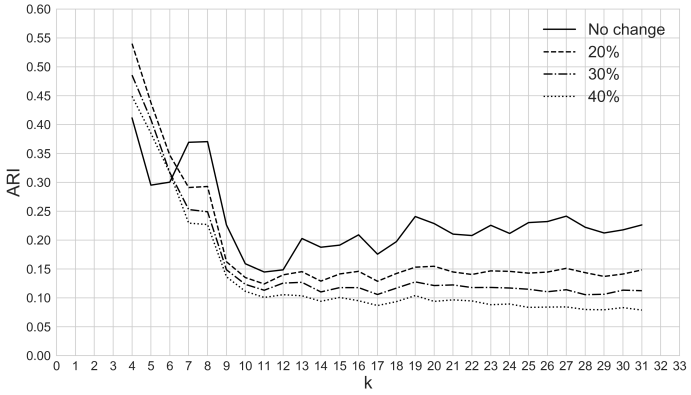
3.3.5 Robustness

As done by Huang et al. [2009], one method to investigate the robustness or stability of network is removing a subset of its vertices or edges at a certain rate. On both networks, we removed 100 different random samples of 20%, 30%, and 40% of the edges and applied NSC on them. Then we plotted the average ARI of C_K and C_{PD} and that of C_K and C_{PMFG} as shown in Figure 3.6. As expected, there is an overall decrease in the ARI performance of both networks as the percentages of edge removal increases. That being said, there is an increase in the average ARI for small k's ($k \leq 8$ and $k \leq 6$ for the PD and PMFG networks respectively), which could be another indicator that small values of k are not valid. Hence, PD has a

better clustering behaviour than PMFG since for larger values of k, it displays a better ARI performance.



(a) PD



(b) PMFG

Figure 3.6: Fluctuations in ARI for NSC of the networks for different proportions of edge removal

In order to see which network has more change of clusters by edge removal, we took the variance of ARIs of both networks for each k in four states, being firstly the networks with no change, and then the networks with 20%,30%, and 40% random edge removal respectively. The results are plotted on Figure 3.7, and we can see that for every k, there is either not a significant difference in the variance of the ARIs or PD has a significantly smaller variance than PMFG; thus, more robust with respect to change in clusters.

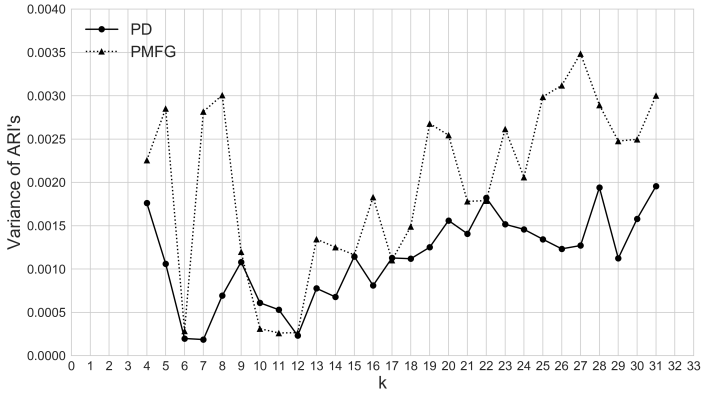


Figure 3.7: Robustness of the networks clusters in presence of edge removal for each k

3.4 Summary

We proposed the PD algorithm to build a stock-correlation network in this work. We used the NMI measure to build a cross-correlation similarity matrix across stocks and applied the PD and PMFG algorithms to generate the corresponding stock-correlation networks. We showed that maximal cliques, 3-cliques, and 4-cliques had a higher homogeneity in the PD network than the PMFG network as to the financial sectoral classification of stocks. We made the same comparison between these two networks—PD and PMFG—built based on correlation coefficient and showed that still PD demonstrates a significantly higher homogeneity of cliques compared to PMFG. We also showed that the NMI-based PD and PMFG demonstrate roughly the same homogeneity of cliques compared to their correlation-coefficient-based counterparts. Moreover, we showed that for a realistic number of clusters in the NSC algorithm, the NMI-based PD network has a better ARI performance than the NMI-based PMFG network in terms of matching the clusters achieved through applying the NSC algorithm on the similarity matrix of stocks.

CHAPTER 4

On Finding the Optimal Tree of a Complete Weighted Graph

4.1 Introduction

In this chapter, we propose an algorithm for building a tree-based stock-correlation network. We begin with a quick reminder of MST and its appealing features and then expand on the motivation behind the algorithm proposed in this chapter.

As mentioned in the previous chapters, of all the stock-correlation generating algorithms, MST is probably the most popular one. Beginning with the seminal work of Mantegna [1999], and as discussed by them and Birch et al. [2016], one of the most attractive features of MST is that it is associated with a hierarchical clustering (explained in Section 2.4.2) of stocks in the following manner. Let D be a distance matrix and T its corresponding MST. The SLCA (Algorithm 2) of D is the same as that of the ultrametric distance matrix $D^<$ (explained in the second paragraph of Section 2.7.1) of T. Precisely, the hierarchical tree (HT) obtained from SLCA on D is exactly the same as that obtained from SLCA on the ultrametric distance matrix of MST.

It has been pointed out in the literature—as in Garas and Argyrakis [2007]; Tumminello et al. [2010]; Coronnello et al. [2005]; Mantegna [1999]—that the aforementioned HT, namely the output of SLCA, performs reasonably well in identi-

fying groups of stocks belonging to the same economic sector. Economic sectoral classification has been quite often used as a benchmark against which hierarchical clustering or any other clustering method of stocks is evaluated qualitatively and quantitatively (see Wang et al. [2017]; Musmeci et al. [2014]; Tabak et al. [2010]; Tumminello et al. [2007b] for examples).

To get the MST of a complete weighted graph K_n in which weights are obtained from the distance matrix D in this context, we apply Kruskal's algorithm (Algorithm 1). Suppose that vertices v_i and v_j are connected in MST. Also suppose that by removing the edge e_{ij} from this MST, the two connected components are C_i and C_j such that $v_i \in C_i$ and $v_j \in C_j$. No information is incorporated from D to join v_i and v_j other than the fact that d_{ij} is the shortest edge from D—or equivalently K_n—that can join connect C_i and C_j. Hypothetically, a tree whose edges and their corresponding weights are determined such that we incorporate all the distances between vertices in K_n (not just the ones that can connect C_i and C_j) would be a better representation of K_n. Such a tree might yield a better clustering of stocks in accordance with the economic sectoral classification of them.

To obtain the output (HT) of SLCA from D, the distance between two clusters is decided solely based on the shortest edge between them connecting their vertices in K_n. In other words, SLCA essentially performs MST as follows. Suppose that the shortest distance between two clusters C_i and C_j corresponds to the edge e_{ij} connecting vertices v_i and v_j where $v_i \in C_i$ and $v_j \in C_j$. Once again, we do not incorporate any information from the distance matrix D to join clusters C_i and C_j other than the length of edges that can join these two clusters. Notice that this issue is closely related to the chaining phenomenon of SLCA which is a shortcoming of this clustering method as explained in paragraph 4 of Section 2.4.2.

Accordingly, this brings up the question of how to incorporate the information from all distances between vertices in K_n to build a tree that represents K_n. We address this question as follows. We look for a tree that estimates the complete weighted graph K_n of distances between vertices in an attempt to minimise the discrepancy between the path length between any two vertices in the tree, and their distance in K_n. We use the residual sum of squares (RSS)—explained in the context of this problem in the next sections—since it is a typical optimality criterion for these types of problems. We call the resulting tree the residual sum of squares optimal tree (RSSOT), and compare its structure in terms of identifying groups of stocks belonging to the same economic sectors with that of MST. We apply two metaheuristics—Simulated Annealing (SA) and Iterated Local Search (ILS) as

described in Sections 2.3.1 and 2.3.2 respectively—to find this tree.

We should point out that the underlying idea for this problem originates from three areas:

1. stock-correlation networks (as described above)
2. phylogenetic trees which represent the relationship in the evolution of different species (see Felsenstein [2004] for an introduction to various methods in this area)
3. t-spanners, an area in graph theory which discusses the spanning trees of a graph with certain properties regarding the distances between vertices in those trees (see Narasimhan and Smid [2007] for an introduction to this area).

We take an approach similar to some investigations in phylogenetic trees, namely least squares methods (see Felsenstein [2004, p. 148–153] for a description of least squares methods in inferring phylogenetic trees), but we have a different treatment of a basic improvement step used in local search heuristics. Also, in contrast to phylogenetic trees, we consider distances between all vertices of the tree, not just leaves.

In Section 4.2, we discuss how to optimise edge weights of a given tree based on the distance matrix or equivalently based on the edge weights of K_n. In Section 4.3, we use the aforementioned metaheuristics to optimise the tree structure—find RSSOT. Section 4.4 discusses our results, and ultimately Section 4.5 is a quick recap of the findings in this chapter.

4.2 Sub-problem: Tree Weight Optimisation

For the complete weighted graph $K_n = (V, \mathbf{E}, d)$, we want to come up with a weighted spanning tree $T = (V, E, w)$ where $E \subset \mathbf{E}$ so that the path length between any two vertices on the tree best estimates the distance between them in K_n. To be precise, we want to minimise the RSS between path lengths in T and their corresponding edge distances in K_n such that

$$RSS\left(T, K_n\right) = \sum_{\substack{m,k \\ m<k}} \left(S\left(P_{m,k}\right) - d_{mk}\right)^2. \tag{4.1}$$

In the equation above, $P_{m,k}$ is the path with the start vertex v_m and the end vertex v_k, and $S(P_{m,k})$ denotes the sum of edge weights on this path. For example, for the path $P_{m,k} = (e_{ma}, e_{ab}, e_{bc}, \ldots, e_{dk})$, $S(P_{m,k}) = w_{ma} + w_{ab} + w_{bc} + \ldots + w_{dk}$. Thus,

equation (4.1) can be reformulated as

$$RSS(T, K_n) = \sum_{\substack{m,k \\ m<k}} \left(\sum_{\substack{i,j \\ e_{ij} \in P_{m,k}}} w_{ij} - d_{mk} \right)^2 . \tag{4.2}$$

In order to find the edge weights for a given spanning tree, we take the derivative of RSS with respect to the w_{ij}'s, so that $\frac{\partial RSS}{\partial w_{ij}} = 0$. It gives us

$$\frac{\partial RSS}{\partial w_{ij}} = 2 \left(\sum_{\substack{m,k:\, e_{ij} \in P_{m,k} \\ m<k}} \left(w_{ij} + \sum_{\substack{e_{rs} \in P_{m,k} \\ e_{rs} \neq e_{ij}}} w_{rs} \right) - \sum_{\substack{m,k:\, e_{ij} \in P_{m,k} \\ m<k}} d_{mk} \right) = 0 \qquad \forall e_{ij} \in E. \tag{4.3}$$

The equation above can be written as

$$\frac{\partial RSS}{\partial w_{ij}} = 2 \left(\alpha_{ij} w_{ij} + \sum_{e_{rs} \neq e_{ij}} \beta_{rsij} w_{rs} - \sum_{\substack{m,k:\, e_{ij} \in P_{m,k} \\ m<k}} d_{mk} \right) = 0 \qquad \forall e_{ij} \in E \tag{4.4}$$

where α_{ij} denotes the number of paths that edge e_{ij} is on, and β_{rsij} denotes the number of paths on which both edges e_{ij} and e_{rs} are. The reason being each term $(.)^2$ in RSS denotes the square error between a path length in T and its corresponding edge distance in K_n. We have $\binom{n}{2}$, equal to the number of paths between each two vertices in T, of these terms. Taking the derivative with respect to w_{ij}, we are considering only the terms $(.)^2$ that include the edge e_{ij} which correspond to the paths that include edge e_{ij}. From equation (4.3), we have the following $n-1$ equations

$$\sum_{\substack{m,k:\, e_{ij} \in P_{m,k} \\ m<k}} \left(w_{ij} + \sum_{\substack{e_{rs} \in P_{m,k} \\ e_{rs} \neq e_{ij}}} w_{rs} \right) = \sum_{\substack{m,k:\, e_{ij} \in P_{m,k} \\ m<k}} d_{mk} \qquad \forall e_{ij} \in E \tag{4.5}$$

or as with equation (4.4) the above can be written as

$$\alpha_{ij} w_{ij} + \sum_{e_{rs} \neq e_{ij}} \beta_{rsij} w_{rs} = \sum_{\substack{m,k:\, e_{ij} \in P_{m,k} \\ m<k}} d_{mk} \qquad \forall e_{ij} \in E. \tag{4.6}$$

The above linear system can be expressed in matrix form as $A\mathbf{w} = \mathbf{d}$, and the entries of matrix A are

$$a_{ij} = \begin{cases} \text{\# of paths including the edge corresponding to the } i\text{-th entry of vector } \mathbf{w} & i = j \\ \text{\# of paths including the edges corresponding to the } i\text{-th and the } j\text{-th entries of vector } \mathbf{w} & i \neq j \end{cases}. \tag{4.7}$$

Let us go through the following example to make it more clear.

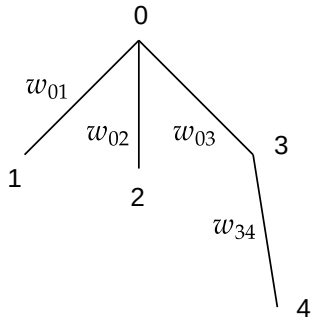


Figure 4.1: An example of what matrix A and vector $\mathbf{d}$ look on this tree

For the tree above, the system of linear equations is as below.

$$\underbrace{\begin{bmatrix} 4 & 1 & 2 & 1 \\ 1 & 4 & 2 & 1 \\ 2 & 2 & 6 & 3 \\ 1 & 1 & 3 & 4 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} w_{01} \\ w_{02} \\ w_{03} \\ w_{34} \end{bmatrix}}_{\mathbf{w}} = \underbrace{\begin{bmatrix} d_{01} + d_{12} + d_{13} + d_{14} \\ d_{02} + d_{12} + d_{23} + d_{24} \\ d_{03} + d_{04} + d_{23} + d_{13} + d_{14} + d_{24} \\ d_{34} + d_{04} + d_{24} + d_{14} \end{bmatrix}}_{\mathbf{d}} \tag{4.8}$$

In the linear system above, the diagonal entries of A—a_{11}, a_{22}, a_{33} and a_{44}—are the number of paths passing respectively through the edges e_{01}, e_{02}, e_{03} and e_{34}. Also, for example, a_{12} is the number of paths passing through both edges e_{01} and e_{02}, and a_{34} is the number of paths passing through both edges e_{03} and e_{34}. In vector $\mathbf{d}$, in the first entry—$d_{01} + d_{12} + d_{13} + d_{14}$—the indices correspond to the beginning vertex and end vertex of the paths that the edge e_{01} is on, and d_{ij} is the distance between the vertices in K_n.

The question is how do we count the number of paths passing through one

specific edge or two specific edges in a tree effectively? To answer the first part of the question, let us say the edge is e_{ij}. We remove this edge from the tree. Then we have two connected components. Let us denote them by S and T. Then the number of edges passing through the edge e_{ij} is $|S||T|$. Also, the indices of d in the vector $\mathbf{d}$ are $\{(s,t) : v_s \in S, v_t \in T\}$. For example, the number of paths passing through the edge e_{03} in Figure 4.2 is $|S||T| = 25$ in which $S = \{v_0, v_1, v_2, v_4, v_5\}$ and $T = \{v_3, v_6, v_7, v_8, v_9\}$. From another perspective, let us set one of the vertices as the root and denote the set of descendants (defined in Section 2.2.7) of the vertex v_i by D_i. We can say the number of paths passing through edge e_{ij} in a tree is equal to $(|D_j|+1)(n-(|D_j|+1))$. It should be noted that v_j is the vertex that is further from the root than v_i, that is, it is a child (defined in Section 2.2.7) of v_i.

To answer the second part of the question—to count the number of edges passing through two edges—say e_{ij} and e_{rs} where $v_j \in D_i$ and $v_s \in D_r$—we do as follows. If $D_j \cap D_s = \emptyset$, the number of paths passing through the two edges is $(|D_j|+1)(|D_s|+1)$. For example, the number of paths passing through the edges e_{02} and e_{37} is 9 since the descendants of vertices v_2 and v_7 are respectively $D_2 = \{v_4, v_5\}$ and $D_7 = \{v_8, v_9\}$. If $D_j \cap D_s \neq \emptyset$, then either $D_j \subset D_s$ or $D_s \subset D_j$. For the former, the number of paths passing through both edges is $(|D_j|+1)\,(n-(|D_s|+1))$, and for the latter this value is $(|D_s|+1)\,\big(n-(|D_j|+1)\big)$. For example, the number of paths passing through both edges e_{03} and e_{37} is $(|D_7|+1)(n-(|D_3|+1)) = (2+1)(10-(4+1)) = 15$.

After finding all entries of A, we can find the edge weights using $\mathbf{w} = A^{-1}\mathbf{d}$. Yet, is the matrix A necessarily invertible? In the following, we prove that not only is A invertible, but positive definite (defined in Section 2.5).

Lemma 4.2.1. *A is a positive definite matrix.*

Proof. We define the function Z on a spanning tree T as follows. For each edge e_{ij}, we assign a variable v_{ij}. Then we define $Z = \sum_{\substack{m,k \\ m<k}} \left(\sum_{\substack{i,j \\ e_{ij} \in P_{m,k}}} v_{ij} \right)^2$. We can see that the terms $(.)^2$ in Z are the same as those in RSS (equation (4.2)). The only difference being the variables w_{ij} are replaced by v_{ij} and the constants d_{ij} are replaced by 0. Z can be written as $Z = \mathbf{v}^\intercal B \mathbf{v} > 0$ where $\mathbf{v}$ is the vector of variables v_{ij} and B is a matrix whose entries are as follows: b_{pq} is the number of terms $(.)^2$ in Z including the variable v_{ij} assigned to the *p-th* entry of vector $\mathbf{v}$ for $p = q$, and for $p \neq q$, b_{pq} is the number of terms $(.)^2$ including both variables v_{ij} and v_{rs} assigned to the *p-th*

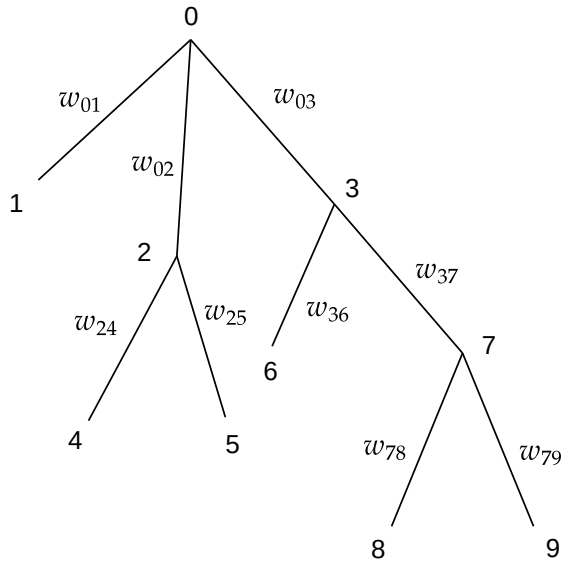


Figure 4.2: An example of finding the entries of matrix A on a tree

and q-*th* entries of vector $\mathbf{v}$. Since each term $(.)^2$ denotes a path in T, we can say that b_{pq} is the number of paths including the edge e_{ij} assigned to the p-*th* entry of vector $\mathbf{v}$ for $p = q$, and for $p \neq q$, b_{pq} is the number of paths including both edges e_{ij} and e_{rs} assigned to the p-*th* and q-*th* entries of vector $\mathbf{v}$. Thus, $B = A$, and since B is positive definite, so is A.

□

Since A is positive definite, we can use Cholesky decomposition—as defined in Section 2.6—of A in the form $A = LL^\intercal$. From there, we can solve $Ly = \mathbf{d}$, and then $L^\intercal \mathbf{w} = y$ to find the weights. In the following, we discuss how we find the optimal tree.

4.3 Problem: Tree Structure Optimisation

So far, we discussed how we can find the edge weights for a given tree based on the edge weights in the complete weighted graph. The question is: how can we find the tree with the minimum RSS? We can build n^{n-2} spanning trees on any n number of labelled vertices. That means for as few as 50 labelled vertices, we can have roughly as many spanning trees as the number of atoms in the known universe.

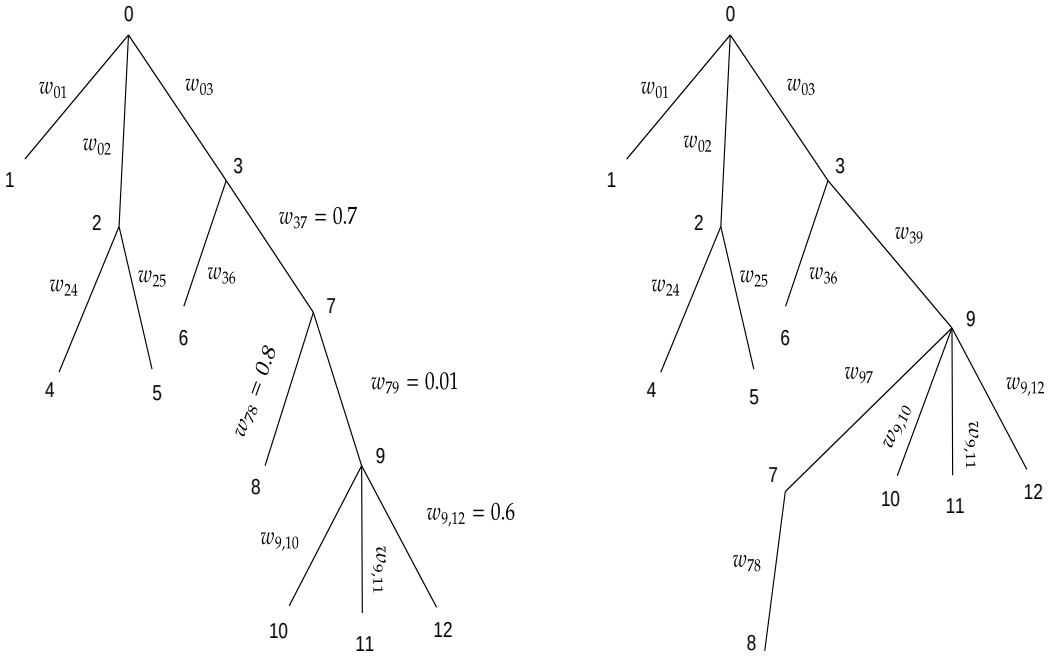
Due to the large scale of the problem, we make use of two metaheuristics—in this case, Simulated Annealing (SA) and Iterated Local Search (ILS)—to approximate the optimal tree.

4.3.1 Tree Structure Change for Optimisation

Before discussing SA and ILS on a tree, let us explain how we make a change in the structure of a given tree in order to accept or reject the transition between two tree structures. Let T_t be the tree at time t and let us denote its corresponding structure by $T(V, E)$. Let us also denote the structure after change by $T'(V, E')$—the structure that we want to accept or reject. For $v_i \in V$, we denote the neighbours of v_i by $N(v_i)$. We pick one edge $e_{ij} \in E$ (we are going to discuss how we pick this edge in the rest). Then we define set C as $C = N(v_i) \cup N(v_j) \setminus \{v_i, v_j\}$. We pick $v_k \in C$ uniformly at random. If $v_k \in N(i)$, then $E' = E \cup \{e_{jk}\} \setminus \{e_{ik}\}$; otherwise, if $v_k \in N(j)$, then $E' = E \cup \{e_{ik}\} \setminus \{e_{jk}\}$. We denote the former structure change by $SC(T, e_{ij}, e_{jk}, e_{ik})$ and the latter by $SC(T, e_{ij}, e_{ik}, e_{jk})$. In $SC(T, ., ., .)$, the second, third, and forth terms are respectively the picked edge, the edge that is added to, and the edge that is removed from the tree.

How do we pick the edge e_{ij} to make the structure change in the tree? We can either pick it uniformly at random, or we can be biased towards the smaller edges. We explain the latter in the context of the following example. In the tree in figure 4.3a, suppose that after finding the edge weights, e_{79} has a weight significantly smaller than the rest of the edges. This means that the distance between the vertices in $N(v_7) \setminus v_9 = \{v_3, v_8\}$ and those in $N(v_9) \setminus v_7 = \{v_{10}, v_{11}, v_{12}\}$ is approximately the sum of the edge weights between them without counting w_{79}. In other words, for $i \in \{v_3, v_8\}$ and $j \in \{v_{10}, v_{11}, v_{12}\}$, $S(P_{i,j}) \approx \sum_{e_{rs} \in P_{ij} \setminus e_{79}} w_{rs}$. Thus, for the next tree structure (the next state), we take one of the vertices—picked uniformly at random—in $N(v_7) \setminus v_9$ or $N(v_9) \setminus v_7$ and connect it respectively to v_9 or v_7. That is to say, in the former, one structure change can be $SC(T, e_{79}, e_{9k}, e_{7k})$ such that $k \in \{3, 8\}$, and in the latter, it can be $SC(T, e_{79}, e_{7k}, e_{9k})$ such that $k \in \{10, 11, 12\}$. For example, the resulting tree of $SC(T, e_{79}, e_{93}, e_{73})$ will look like Figure 4.3b.

If we prefer edges with smaller weights—whose effect we are going to investigate in the rest—to make the structure change, we want the smaller an edge, the more probable to be selected. To this end, for $T(V, E, w)$ over n vertices, we sort the edges according to their weights in ascending order. Let $\mathbf{e}^s$ denote the column vector of the edges E in the order of their weights. We take a number r uniformly



(a) Tree before structure change

(b) Tree after structure change

Figure 4.3: Demonstration of the structure change $SC(T, e_{79}, e_{39}, e_{37})$

at random from the interval $(0, 1)$. Let us define $\lceil x \rceil$ as the smallest integer that is larger than or equal to x. Then the edge that we pick for the structure change is the entry $\lceil r^l n \rceil$ of $\mathbf{e}^s$, which is $e^s_{\lceil r^l n \rceil}$ where $l \in \{z : z \in \mathbb{Z}^+, z \geq 1\}$. The larger we choose the value of l, the more biased we are towards smaller edges, and $l = 1$ picks an edge uniformly at random.

The other thing we investigate before discussing the SA and ILS algorithms on a tree is the change in matrix A and vector $\mathbf{d}$ following the structure change in $T(V, E)$. Should we recompute every entry of A and $\mathbf{d}$ after every structure change? Let us define A' and $\mathbf{d}'$ as the matrix and vector corresponding to $T'(V, E')$.

Lemma 4.3.1. *Suppose we have the structure change $SC(T, e_{ij}, ., .)$ resulting in tree $T'(V, E')$. All the entries of A and A' are the same except the rows and columns corresponding to e_{ij}. So are all the entries in $\mathbf{d}$ and $\mathbf{d}'$ except the entry corresponding to e_{ij}. Thus, we only need to recompute the entry in $\mathbf{d}$ and the rows and columns in A corresponding to e_{ij} to obtain A' and $\mathbf{d}'$.*

Proof. Consider the tree $T(V, E)$ in Figure 4.4 on which we want to make the structure change based on the picked edge e_{ij} and $v_k \in C$ (C as defined above in the first

paragraph of Section 4.3.1). Here, α and β are as defined as in equation (4.4) for $T(V, E)$, and the equivalents of them are α' and β' for $T'(V, E')$. If $v_k \in N(v_j) \setminus \{v_i\}$, the structure change is $SC\left(T, e_{ij}, e_{ik}, e_{jk}\right)$.

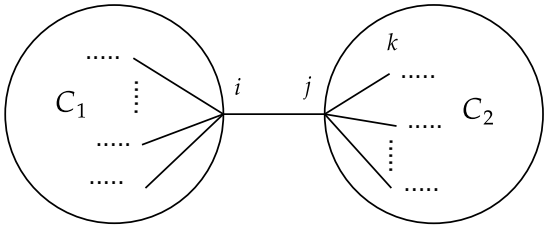
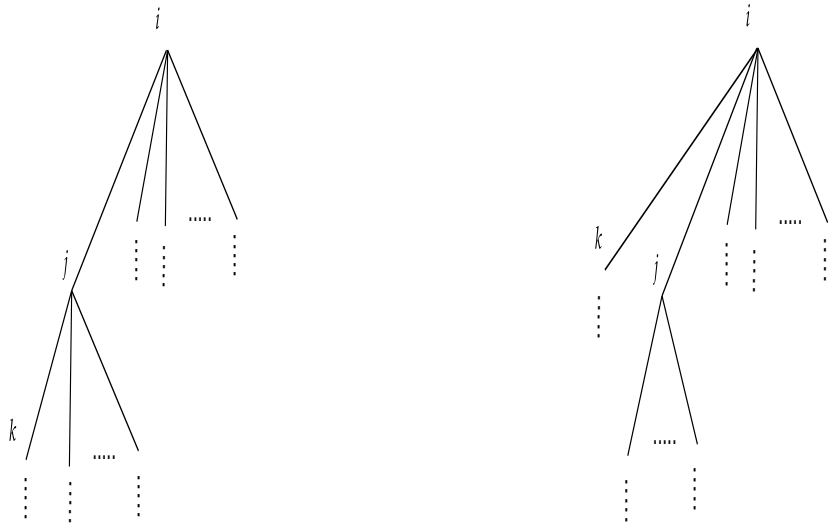


Figure 4.4: Tree $T(V, E)$ before the structure change with picked edge e_{ij} connecting components C_1 and C_2, and randomly picked vertex $v_k \in C$

Let us look at $T(V, E)$ as a directed tree with the root vertex v_i. This tree before and after the structure change is illustrated in Figure 4.5. Consider the subgraph $S = G(V, E'')$ in $T'(V, E')$ where $E'' = E' \setminus \{e_{ij}, e_{ik}\}$. It can be seen that every vertex but v_j in this subgraph has the exact same descendants in T' as they have in T. Thus, since $E'' \subset E'$ and $E'' \subset E$, we can say that the number of paths that pass through any edge or any two edges in E'' is the same in T and T'. Similarly, regarding $e_{ik} \in E'$ and $e_{jk} \in E$, $\alpha'_{ik} = \alpha_{jk}$ and $\beta'_{ikrs} = \beta_{jkrs}$ for all $e_{rs} \in E''$. Hence, in all the three cases above, we see that e_{ij} is the only edge for which $\alpha'_{ij} \neq \alpha_{ij}$ and $\beta'_{ijrs} \neq \beta_{ijrs}$ where $e_{rs} \in E \cap E' \setminus e_{ij}$.

□

As an example, let us see how A changes when we make the structure change $SC(T, e_{79}, e_{93}, e_{73})$ in $T(V, E)$ in Figure 4.3a to get $T'(V, E')$ in Figure 4.3b. In equation (4.9), we see the rows and columns corresponding to the picked edge e_{79} change from A to A'. On the other hand, we see that the rows and columns corresponding to the removed edge e_{37} in A remain the same in A' (other than entries that belong to the rows and columns corresponding to e_{79} as well) where those rows and columns correspond to e_{39}.



(a) Tree before structure change

(b) Tree after structure change

Figure 4.5: Demonstration of the structure change $SC(T, e_{ij}, e_{ik}, e_{jk})$. Only v_j has a different number of descendants in T' than it has in T

$$
A = \begin{array}{c|ccccccccccc}
 & e_{01} & e_{02} & e_{03} & e_{24} & e_{25} & e_{36} & e_{37} & e_{78} & e_{79} & e_{9,10} & e_{9,11} & e_{9,12} \\
e_{01} & 12 & 3 & 8 & 1 & 1 & 1 & 6 & 1 & 4 & 1 & 1 & 1 \\
e_{02} & 3 & 30 & 24 & 10 & 10 & 3 & 18 & 3 & 12 & 3 & 3 & 3 \\
e_{03} & 8 & 24 & 40 & 8 & 8 & 5 & 30 & 5 & 20 & 5 & 5 & 5 \\
e_{24} & 1 & 10 & 8 & 12 & 1 & 1 & 6 & 1 & 4 & 1 & 1 & 1 \\
e_{25} & 1 & 10 & 8 & 1 & 12 & 1 & 6 & 1 & 4 & 1 & 1 & 1 \\
e_{36} & 1 & 3 & 5 & 1 & 1 & 12 & 6 & 1 & 4 & 1 & 1 & 1 \\
e_{37} & 6 & 18 & 30 & 6 & 6 & 6 & 42 & 7 & 28 & 7 & 7 & 7 \\
e_{78} & 1 & 3 & 5 & 1 & 1 & 1 & 7 & 12 & 4 & 1 & 1 & 1 \\
e_{79} & 4 & 12 & 20 & 4 & 4 & 4 & 28 & 4 & 36 & 9 & 9 & 9 \\
e_{9,10} & 1 & 3 & 5 & 1 & 1 & 1 & 7 & 1 & 9 & 12 & 1 & 1 \\
e_{9,11} & 1 & 3 & 5 & 1 & 1 & 1 & 7 & 1 & 9 & 1 & 12 & 1 \\
e_{9,12} & 1 & 3 & 5 & 1 & 1 & 1 & 7 & 1 & 9 & 1 & 1 & 12
\end{array}
\rightarrow
A' = \begin{array}{c|ccccccccccc}
 & e_{01} & e_{02} & e_{03} & e_{24} & e_{25} & e_{36} & e_{39} & e_{78} & e_{79} & e_{9,10} & e_{9,11} & e_{9,12} \\
e_{01} & 12 & 3 & 8 & 1 & 1 & 1 & 6 & 1 & 2 & 1 & 1 & 1 \\
e_{02} & 3 & 30 & 24 & 10 & 10 & 3 & 18 & 3 & 6 & 3 & 3 & 3 \\
e_{03} & 8 & 24 & 40 & 8 & 8 & 5 & 30 & 5 & 10 & 5 & 5 & 5 \\
e_{24} & 1 & 10 & 8 & 12 & 1 & 1 & 6 & 1 & 2 & 1 & 1 & 1 \\
e_{25} & 1 & 10 & 8 & 1 & 12 & 1 & 6 & 1 & 2 & 1 & 1 & 1 \\
e_{36} & 1 & 3 & 5 & 1 & 1 & 12 & 6 & 1 & 2 & 1 & 1 & 1 \\
e_{39} & 6 & 18 & 30 & 6 & 6 & 6 & 42 & 7 & 14 & 7 & 7 & 7 \\
e_{78} & 1 & 3 & 5 & 1 & 1 & 1 & 7 & 12 & 11 & 1 & 1 & 1 \\
e_{79} & 2 & 6 & 10 & 2 & 2 & 2 & 14 & 11 & 22 & 2 & 2 & 2 \\
e_{9,10} & 1 & 3 & 5 & 1 & 1 & 1 & 7 & 1 & 2 & 12 & 1 & 1 \\
e_{9,11} & 1 & 3 & 5 & 1 & 1 & 1 & 7 & 1 & 2 & 1 & 12 & 1 \\
e_{9,12} & 1 & 3 & 5 & 1 & 1 & 1 & 7 & 1 & 2 & 1 & 1 & 12
\end{array}
\tag{4.9}
$$

Lastly, we want the initial tree (the input tree of the optimisation scheme) to be a random one since we want to make sure that we obtain a reasonable optimal tree regardless of the initial tree. For this purpose, we do as follows.

Algorithm 8 Initial tree

1: **Input:**
2: $MST\,(K_n(V, \mathbf{E}, d))$: MST of the complete weighted graph
3: **Output:**
4: T_0 : the initial tree
5:
6: $T(V, E) \leftarrow MST(K_n(V, \mathbf{E}, d))$
7: **for** $i = 1$ to M **do**
8: $e_{ij} \leftarrow$ Picked from E uniformly at random
9: $C \leftarrow N(v_i) \cup N(v_j) \setminus \{v_i, v_j\}$
10: $v_k \leftarrow$ uniformly picked from C
11: **if** $v_k \in N(i)$ **then**
12: $T'(V, E') \leftarrow SC(T, e_{ij}, e_{jk}, e_{ik})$
13: **else**
14: $T'(V, E') \leftarrow SC(T, e_{ij}, e_{ik}, e_{jk})$
15: $T \leftarrow T'$
16: $T_0 \leftarrow T$

In the above, we make a large number M of random structure changes to the MST of the complete weighted graph. The resulting tree is the initial tree that we implement the optimisation algorithms on. Let us define the *for* loop of the above algorithm as $RT(T, M)$ which denotes the random tree that is obtained by M random structure changes on the tree T. We use it in the ILS optimisation scheme in the rest.

SA on Tree

Below is the SA algorithm on a tree with the picked edge e_{ij} for making a structure change. As mentioned above in Section 4.3.1, the larger the value of l (line 15 of Algorithm 9), the more biased towards smaller edges we are. The edge can also be picked uniformly at random. Also, *end_time* denotes the amount of time that we want to run the algorithm, and *time* is the amount of time already passed. The function that specifies the probability of accepting a worse tree in terms of RSS (see Section 2.3.1) is $P(RSS', RSS, t) = a_1 e^{-a_2 (\ln t)^{a_3} \frac{RSS' - RSS}{RSS'}}$ where the parameters a_1, a_2 and a_3 are tuned according to how often we are willing to accept a transition with a larger objective function value RSS' compared to the current RSS value.

Algorithm 9 SA on tree

Input:
  K_n : the complete weighted graph
  T_0 : initial tree structure
Output:
  $T_{optimal}$: the optimal tree

$t \leftarrow 0$
$T(V, E) \leftarrow T_0$
$\mathbf{w} \leftarrow$ Solve $A\mathbf{w} = \mathbf{d}$ via Cholesky decomposition
$RSS \leftarrow RSS(T, K_n)$
$RSS_{best} \leftarrow RSS(T, K_n)$
while $time < end\_time$ **do**
  $t \leftarrow t + 1$
  $\mathbf{e}^s \leftarrow$ column vector of edges in E sorted based on weight
  $e_{ij} \leftarrow e^s_{\lceil r^l n \rceil}$
  $C \leftarrow N(v_i) \cup N(v_j) \setminus \{v_i, v_j\}$
  $v_k \leftarrow$ uniformly picked from C
  if $v_k \in N(i)$ **then**
    $T'(V, E') \leftarrow SC(T, e_{ij}, e_{jk}, e_{ik})$
    $\mathbf{w}' \leftarrow$ Solve $A'\mathbf{w}' = \mathbf{d}'$ via Cholesky decomposition
  else
    $T'(V, E') \leftarrow SC(T, e_{ij}, e_{ik}, e_{jk})$
    $\mathbf{w}' \leftarrow$ Solve $A'\mathbf{w}' = \mathbf{d}'$ via Cholesky decomposition
  $RSS' \leftarrow RSS(T', K_n)$
  if $RSS' < RSS_{best}$ **then**
    $RSS_{best} \leftarrow RSS'$
    $T_{optimal} \leftarrow T'$
  if $RSS' < RSS$ **then**
    $A \leftarrow A'$
    $\mathbf{d} \leftarrow \mathbf{d}'$
    $RSS \leftarrow RSS'$
    $T \leftarrow T'$
  else if $P(RSS', RSS, t) > random(0, 1)$ **then**
    $A \leftarrow A'$
    $\mathbf{d} \leftarrow \mathbf{d}'$
    $RSS \leftarrow RSS'$
    $T \leftarrow T'$

ILS on Tree

In contrast to SA, as shown below, the picked edge e_{ij} in ILS is picked uniformly at random. The reason being we hope to try every possible structure change to "make

sure" the function RSS is stuck at a local minimum. The variable $count$ controls whether we have got stuck at a local minimum as we explain in the following. For any picked edge e_{ij}, the number of structure changes that we can make depends on $|C_i| = |N(v_i) \setminus \{v_j\}|$ and $|C_j| = |N(v_j) \setminus \{v_i\}|$. If we remove the edge e_{ij} from T, the resulting graph $G(V, E \setminus \{e_{ij}\})$ consists of two trees T_i and T_j where $v_i \in T_i$ and $v_j \in T_j$. We assume the average degree of a tree to be 2; thus, we assume the degree of both v_i and v_j to be 2. With this assumption, the number of possible structure changes based the edge e_{ij} is four, so for the whole tree we estimate it at $4n$. Let us define the k-th harmonic number as $H_k = \sum_{i=1}^{k} \frac{1}{i}$. If we try structure changes on a tree uniformly at random, the average number of times that we need to try all possible structure changes is $4nH_{4n}$—based on the well-known coupon collector's problem. That is why we set $4nH_{4n}$ as the threshold of $count$; after which, we modify the current local minimum by $RT(T, n)$.

Algorithm 10 ILS on tree

Input:
  K_n : the complete weighted graph
  T_0 : initial tree structure
Output:
  $T_{optimal}$: the optimal tree

$T(V, E) \leftarrow T_0$
$\mathbf{w} \leftarrow$ Solve $A\mathbf{w} = \mathbf{d}$ via Cholesky decomposition
$RSS \leftarrow RSS(T, K_n)$
$RSS_{best} \leftarrow RSS(T, K_n)$
while $time < end\_time$ **do**
    $e_{ij} \leftarrow$ Picked from E uniformly at random
    $C \leftarrow N(v_i) \cup N(v_j) \setminus \{v_i, v_j\}$
    $v_k \leftarrow$ uniformly picked from C
    if $v_k \in N(i)$ **then**
        $T'(V, E') \leftarrow SC(T, e_{ij}, e_{jk}, e_{ik})$
        $\mathbf{w}' \leftarrow$ Solve $A'\mathbf{w}' = \mathbf{d}'$ via Cholesky decomposition
    else
        $T'(V, E') \leftarrow SC(T, e_{ij}, e_{ik}, e_{jk})$
        $\mathbf{w}' \leftarrow$ Solve $A'\mathbf{w}' = \mathbf{d}'$ via Cholesky decomposition
    $RSS' \leftarrow RSS(T', K_n)$
    if $RSS' < RSS_{best}$ **then**
        $RSS_{best} \leftarrow RSS'$
        $T_{optimal} \leftarrow T'$
    if $RSS' < RSS$ **then**
        $count \leftarrow 0$
        $A \leftarrow A'$
        $\mathbf{d} \leftarrow \mathbf{d}'$
        $RSS \leftarrow RSS'$
        $T \leftarrow T'$
    else
        $count \leftarrow count + 1$
        if $count > 4nH_{4n}$ **then**
            $T \leftarrow RT(T, n)$
            $count \leftarrow 0$
            $\mathbf{w} \leftarrow$ Solve $A\mathbf{w} = \mathbf{d}$ via Cholesky decomposition
            $RSS \leftarrow RSS(T, K_n)$
            if $RSS < RSS_{best}$ **then**
                $RSS_{best} \leftarrow RSS$
                $T_{optimal} \leftarrow T$

4.4 Results

Our data set comprises stocks in S&P 500 from the 5-year period of 2013 to 2018. We used the formula $d_{ij} = \sqrt{2\left(1 - \rho_{ij}\right)}$—taken from Mantegna [1999]—to convert the correlation between stocks to distance. We applied SA and ILS (as in Algorithms 9 and 10 respectively) on this data set to evaluate the performance of these two metaheuristics in different scenarios. We evaluated whether bias towards smaller edges has any advantage in SA over no bias in SA. Then we compared the performance of SA and ILS—factoring in the underlying distribution of edge distances in different samples of stocks.

4.4.1 Biased vs Unbiased SA

As discussed before, if we set the value of l to 1 in SA, the algorithm picks an edge uniformly at random at each stage. In other words, the algorithm is not biased towards smaller edges. Otherwise, the larger the value of l, the more biased towards smaller edges the algorithm is. Below, we outline the result of the biased and unbiased runs of SA for samples of different sizes. For each sample, we do the biased SA by setting $l = 6$. Also, we ran the biased and unbiased SA on each sample 10 times; each starting from a random tree as per the initial tree algorithm (Algorithm 8).

Table 4.1: Biased vs unbiased SA on a complete weighted graph. For each tree, the metaheuristic with a better performance has been highlighted.

	Sample size					
	20		30		50	
Run	Unbiased	Biased	Unbiased	Biased	Unbiased	Biased
1	4.26687812	4.26687812	12.7055242	12.7055242	32.3461206	**32.1886367**
2	4.26687812	4.26687812	12.7798981	**12.7055242**	32.4564873	**31.7803389**
3	**4.26687812**	5.0111434	**12.7207733**	12.7464911	32.60437806	**31.6861588**
4	4.26687812	4.26687812	12.7055242	12.7055242	**31.85676394**	32.0511834
5	4.26687812	4.26687812	12.7055242	12.7055242	32.39138879	**31.6578321**
6	4.26687812	4.26687812	12.7343806	**12.7055242**	32.28023281	**32.0794777**
7	4.26687812	4.26687812	**12.7361441**	12.7615612	32.08112489	**31.9106985**
8	**4.26687812**	4.37441678	12.7615612	**12.7055242**	**31.92894628**	32.0031474
9	4.27373433	**4.26687812**	12.7207733	**12.7055242**	32.36184357	**31.9668845**
10	4.26687812	4.26687812	**12.7055242**	12.7207733	32.26931596	**31.9372013**
Average	**4.26756374**	4.35205851	12.7275627	**12.7167495**	32.25766022	**31.9261559**

In Table 4.1, we ran the algorithm on random samples of size 20, 30, and 50

respectively for ten minutes, two hours, and 18 hours. The values in this table are for the *RSS* function defined in equation (4.1)—according to which we evaluate the performance of the algorithm. The better performance—smaller *RSS*—for each sample in each run is highlighted. We can see that other than the random sample of size 20, in the other samples, the biased SA has an overall better performance. For the random sample with size 20, possibly since there are only 19 edges, the effect of biased or unbiased edge selection is less. Also, judging by how many times all the runs yield the same *RSS* value—if that value is the minimum compared to the other runs—there is not a considerable difference between biased and unbiased SA for the random sample of size 20. The reason being, there are 9 minimum values for unbiased and 8 for biased whereas, for the sample of size 30, 4 of the runs in unbiased SA and 7 in biased SA yield the minimum value of *RSS*. When the sample size is 50, the performance gap between unbiased and biased SA becomes wider—apparently because the importance of biased edge selection increases as the number of edges in the network goes up. It should be noted that we ran the unbiased and biased SA on other random samples, and we got similar results. Thus, apparently, biased edge selection indeed makes a difference. For the remainder, wherever we mention SA, we mean the biased SA.

4.4.2 SA vs ILS

As with the comparison between unbiased and biased SA, we compared the performance of SA and ILS based on ten runs of them over the same random samples. Look at Tables 4.2 and 4.3 for a performance comparison of SA and ILS. It can be seen in Table 4.2 that the performance of ILS is much better than that of SA. However, in Table 4.3, we can see that there is no apparent difference between the SA and ILS performances.

The reason for performance inconsistency of SA in Tables 4.2 and 4.3 seems to be the distance dispersion of samples used in each of them. For example, for the random sample of size 50 in each table, the dispersion of distance matrix entries is illustrated in Figure 4.6. It can be seen that for samples with high dispersion, SA and ILS have a similar performance while for samples with low dispersion, ILS maintains a solid performance while the performance of SA sharply decreases. It is noteworthy that for distance values with low dispersion, both biased and unbiased SA have a discouraging performance. The reason is probably that when distance values are close to each other, smaller distance values are not considerably smaller than the large distance values. Next, we discuss the structures of the resulting trees

Table 4.2: SA vs ILS on a complete weighted graph with low dispersion of distances. For each tree, the metaheuristic with a better performance has been highlighted.

	Tree size					
	20		30		50	
Run	SA	ILS	SA	ILS	SA	ILS
1	5.291951597	**4.84414153**	9.501749529	**8.4082242**	24.8579367	**16.8823573**
2	8.21566953	**4.84414153**	11.66835384	**8.4082242**	26.7081655	**16.8823573**
3	7.797470793	**4.84414153**	11.16650355	**7.97443788**	19.6536142	**16.8823573**
4	6.875995126	**4.84414153**	11.66835384	**7.97443788**	25.5941918	**16.8823573**
5	6.875995126	**4.84414153**	8.408224203	**7.97443788**	26.284257	**16.8823573**
6	6.019906558	**4.84414153**	12.98953771	**7.97443788**	30.2829294	**16.8823573**
7	7.016393465	**4.84414153**	10.3933439	**7.97443788**	34.8261135	**16.8823573**
8	7.016393465	**4.84414153**	11.80185307	**7.97443788**	31.0322914	**16.8823573**
9	**4.844141529**	5.2919516	13.02265027	**7.97443788**	31.1982311	**16.8823573**
10	7.016393465	**4.84414153**	10.58531443	**7.97443788**	26.265805	**16.8823573**
Average	6.697031065	**4.88892254**	11.12058844	**8.06119515**	27.6703536	**16.8823573**

Table 4.3: SA vs ILS on a complete weighted graph with high dispersion of distances. For each tree, the metaheuristic with a better performance has been highlighted.

	Tree size					
	20		30		50	
Run	SA	ILS	SA	ILS	SA	ILS
1	5.75665216	5.75665216	**12.7055242**	12.8162316	32.18863674	**31.8890846**
2	5.75665216	5.75665216	**12.7055242**	12.7495148	**31.78033885**	31.8140415
3	5.75665216	5.75665216	**12.7464911**	12.8550396	**31.68615883**	31.8986455
4	5.75665216	5.75665216	**12.7055242**	12.8345686	32.0511834	**31.9636283**
5	5.75665216	5.75665216	**12.7055242**	13.0379292	**31.65783212**	31.6641679
6	5.75665216	5.75665216	**12.7055242**	12.832506	32.07947769	**31.7900896**
7	5.75665216	5.75665216	**12.7615612**	12.8750193	**31.91069847**	32.0325964
8	5.75665216	5.75665216	**12.7055242**	12.8785592	**32.00314739**	32.2380063
9	5.75665216	5.75665216	**12.7055242**	12.7207733	**31.96688445**	31.9796543
10	5.93707406	**5.75665216**	**12.7207733**	12.8472376	31.93720134	**31.792651**
Average	5.77469435	**5.75665216**	**12.7167495**	12.8447379	31.92615593	**31.9062565**

of RSSOT.

4.4.3 Structure of RSSOT

The tree in Figure 4.7 is the RSSOT of a random sample of 20 stocks. It can be seen that it is a star (defined in Section 2.2.7). In Figure 4.8, we see different examples of RSSOT with different sizes, and all of them are a star or star-like. It appears that if you find the best star—the star with the smallest RSS value—for a set of stocks, that

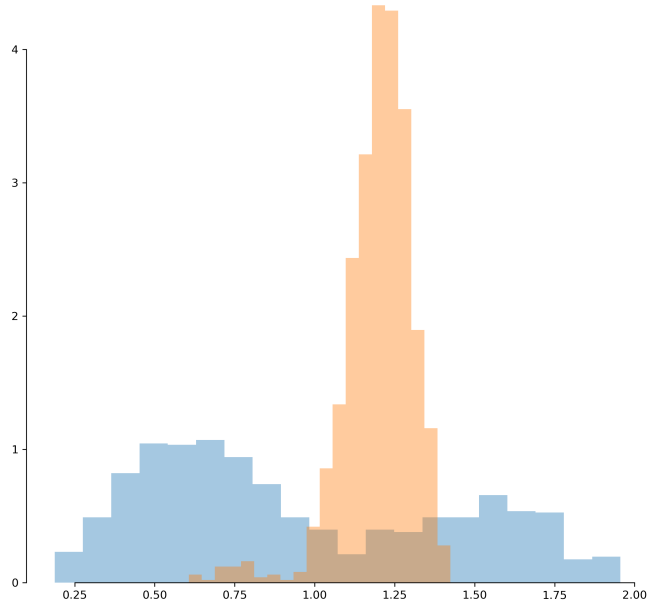


Figure 4.6: Dispersion of the sample of size 50 in Tables 4.2 and 4.3

star is either the RSSOT or a tree with an RSS value very close to RSSOT. Rephrased, it seems like the best star is never that far from RSSOT in terms of structure for these data sets. Even a random star for a set of stocks has a much smaller RSS value than a random tree that is not star-like for the same set of stocks.

In light of the above, we made a modification in the ILS procedure such that when the algorithm gets stuck in a local minimum, it starts from a random star—not just a random tree. We call this procedure ILS*. ILS* turned out dramatically faster than ILS. To provide a perspective, Table 4.4 illustrates the output of 10 runs of the typical ILS algorithm over a random sample of 100 stocks, each run for one week (10080 minutes) versus 10 6-hour runs of ILS* over the same sample. In can be seen in this table how ILS* far surpasses the typical ILS. The average RSS value is smaller in ILS*, and the RSS values are closer to each other although the algorithm is run only for six hours instead of one week.

Now we investigate the structural differences of resulting trees of Table 4.4. First, although the RSS value of the first and second runs in ILS* are the same (48.05502034), they do not yield the same star-like tree structure unless we consider the vertices unlabelled. Thus, the same RSS values do not necessarily reflect the

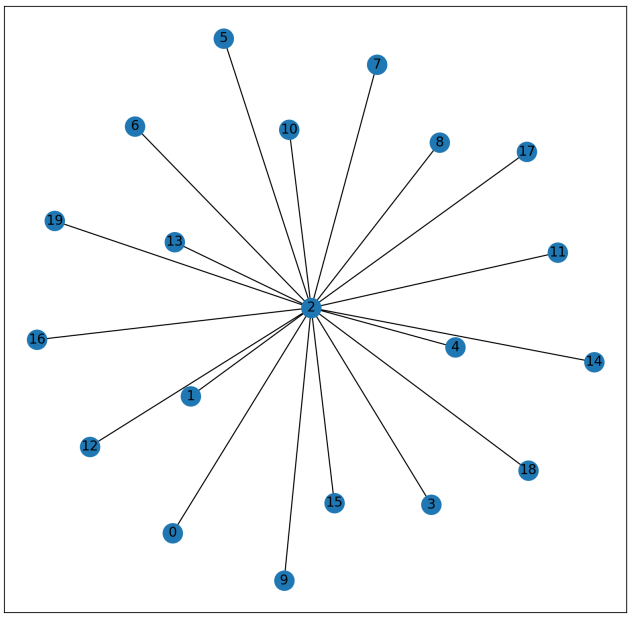


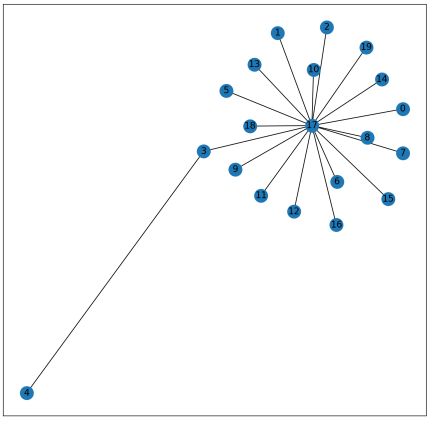
Figure 4.7: RSSOT of a random sample of 20 stocks

Table 4.4: ILS vs ILS* results of 10 runs on a random sample of 100 stocks.

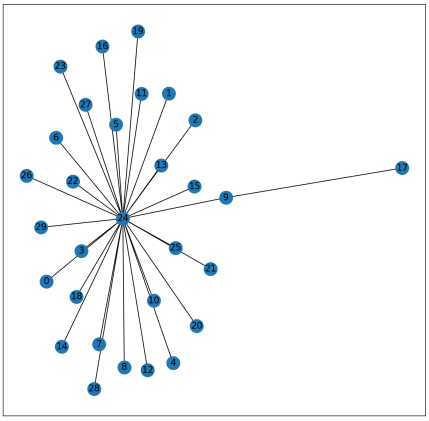
Run	ILS	ILS*
1	48.35469924	**48.05502034**
2	48.35469924	**48.05502034**
3	49.65978964	**48.05502034**
4	50.62349774	**47.7269775**
5	51.38174674	**47.7269775**
6	49.76388906	**48.05502034**
7	50.14204721	**47.66891197**
8	48.61622866	**48.05502034**
9	**47.66891197**	47.95537877
10	50.66033224	**48.05502034**
Average	49.52258417	**47.94083678**

same labelled tree structures. Second, although the RSS values are different in the 10 runs, they have very little structural differences. For example, Figure 4.9 shows the tree structure of runs 7 and 9 of the ILS* of Table 4.4.

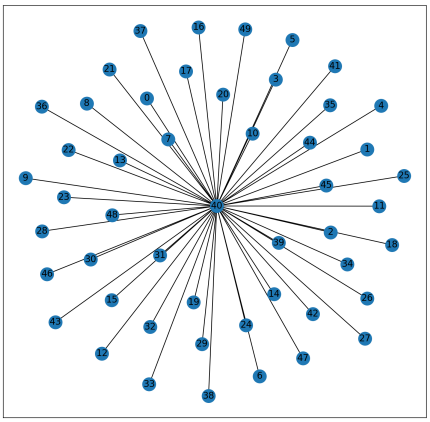
The clustering feature of RSSOT in the context of stock-correlation networks is poor. In other words, the star-like structure of these trees does not have a meaning-



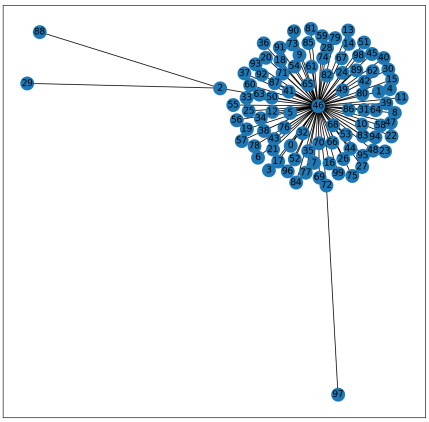
(a) 20 stocks



(b) 30 stocks



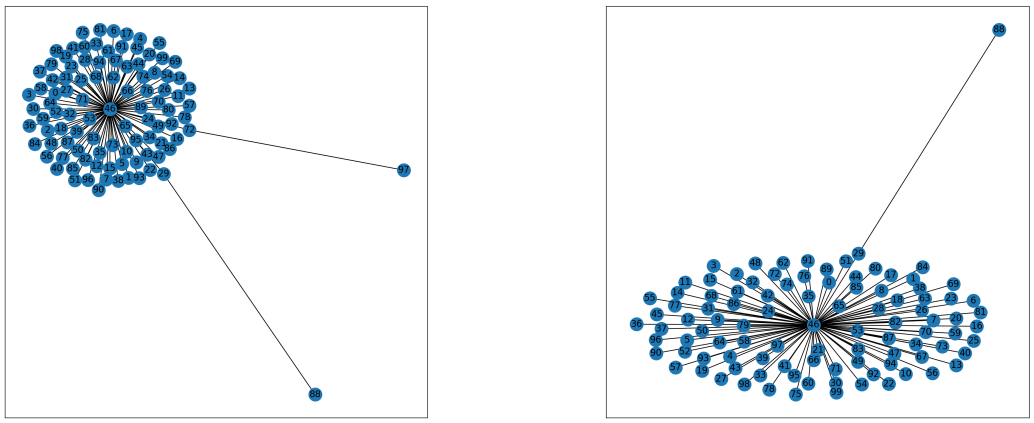
(c) 50 stocks



(d) 100 stocks

Figure 4.8: RSSOT of different random stock samples

ful interpretation in terms of clusters for stock-correlation networks. The underlying reason for the star-like structure seems to be that in stock-correlation networks, the distance values in the distance matrix are fairly similar. Also, the HT from the SLCA associated with these trees does not give us clusters that match the economic sectoral classification of stocks fairly well.



(a) Run 7 (b) Run 9

Figure 4.9: Structural difference of runs 7 and 9 in ILS* of Table 4.4

4.5 Summary

We presented a scheme to optimise the edge weights and structure of a tree to approximate a complete weighted graph using a measure involving the path distances in the tree. We have proposed a very efficient way of computing modifications to the tree that assist with local search metaheuristics, and evaluated the performance of two of these: SA and ILS. We showed that when the dispersion in distance values of the distance matrix—or equivalently K_n—is small, ILS has a solid performance while that of SA is inconsistent. However, if the dispersion in distance values is large, there is no apparent difference in the performance of SA and ILS. We also demonstrated that when the dispersion of distance values in the distance matrix is large, being biased in picking the smaller edges in each step of SA to make a structure change in the tree seems to have a slight advantage over unbiased picking of edges.

We applied this tree approximation problem on finding a network representation of stock sets. We found that the structure of this network is star-like, from which we cannot infer a meaningful interpretation of the clusters of stocks.

CHAPTER 5

Semi-Labelled Binary Tree Optimisation Subject to Non-Negativity

5.1 Introduction

We know from the previous chapters that the most common procedure to build a stock-correlation network is the minimum spanning tree (MST). We also explained in Sections 2.7.1 and 4.1 how MST is associated with the SLCA of stocks. The output of SLCA is a hierarchical tree (HT)—explained in Section 2.4.2—whose clusters agree reasonably well with the economic sectoral classification of stocks. If we join the two neighbours of the root in this HT and remove the two edges that join them to the root, we get a binary tree (defined in Section 2.2.7). So the tree structure of the output of SLCA is very similar to a binary tree as per our definition. (Actually, in accordance with the definition of binary tree in computer science, the output of SLCA is a complete binary tree.)

As mentioned in Chapter 4, in the HT that is the output of SLCA, the only factor that decides whether to join two vertices—or clusters—is the shortest distance between them taken from a distance matrix D. Since this HT can be easily modified into a binary tree as explained above, this brings up the question of how

to incorporate more of the information from all distances between vertices to build a binary tree and obtain the clusters based on this tree. We address this question as follows. We look for a binary tree where the path lengths between the leaves are as close as possible to their distance in D. Here the length of a path is defined as the total sum of edge weights on that path. To this end, we try to find an edge-weighted binary tree where the residual sum of squares (RSS) between the path lengths between leaves, and their corresponding distance in D, is minimised, subject to a non-negativity constraint on edge weights. The reason we opt for the non-negativity constraint is to make our approach comparable with the output HT of SLCA.

This problem is closely related to some variants of distance methods for phylogenetic tree inference (see Felsenstein [2004, Chapter 11] for an introduction to distance methods in this area)—in particular, ordinary least squares (OLS), weighted least squares (WLS), generalised least squares (GLS) and minimum evolution (ME). OLS solves the same problem as considered in this work but with no constraint on edge weights. OLS has been considered in the literature of phylogenetic tree inference, however, the other three methods above—WLS, GLS and ME—have been preferred in the context of this area (phylogenetic tree inference) as follows. Bulmer [1991] discusses that GLS is more efficient than OLS and WLS. Rzhetsky and Nei [1992] argues that unlike OLS and GLS, ME is statistically unbiased. Bryant and Waddell [1997] mention that WLS and GLS are more accurate methods of tree edge estimation. We need to mention that ME uses the same criteria as OLS for finding edge weights, yet the optimal tree is one with the least total sum of edge weights (see Rzhetsky and Nei [1993] and Desper and Gascuel [2004] for more details of this method).

To the best of our knowledge, there is still no satisfactory optimisation method for the OLS tree inference problem in which the edge weights are forced to be non-negative. In this work, we put forward a heuristic algorithm to search for such a weighted tree.

We want to apply our method which we have proposed for the aforementioned optimisation problem to stock data sets in order to obtain a good clustering of stocks with respect to their economic sectoral classification. However, this problem can be defined in general for any n objects as long as the distance matrix satisfies the triangle inequality—that is for any three objects i, j and k, $d_{ik} \leq d_{ij} + d_{jk}$.

In Section 5.2, we outline the problem and discuss how to address the negative edge weights. Section 5.3 contains our optimisation scheme which is iterated local

search (ILS) as described in Section 2.3.2. Section 5.4 discusses our results, and Section 5.5 is a summary of our findings.

5.2 Problem Statement

Let $D_{n\times n} = (d_{mk})$ be the distance matrix of n objects. We consider the binary tree $T(V, E, \boldsymbol{\omega})$ where the leaves correspond to those n objects. In this tree, V denotes the set of vertices, E denotes the set of edges, and $\boldsymbol{\omega}$ denotes the vector of weights of those edges where the entry ω_i is the weight of edge e_i. (We need to mention that unlike previous chapters, in this chapter, we denote each edge with one index instead of two.) This is a semi-labelled tree since only the leaf vertices are labelled. Such a tree has $K = 2n - 3$ edges—in which n is the number of leaves—with a total of $N = \binom{n}{2}$ paths connecting its leaves. In this tree, we index the leaf vertices from v_1 to v_n. We want to find the edge weights ω_i that minimise

$$RSS(T, D) = \sum_{\substack{m,k \\ 1\le m<k\le n}} \left(d_{mk} - \widehat{d}_{mk}\right)^2 \tag{5.1}$$

where $\widehat{d}_{mk}$ denotes the length of the path connecting the leaf vertices v_m and v_k. We define $q_{imk} = 1$ if e_i lies on the path between the leaf vertices v_m and v_k, and $q_{imk} = 0$ otherwise. Then we can rewrite the equation above as

$$RSS(T, D) = \sum_{\substack{m,k\in L \\ m<k}} \left(d_{mk} - \sum_{e_i\in E} q_{imk}\omega_i\right)^2. \tag{5.2}$$

In order to find the path lengths $\widehat{d}_{mk}$ that estimate the distance matrix entries, we need to find the optimal edge weights ω_i^*, and to do so, we define the following. First, $\boldsymbol{d}_{N\times 1}$ is a column vector corresponding to all the distances derived from the distance matrix $D_{n\times n}$. Second, $\boldsymbol{\omega}_{K\times 1}$ is a column vector corresponding to all the edge weights. Lastly, we define the binary matrix $Q_{N\times K}$ where the rows correspond to the entries of $\boldsymbol{d}_{N\times 1}$, the columns correspond to the entries of $\boldsymbol{\omega}$, and the entries correspond to the q values defined above. Then $RSS(T, D)$ can be re-formulated as

$$RSS(T, D) = (\boldsymbol{d} - Q\boldsymbol{\omega})^\intercal (\boldsymbol{d} - Q\boldsymbol{\omega}). \tag{5.3}$$

To optimise the edge weights with respect to $\boldsymbol{\omega}$, after getting the gradient vector of

RSS with respect to $\boldsymbol{\omega}$, we have

$$\nabla_{\boldsymbol{\omega}} RSS = -2Q^\intercal \boldsymbol{d} + 2Q^\intercal Q\boldsymbol{\omega} = 0 \tag{5.4}$$

from which we get the optimised edge weights $\boldsymbol{\omega}^*$ as

$$\boldsymbol{\omega}^* = \left(Q^\intercal Q\right)^{-1} Q^\intercal \boldsymbol{d}. \tag{5.5}$$

This is the standard solution for the ordinary least squares (OLS) method. This also means that $Q^\intercal Q$ requires to be invertible, but is it? The answer is, it is, and to prove it, we do as follows.

First, we prove that Q is full rank.

Lemma 5.2.1. *Q is full rank.*

Proof. The proof is taken from Mihaescu and Pachter [2008, Lemma 1]. Let us define the row vector $\boldsymbol{v}_r^\intercal = (0, \ldots, 1, \ldots, 0)$ of length $|E|$ for the binary tree $T(V, E)$ such that the entry that corresponds to the edge e_r is 1, and all the other entries are zero. Also, let P_{ab} be the edge sequence that corresponds to the path that connects the leaves v_a and v_b. We define $\boldsymbol{p}_{ab}^\intercal$ as the row vector in Q that corresponds to P_{ab}. We choose leaves v_a, v_b, v_c and v_d such that $P_{ab} \bigcap P_{cd} = \emptyset$ and $P_{ad} \bigcap P_{bc} = \{e_r\}$. Then $\boldsymbol{v}_r^\intercal = \frac{1}{2}\left(\boldsymbol{p}_{ad}^\intercal + \boldsymbol{p}_{bc}^\intercal - \boldsymbol{p}_{ab}^\intercal - \boldsymbol{p}_{cd}^\intercal\right)$. So $\boldsymbol{v}_r^\intercal$ is in the row space of Q, and Q is full rank. □

Remember the definition of p-norm from Section 2.4.1. Let $Q^\intercal Q\boldsymbol{x} = \boldsymbol{0}$. Then $\boldsymbol{x}^\intercal Q^\intercal Q\boldsymbol{x} = (Q\boldsymbol{x})^\intercal (Q\boldsymbol{x}) = \|Q\boldsymbol{x}\|_2^2 = 0 \implies Q\boldsymbol{x} = \boldsymbol{0}$. Since Q is full rank, this means that the only solution to $Q^\intercal Q\boldsymbol{x} = \boldsymbol{0}$ is $\boldsymbol{x} = \boldsymbol{0}$. So $Q^\intercal Q$ is also full rank. Then $\det(Q^\intercal Q) \neq 0$ (see Lipschutz and Lipson [2001, Theorem 8.5] as to why the determinant should be non-zero). Thus, our assumption that $Q^\intercal Q$ is invertible was correct, and using equation (5.5), we have $\widehat{\boldsymbol{d}} = Q\boldsymbol{\omega}^*$.

5.2.1 Normal Equation

Since Q is full rank, $\forall \boldsymbol{x} \neq \boldsymbol{0}$, $(Q\boldsymbol{x})^\intercal (Q\boldsymbol{x}) = \boldsymbol{x}^\intercal Q^\intercal Q\boldsymbol{x} > 0$. Hence, $A = Q^\intercal Q$ is positive-definite (defined in Section 2.5). Let $\boldsymbol{b} = Q^\intercal \boldsymbol{d}$. Then we have the normal equation $A\boldsymbol{\omega} = \boldsymbol{b}$, and we can get the optimal edge weights $\boldsymbol{\omega}^*$, as in equation (5.5) using Cholesky decomposition (2.6). This is going to be an integral part of our optimisation scheme in the rest of this work.

Since $A = Q^\intercal Q$, and $\boldsymbol{b} = Q^\intercal \boldsymbol{d}$, the entries of A and $\boldsymbol{b}$ are as follows. The

diagonal entry a_{ii} is the number of paths passing through the edge e_i corresponding to ω_i in $\boldsymbol{\omega}$. The entry b_i in $\boldsymbol{b}$ is the sum of distances corresponding to these paths taken from the distance matrix D. Lastly, a_{ij} is the number of paths passing through edges e_i and e_j. The question is, how do we get those paths? In other words, how can we efficiently get the entries of A and $\boldsymbol{b}$ from the tree structure and the distance matrix?

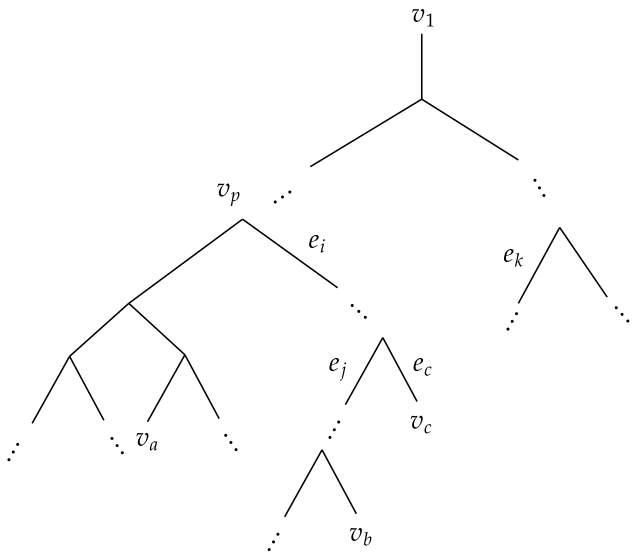


Figure 5.1: An example binary Tree T with n leaves. We take one of the leaves (v_1) as the root and take T as a directed tree. Vertex v_p is one of the internal vertices, so $p > n$, and v_a, v_b and v_c are three of the leaf vertices, so $a, b, c \leq n$. Edge e_c is a leaf edge, and e_i, e_j and e_k are three of the internal edges.

Consider the edges e_i and e_j in the tree T with n leaves. Let us assume that T is directed and take the leaf vertex v_1 as its root. Remember that we index the leaf vertices from v_1 to v_n, that is $L = \{v_1, \ldots, v_n\}$. We define L^i for the internal edge e_i as the set consisting of the descendants of the bottom vertex of e_i that are leaf vertices. Here, the bottom vertex of e_i refers to the one further from the root vertex v_1. For a leaf edge e_i other than that with one end being the root vertex v_1, L^i denotes the set containing only the bottom vertex of e_i. For example, for the leaf edge e_c, as in Figure 5.1, L^c is the set consisting of only the bottom vertex—which is a leaf vertex—of e_c. So $L^c = \{v_c\}$.

Then the following formulas give us the entries of A and b.

$$a_{ii} = \left|L^i\right| \left|L \setminus L^i\right| \tag{5.6}$$

$$b_i = \sum_{i,j} d_{ij} \qquad v_i \in L^i, v_j \in L \setminus L^i \tag{5.7}$$

$$a_{ij} = \begin{cases} \left|L^i\right| \left|L^j\right| & L^i \cap L^j = \emptyset \\ \left|L^i\right| \left|L \setminus L^j\right| & L^i \subset L^j \\ \left|L^j\right| \left|L \setminus L^i\right| & L^j \subset L^i \end{cases} \tag{5.8}$$

For example, for the tree T in Figure 5.1, in matrix A associated with this tree, $a_{ik} = \left|L^i\right| \left|L^k\right|$, and $a_{ij} = \left|L^j\right| \left|\left(L \setminus L^i\right)\right|$.

5.2.2 Nearest Neighbour Interchange (NNI)

NNI is an operation used in neighbourhood search of tree structures based on swapping two subtrees (defined in Section 2.2.7) on two ends of an internal edge. We use this operation to obtain neighbouring tree structures of a binary tree in our optimisation scheme.

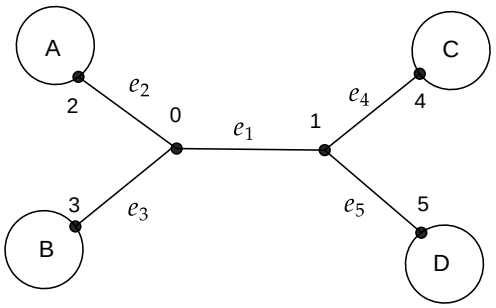
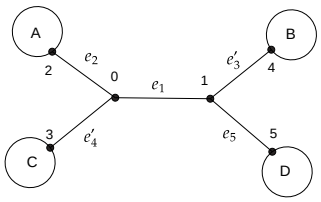


Figure 5.2: A general binary tree with leaf-sets A, B, C , and D corresponding to the neighbouring edges of the internal edge e_1

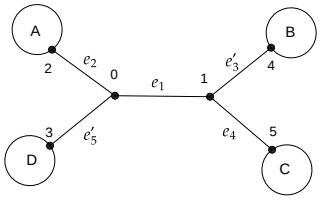
Consider the tree in Figure 5.2 in which A, B, C and D denote the leaves in the four subtrees containing the neighbouring edges of e_1. If we perform NNI based on e_1, the possible resulting trees are T_1 and T_2 in Figure 5.3a and 5.3b respectively. We define $NNI_1(T, e_1)$ and $NNI_2(T, e_1)$ to denote these two trees.

5.2.3 Making Negative Edge Weights Positive

If an optimal edge weight is negative, we would like to be able to make it positive with minimum alteration to the structure of the binary tree and such that all edges



(a) T_1



(b) T_2

Figure 5.3: The two possible tree structures after performing NNI on the tree in Figure 5.2 based on the edge e_1

still have optimal weight. Here, we want to show that for any binary tree, if an internal edge weight is negative, in at least one of the resulting trees after an NNI structure change based on that edge, that edge weight is going to be positive after optimising the edge weights. (We should point out that by fixing one negative edge weight, we may as well make neighbouring edge weights negative.) It suggests a possible step in our algorithm to avoid negative-weighted edges. Let us denote the optimal weight of e_1 in Figure (5.2) by ω_1. Let us also denote the optimal weight of this edge in Figure (5.3a) and (5.3b) by ω'_1 and ω''_1. We want to show that if $\omega_1 < 0$, at least one of ω'_1 and ω''_1 is positive. To this end, we use equations (2)–(4) in Rzhetsky and Nei [1993] to calculate the edge weights as follows.

For the tree in Figure 5.2, let us denote the leaf sets corresponding to the four neighbouring subtrees of e_1 by A, B, C and D. Let us also denote the average distance based on the distance matrix between the leaf vertices in two leaf sets such as A and B by $\delta_{AB} = \frac{1}{|A||B|} \sum_{v_i \in A, v_j \in B} d_{ij}$. Then the optimal weight of e_1 is given by

$$\omega_1 = 1/2 \left[\alpha_1 \left(\delta_{AC} + \delta_{BD}\right) + \left(1 - \alpha_1\right) \left(\delta_{AD} + \delta_{BC}\right) - \left(\delta_{AB} + \delta_{CD}\right)\right] \tag{5.9}$$

where $\alpha_1 = \frac{|A||D|+|B||C|}{(|A|+|B|)(|C|+|D|)}$. Also, the optimal weight of a leaf edge as shown in Figure 5.4 is calculated as

$$\omega_i = 1/2 \left[\delta_{Ci} + \delta_{Di} - \delta_{CD}\right]. \tag{5.10}$$

Lemma 5.2.2. *In a general binary tree, let ω_1, ω'_1 and ω''_1 denote the optimal weight of the internal edge e_1 for the three possible tree structures based on NNI around e_1. If $\omega_1 < 0$, at least one of ω'_1 and ω''_1 is strictly positive.*

Proof. We use equation (5.9) to calculate ω'_1 in Figure 5.3a by swapping B and C.

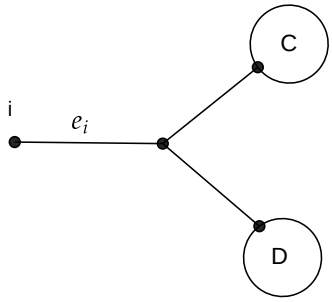


Figure 5.4: The leaf edge/vertex of a general binary tree

We also use this equation to calculate ω_1'' in Figure 5.3b by swapping B and D. We see that $\omega_1 + \omega_1' + \omega_1'' = 0$. So at least one of the weights ω_1' and ω_1'' is strictly positive. □

Regarding the leaf edges, since the triangle inequality holds for distances between leaves (as mentioned in the second last paragraph of Section 5.1) this inequality also holds for the average distance between sets of leaves. In light of this, the weight of a leaf edge as calculated in equation (5.10) is always non-negative.

5.2.4 Finding Tree Edge Weights after NNI

Let $T'(V, E', \boldsymbol{\omega}')$ be the binary tree resulting from performing NNI on $T(V, E, \boldsymbol{\omega})$ based on the edge e_i—basically, T' is either $NNI_1(T, e_i)$ or $NNI_2(T, e_i)$. It can be seen that according to equation (5.9), the only edge weights that are different in T' versus T are the weights of e_i and its set of neighbours defined by $N(T', e_i)$—or equivalently $N(T, e_i)$. This gives us

$$\omega_k' = \omega_k, \quad \forall e_k \notin \{e_i\} \cup N(T, e_i). \tag{5.11}$$

This is useful since knowing this, evaluating each tree structure to find its corresponding RSS can be done much faster.

5.2.5 Substituting Negative Edge Weights with Zero

It is evident that after optimising the edge weights, if we simply substitute a negative edge weight with zero, the resulting RSS value is smaller than substituting

with any other non-negative value. Now we want to show that substituting a negative edge weight with zero and holding it fixed while re-optimising the other edge weights is better than substituting that edge weight with any other non-negative value and similarly re-optimising the other edge weights. This gives us another technique for trying to avoid negative-weighted edges.

Lemma 5.2.3. *Substitution of the optimal edge weight $\omega_i^* < 0$ with zero and re-optimising the other edge weights will lead to a smaller RSS value than substitution of ω_i^* with any strictly positive value and re-optimising the other edge weights.*

Proof. Let $\boldsymbol{\omega}^*$ be the vector of optimised edge weights for the binary tree T with n leaf vertices and K edges, and let $\boldsymbol{\omega}_i^* < 0$. We swap the index of e_i with that of e_K. So now $\omega_K^* < 0$. We then generate the tree topology matrix Q where the order of the column vectors correspond to the order of edge weights in the vector $\boldsymbol{\omega}$. Also, let $\widehat{d}_{ij}$ denote the length of the path connecting leaf vertices v_i and v_j in T. Then we have

$$Q\boldsymbol{\omega} = \left[\begin{array}{c|c} \widetilde{Q}_{N\times(K-1)} & \begin{array}{c}|\\ \boldsymbol{q}_K \\ |\end{array} \end{array}\right] \left[\begin{array}{c} \widetilde{\boldsymbol{\omega}}_{(K-1)\times 1} \\ \hline \omega_K \end{array}\right] = \widehat{\boldsymbol{d}} \tag{5.12}$$

where $\widetilde{Q}$ is Q with column q_k removed. We write this as $\widetilde{Q} = Q\,(;q_k)$.

Let $e_{ij} = d_{ij} - \widehat{d}_{ij}$ be the path length error between leaf vertices v_i and v_j. It can then be seen that

$$\boldsymbol{e} = \boldsymbol{d} - Q\boldsymbol{\omega} \tag{5.13}$$

where $\boldsymbol{e}$ is the vector of errors for every path and $\boldsymbol{d}$ is a column vector derived from the distance matrix. Thus, $RSS = \boldsymbol{e}^\intercal\boldsymbol{e}$, and we can say that

$$RSS = \left(\boldsymbol{d} - \widetilde{Q}\widetilde{\boldsymbol{\omega}} - \boldsymbol{q}_K\omega_K\right)^\intercal \left(\boldsymbol{d} - \widetilde{Q}\widetilde{\boldsymbol{\omega}} - \boldsymbol{q}_K\omega_K\right). \tag{5.14}$$

Now, we want to find the optimal edge weights with the value of ω_K fixed. We have

$$\begin{aligned} &\nabla_{\widetilde{\omega}} RSS = 2\widetilde{Q}^\intercal\widetilde{Q}\widetilde{\boldsymbol{\omega}} + 2\boldsymbol{q}_K^\intercal\widetilde{Q}\omega_K - 2\boldsymbol{d}^\intercal\widetilde{Q} = \boldsymbol{0} \\ &\implies \widetilde{\boldsymbol{\omega}}^* = \left(\widetilde{Q}^\intercal\widetilde{Q}\right)^{-1}\left(\widetilde{Q}^\intercal\boldsymbol{d} - \widetilde{Q}^\intercal\boldsymbol{q}_K\omega_K\right) \end{aligned} \tag{5.15}$$

which means that

$$\boldsymbol{d} - \widetilde{Q}\widetilde{\boldsymbol{\omega}}^* - \boldsymbol{q}_K\omega_K = \boldsymbol{d} - \boldsymbol{q}_K\omega_K + \underbrace{\widetilde{Q}\left(\widetilde{Q}^\intercal\widetilde{Q}\right)^{-1}\widetilde{Q}^\intercal}_{H}\left(\boldsymbol{q}_K\omega_K - \boldsymbol{d}\right). \tag{5.16}$$

Notice that H is a symmetric idempotent matrix. Taking that into account, if we replace the above in equation (5.14), we get

$$\begin{aligned} RSS = &\left(\boldsymbol{q}_K^\intercal \boldsymbol{q}_K - \boldsymbol{q}_K^\intercal H \boldsymbol{q}_K\right) \omega_K^2 + \\ &2\left(\boldsymbol{d}^\intercal H \boldsymbol{q}_K - 2\boldsymbol{d}^\intercal \boldsymbol{q}_K\right) \omega_K + \boldsymbol{d}^\intercal \boldsymbol{d} - \boldsymbol{d}^\intercal H \boldsymbol{d}. \end{aligned} \tag{5.17}$$

We can see that the RSS value is a simple convex quadratic function of ω_K for which $\omega_K^* < 0$. Thus, it is evident that the value of RSS for $\omega_K = 0$ is smaller that that of $\omega_K > 0$. □

Next, we want to show that if we get a negative edge weight, using NNI to get that edge weight positive—based on Lemma 5.2.2—will lead to a smaller RSS value compared to making that weight zero and re-optimising the other edge weights. This gives us a direction for handling a negative weighted edge depending on the situation.

Lemma 5.2.4. *Suppose $\omega_i^* < 0$ in the binary tree T(V,E). Let f be the RSS value of this tree after setting $\omega_i = 0$ and re-optimising the other edge weights. Also, let f_1 and f_2 be the RSS values—after optimising edge weights—of the trees generated by performing NNI on $T(V, E)$ around e_i. Then $f_1 \leq f$, and $f_2 \leq f$.*

Proof. Suppose $w_1^* < 0$ in $T(V, E)$ with the topology matrix Q in Figure 5.2. We generate both trees $T_1(V, E_1)$ and $T_2(V, E_2)$ in Figure 5.3a and 5.3b respectively by performing NNI on T. $E_1 = E \cup \{e_3', e_4'\} \setminus \{e_3, e_4\}$ and $E_2 = E \cup \{e_3', e_5'\} \setminus \{e_3, e_5\}$. Let Q' and Q'' denote the topology matrices of T_1 and T_2 respectively. It can be seen that the columns corresponding to the edges in Q' and Q'' are the same as those in Q except the column corresponding to e_1. We swap this column with the last column—$\boldsymbol{q}_k$, $\boldsymbol{q}_k^{'}$ and $\boldsymbol{q}_k^{''}$—in all the three matrices. Then we have

$$Q(;\boldsymbol{q}_k) = Q'(;\boldsymbol{q}_k^{'}) = Q''(;\boldsymbol{q}_k^{''}) = \widetilde{Q} \tag{5.18}$$

where $Q(;\boldsymbol{q}_k)$ is as defined in the previous lemma. Thus, if $w_i = 0$ in T, $\widehat{\boldsymbol{d}} = \widetilde{Q}\widetilde{\boldsymbol{\omega}}$ where $\widetilde{\boldsymbol{\omega}}$ is as defined in equation (5.12). However, $\widehat{\boldsymbol{d}}^{'} = \widetilde{Q}\widetilde{\boldsymbol{\omega}} + \boldsymbol{q}_k^{'}\omega_K$ and $\widehat{\boldsymbol{d}}^{''} = \widetilde{Q}\widetilde{\boldsymbol{\omega}} + \boldsymbol{q}_k^{''}\omega_K$ in T_1 and T_2 respectively. It means that if $\omega_K = 0$,

$$\widehat{\boldsymbol{d}} = \widehat{\boldsymbol{d}}^{'} = \widehat{\boldsymbol{d}}^{''} \implies f = f_1 = f_2.$$

(Another way to see this is by observing that in Figure 5.2, 5.3a and 5.3b, the trees are essentially the same if the weight of e_1 is zero.) Thus, the upper bound of f_1 and f_2 is f. □

We see using Lemmas 5.2.2 and 5.2.4, that in at least one of the trees resulting from performing NNI around e_i in $T(V, E)$, $\omega_i > 0$, and the RSS value is smaller than or equal to f.

5.3 Optimisation Scheme

As mentioned in the above in Section 5.1, we use an ILS metaheuristic to search for the best tree. As a quick reminder, the overall ILS procedure consists of the repeated application of the following two steps: first, we improve the current configuration until we appear to get stuck in the local minimum, second, we start the search over by perturbing the current configuration in a major way.

In the context of our work, these two steps are as follows. As to the first step, for n objects, we start with an initial random binary tree T. We pick a random edge e_i, and perform NNI based on that to get a new tree structure such that $T' \leftarrow NNI_1(T, e_i)$ and $T'' \leftarrow NNI_2(T, e_i)$ as defined in Section 5.2.2. If $RSS(T', D) < RSS(T, D)$, we accept T' as the better tree (as in $T \leftarrow T'$) and repeat the same procedure as above (picking a random edge and performing NNI based on that). Otherwise, if $RSS(T'', D) < RSS(T, D)$, we accept T'' as the better tree and repeat the procedure. We continue until we assume we are stuck in a local minimum. At this point, we work on getting an all-non-negative-edge-weighted tree (explained in the next section). Regarding the second step, we perturb the local minimum by performing $2n$ random NNI's—roughly equal to the number of edges—without calculating the edge weights. We may assume it gives a significantly different tree.

To decide how long to do the first step, we use the well-known coupon collector's problem: there are $g(n) = 4n - 6$ ways to apply NNI, so we repeat NNI approximately $g(n)H_{g(n)}$ times before assuming we are in a local minimum where H_x denotes the x-th Harmonic number. Since our algorithm is a heuristic, we do not need to be absolutely certain of this before moving on.

5.3.1 Efficient Path Length Calculation

In each step of ILS, we need to calculate the path length $\widehat{d}_{ij}$ between any two leaf vertices to obtain RSS. To do this efficiently, we do as follows. For a binary tree T, we take v_1 as the root of the tree. Then we use the depth-first search (DFS) algorithm (explained in Section 2.2.7) to find all the paths between v_1 and the other leaves $\{v_2, \ldots, v_n\}$. In other words, we find $P_{1,2}, P_{1,3}, \ldots, P_{1,n}$ and accordingly their lengths $\widehat{d}_{12}, \widehat{d}_{13}, \ldots, \widehat{d}_{1n}$. Next, for any two leaf vertices v_i and v_j other than the root

v_1, let $P_{1,i} \cap P_{1,j} = R^{ij}$, and let $\left|R^{ij}\right| = \sum_{e_k \in R^{ij}} \omega_k$ be the length of this path. Then it can be seen that

$$\widehat{d}_{ij} = \widehat{d}_{1,i} + \widehat{d}_{1,j} - 2\left|R^{ij}\right|. \tag{5.19}$$

For example, in Figure 5.1, $\widehat{d}_{ab} = \widehat{d}_{1a} + \widehat{d}_{1b} - 2\left|R^{ab}\right|$. We find all the path lengths between leaves in this manner and call this algorithm Path Length Computation (*PLC*)—peseudocode as in Algorithm 11.

Algorithm 11 *PLC*

Input:
  $T(V, E, \boldsymbol{\omega})$: binary tree
Output:
  $\widehat{D}$: matrix of path lengths between any two leaves

Set v_1 as the root of T
$P_{1,2}, P_{1,3}, \ldots, P_{1,n} \leftarrow$ paths from v_1 to all the other leaves using DFS
$\widehat{d}_{12}, \widehat{d}_{13}, \ldots, \widehat{d}_{1n} \leftarrow$ respective path lengths of $P_{1,2}, P_{1,3}, \ldots, P_{1,n}$
for $i \in \{2, 3, \ldots, n-1\}$ **do**
  for $j \in \{i+1, \ldots, n\}$ **do**
    $R^{ij} = P_{1,i} \cap P_{1,j}$
    $\widehat{d}_{ij} = \widehat{d}_{1,i} + \widehat{d}_{1,j} - 2 \sum_{e_k \in R^{ij}} \omega_k$

5.3.2 Working around Negative Weighted Edges

The next question is what do we do about the negative weighted edges after landing in a local minimum? As proved in Lemma 5.2.2, if we perform NNI on the tree corresponding to the local minimum $T(V, E, \boldsymbol{\omega})$ around the edge e_i where $\omega_i < 0$, this edge weight is going to be non-negative in at least one of $T_1 \leftarrow NNI_1(T, e_i)$ and $T_2 \leftarrow NNI_2(T, e_i)$. Also, as proved in Lemma 5.2.4, T_1 and T_2 have a smaller RSS than $T'(V, E, \boldsymbol{\omega}')$ in which the edges are the same as those in T, but $w_i = 0$, and all the other edge weights have been re-optimised accordingly.

Thus, it seems like we need to repeatedly perform NNI on the negative-weighted edges of T to get an all-non-negative-edge-weighted tree. Yet there is one challenge. The four neighbouring edges of e_i, after performing NNI based on this edge in T, might change sign from positive to negative in T_1 and T_2. We also know based on equation (5.11) that in these two trees, all the other edge weights do not change value. In light of this, we do as follows.

Let us define the total negativity of a set of edges $\mathbb{E} \subseteq E$ as the total sum of all the negative edge weights corresponding to $\mathbb{E}$ in T. That is

$$s^-(T, \mathbb{E}) = \sum_{e_i \in \mathbb{E}^-} \omega_i \tag{5.20}$$

in which $\mathbb{E}^- = \{e_i \in \mathbb{E} : \omega_i < 0\}$. For $T(V, E, \boldsymbol{\omega})$ corresponding to the local minimum, we try to reduce the total negativity but trying to not disturb the tree structure too much from the "optimal" one found. Starting from T, we want to get a tree with less total negativity—that is a larger s^-. Let $E^- = \{e_i \in E : \omega_i < 0\}$. For $e_i \in E^-$, we accept the new tree structure $T_1 \leftarrow NNI_1(T, e_i)$, that is $T \leftarrow T_1$, if

$$s^-\left(T_1, \{e_i\} \cup N(T_1, e_i)\right) > s^-\left(T, \{e_i\} \cup N(T, e_i)\right).$$

Otherwise, if

$$s^-\left(T_2, \{e_i\} \cup N(T_2, e_i)\right) > s^-\left(T, \{e_i\} \cup N(T, e_i)\right),$$

we accept T_2. We continue this procedure repeatedly until after going through all the negative-weighted edges in T, there is no change in its structure. We call this algorithm NNI^-, the overall layout of which is demonstrated in Algorithm 12.

5.3.3 Substitution with Zero

The last question is, what do we do with the tree resulting from Algorithm 12 since we cannot reduce its total negativity by performing NNI any more? We use the result of Lemma 5.2.3 and substitute the negative edge weights with zero repeatedly in the following manner. Let $T(V, E, \boldsymbol{\omega})$ be the tree corresponding to the local minimum after applying NNI^-, and $E^- = \{e_i \in E : \omega_i < 0\}$. Also, let $T(e_i^0)$ be the tree after setting $\omega_i = 0$ in T, and $T^*(e_i^0)$ be $T(e_i^0)$ after re-optimising all the other edge weights. For $e_i \in E^-$, if $s^-\left(T^*(e_i^0), E\right) > s^-\left(T(e_i^0), E\right)$, we accept $T^*(e_i^0)$ as the new tree structure ($T \leftarrow T^*(e_i^0)$), otherwise, $T \leftarrow T(e_i^0)$. This procedure continues until we get a tree T such that after going through all its negative-weighted edges, we do not need to re-optimise the other edge weights. We should mention that to re-optimise the other edge weights, that is edges other than e_i where $\omega_i = 0$, we use Cholesky decomposition to solve the normal equation (explained in Section 5.2.1) after removing the column in A and the entries in $\boldsymbol{\omega}$ and $\boldsymbol{b}$ corresponding to e_i. We call this the *Subzero* algorithm whose pseudocode is in Algorithm 13.

Algorithm 12 NNI^{-}

Input:
  $T(V, E, \omega)$: binary tree
  D : distance matrix
Output:
  $T(V, E', \omega')$: tree with less total negativity

$E^{-} \leftarrow \{e_i \in E : \omega_i < 0\}$
$flag \leftarrow 0$
while $flag = 0$ **do**
  $flag \leftarrow 1$
  for $e_i \in E^{-}$ **do**
    for $T_{temp}(V, E', \omega') \in \{NNI_1(T, e_i), NNI_1(T, e_i)\}$ **do**
      Find all $\omega_k^{'}$ for $e_k \in \{e_i\} \cup N(T_{temp}, e_i)$ using equations (5.9) and (5.10)
      if $s^{-}\left(T_{temp}, \{e_i\} \cup N(T_{temp}, e_i)\right) > s^{-}\left(T, \{e_i\} \cup N(T, e_i)\right)$ **then**
        $\omega_k \leftarrow \omega_k' \quad \forall e_k \in E' \setminus \left(\{e_i\} \cup N(T_{temp}, e_i)\right)$
        $T \leftarrow T_{temp}$
        $flag \leftarrow 0$
        Break
    if flag=0 **then**
      $E^{-} \leftarrow \{e_i \in E : \omega_i < 0\}$
      Break

5.3.4 The ILS Layout

In a nutshell, we have our whole optimisation scheme as in Algorithm 14. In this algorithm, which we call non-negative tree (NNT), T_{best} and f_{best} denote the best tree without any constraint on the sign of the edge weights and its corresponding RSS value respectively. Similarly, T_{best}^{+} and f_{best}^{+} denote the best tree with non-negativity constraint on its edge weights and its corresponding RSS value. The algorithm run time is specified by the user. We should point out that wherever in this pseudocode we calculate RSS, we are using the PLC algorithm, as explained in Section 5.3.1, as part of it. In lines 15–31 of NNT, we do the first step of the ILS procedure, that is improving until stuck in the local minimum. In particular, in lines 17 and 18, it can be seen that in each NNI operation only five edge weights (as explained above) need to be calculated. In line 34, we check whether the algorithm is stuck in a local minimum. In lines 35 and 36, the algorithm gets rid of the negative weighted edges by using the NNI^{-} (Algorithm 12) and $Subzero$ (Algorithm 13) respectively—as explained in Sections 5.3.2 and 5.3.3. Then in lines 38–40, it finds the best tree with all non-negative edge weights and its respective RSS up to that point. Lastly, in

Algorithm 13 *Subzero*

Input:
  $T(V, E, \omega)$: binary tree
  D : distance matrix
Output:
  $T(V, E', \omega')$: tree with all non-negative edge weights

$E^- \leftarrow \{e_i \in E : \omega_i < 0\}$
$flag \leftarrow 0$
while $flag = 0$ **do**
  $flag \leftarrow 1$
  for $e_i \in E^-$ **do**
    $T \leftarrow T(e_i^0)$
    Form $\widetilde{A}\widetilde{\omega} = \widetilde{b}$ and solve it using the Cholesky decomposition
    if $s^-\left(T^*(e_i^0), E\right) > s^-\left(T, E\right)$ **then**
      $T \leftarrow T^*(e_i^0)$
      $flag \leftarrow 0$
      $E^- \leftarrow \{e_i \in E : \omega_i < 0\}$
      Break

lines 41 to 43, the algorithm perturbs the current local minimum to start again.

Algorithm 14 *NNT*

Input:
  $D_{n\times n}$: distance matrix
  $time$: amount of time we want to run the algorithm
Output:
  $T^+_{best}(V, E, \omega)$: Optimal tree with all non-negative edge weights

$T(V, E) \leftarrow$ random binary tree with n leaves
Find the edge weights of T using equations (5.9) and (5.10)
$f \leftarrow RSS(T, D)$
$f_{best} \leftarrow f$
$T_{best} \leftarrow T(V, E, \omega)$
$f^+_{best} \leftarrow$ a very large number
$counter \leftarrow 0$

5.4 Results

We ran our ILS optimisation scheme (*NNT*) on three binary trees with different number of leaves. The data is taken from random samples of stocks belonging to

while $passed\_time < time$ **do**
  $e_i \leftarrow$ one random internal edge of T
  for $T_{temp}(V, E', \omega') \in \{NNI_1(T, e_i), NNI_2(T, e_i)\}$ **do**
    Find $\omega'_j \quad \forall e_j \in \{e_i\} \cup N(T_{temp}, e_i)$ using equations (5.9) and (5.10)
    $\omega'_j \leftarrow \omega_j \quad \forall e_i \in E' \setminus \big(\{e_i\} \cup N(T_{temp}, e_i)\big)$
    $f_{temp} \leftarrow RSS(T_{temp}, D)$
    if $f_{temp} < f_{best}$ **then**
      $T_{best} \leftarrow T_{temp}$
      $f_{best} \leftarrow f_{temp}$
      $T \leftarrow T_{temp}$
      $f \leftarrow f_{temp}$
      $counter \leftarrow 0$
      Break
    else if $f_{temp} < f$ **then**
      $T \leftarrow T_{temp}$
      $f \leftarrow f_{temp}$
      $counter \leftarrow 0$
      Break
    else
      $counter \leftarrow counter + 1$
      if $counter > 4nH_{4n}$ **then**
        $T^+ \leftarrow NNI^-(T, D)$
        $T^+ \leftarrow subzero(T^+, D)$
        $f^+ \leftarrow RSS(T^+, D)$
        if $f^+ < f^+_{best}$ **then**
          $T^+_{best} \leftarrow T^+$
          $f^+_{best} \leftarrow f^+$
        $T \leftarrow$ perform $2n$ NNI's on T
        Find the edge weights of T using equations (5.9) and (5.10)
        $f \leftarrow RSS(T, D)$
        $counter \leftarrow 0$
        Break

S&P 500, and the distance matrix has been derived from the pairwise correlation coefficient between those stocks. For each tree, we ran our scheme ten times in parallel, and we recorded the best RSS values as follows: those corresponding to the optimisation problem with a non-negativity constraint on the edge weights and those with no constraint on the edge weights. The former is given by RSS^{+}_{best} and the latter by RSS_{best} in our scheme, respectively.

Table 5.1: Constrained (C) vs unconstrained (U) ILS optimisation scheme on binary trees with different number of leaves

	Number of leaves					
	20		30		50	
Run	C	U	C	U	C	U
1	0.1485	0.1183	0.3912	0.3365	1.4075	1.0625
2	0.1477	0.1164	0.3934	0.3446	1.4169	1.0683
3	0.1482	0.1212	0.3934	0.3378	1.3663	1.0827
4	0.1470	0.1172	0.3960	0.3415	1.4049	1.0649
5	0.1470	0.1168	0.3910	0.3381	1.3514	1.1094
6	0.1495	0.1162	0.3904	0.3444	1.3401	1.0621
7	0.1483	0.1178	0.3931	0.3486	1.3820	1.0731
8	0.1484	0.1185	0.3958	0.3360	1.3585	1.0798
9	0.1491	0.1150	0.3934	0.3421	1.3863	1.0728
10	0.1479	0.1151	0.3927	0.3435	1.3831	1.0534

In Table 5.1, we see the RSS^{+}_{best} and RSS_{best} values for each of the trees with 20, 30, and 50 leaves in their respective columns—C and U. In this table, we ran our scheme for 30 minutes, six hours, and 18 hours for the three trees respectively. The RSS results in each column are reasonably close to one another which is behavior one would expect from a reasonably effective heuristic algorithm. Also, as expected, the RSS results corresponding to the constrained problem are larger than those for the unconstrained problem in each tree. We should lastly mention that we repeated this scheme for trees with different corresponding distance matrices, and we got similar results.

5.4.1 Tree Structure

In order to analyse how well NNT performs compared to the output of SLCA (which we denote for the rest of the chapter by SLCA with a slight abuse of notation) we did as follows. We considered both trees—NNT and SLCA—undirected and unweighted, and we did the comparison based on the average distance, that is the length of the shortest path, of stocks belonging to the same economic sector in both trees. We call this the *sector average distance (SAD)*. However, this is still not a fair comparison since for n leaves, NNT has $2n-3$ edges while SLCA has $2n-2$. To address this, we removed the root of SLCA (the vertex that used to be the root when the tree was considered directed) and joined its two neighbours through an edge. Basically, we modified SLCA into a binary tree as explained in the first paragraph of this chapter. This way, both trees have $2n-3$ edges. For the remainder of the chapter, we denote these two trees with the above mentioned modifications by NNT^m and SLCA^m.

Figures 5.5 and 5.6 are respectively the visualisations of the NNT^m and $SLCA^m$ of a random sample of 70 stocks where the leaves have been labelled by the sector of their corresponding stock. In Table 5.2, we have calculated the SAD of these two trees. The entries in column "Size" in this table refer to the number of stocks belonging to their corresponding rows' sector in both trees. It can be seen that most SAD values for NNT^m are significantly smaller than their counterparts for SLCA^m. Otherwise, there is not a considerable difference in the two values. We investigated varied random samples of different sizes, and we got similar results. This difference might be explained by the chain phenomenon—explained in Section 2.4.2—of SLCA as it can also be seen in Figure 5.6. Thus, our measure suggests that NNT^m outperforms SLCA^m in identifying stocks belonging to the same sector. However, when we use ALCA or CLCA instead of NNT, we do not get the same results.

Let us define ALCA^m and CLCA^m as trees obtained from ALCA and CLCA in the same manner SLCA^m is obtained from SLCA. It seems like NNT^m does not have any advantage over them based on the SAD criterion. Table 5.3 demonstrates the SAD values over a random stock sample of size 40 for CLCA^m and NNT^m. Based on this example, it can be seen that the advantage of either of these two methods over the other is not evident. We performed CLCA and ALCA on different random samples of different sizes, and it is not evident that NNT is better than them based on the SAD measure.

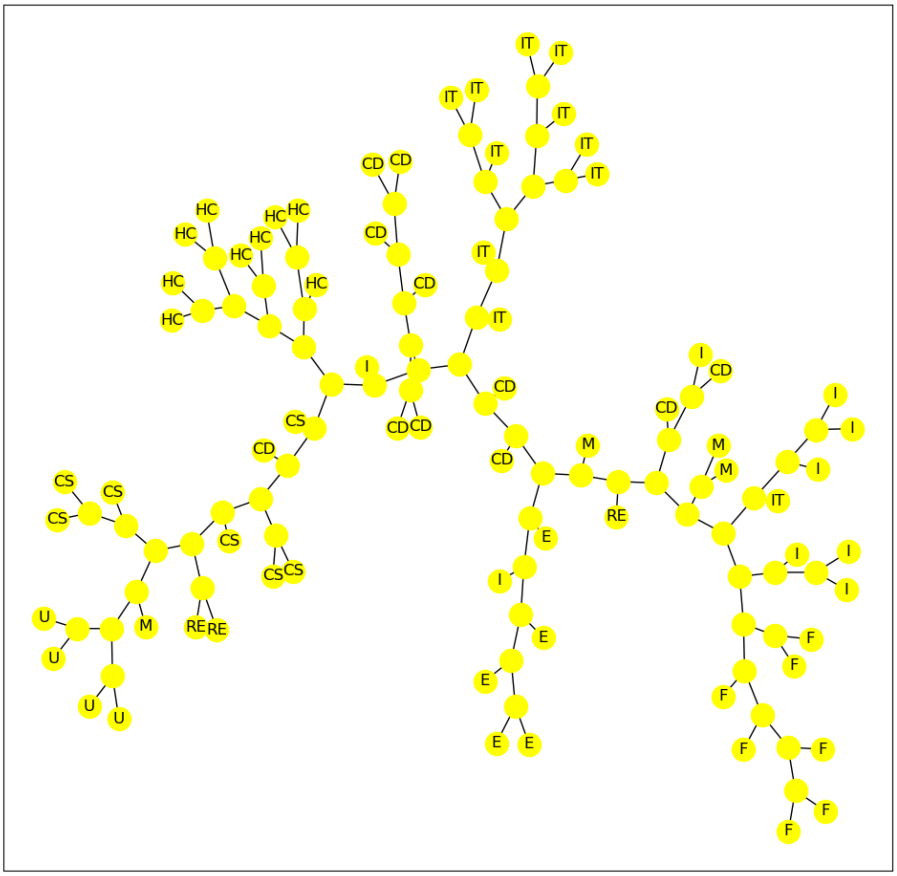


Figure 5.5: The NNT^m of a random sample of 70 stocks. The leaves have been labelled with the sector of the stocks.

5.5 Summary

We proposed an iterated local search (ILS) scheme to optimise the edge weights and structure of a binary tree subject to a non-negativity constraint on edge weights. We showed how to calculate the edge weights in each step of the ILS procedure efficiently. We also showed that performing NNI around a negative-weighted edge can make its optimal weight positive, and doing so is better than substituting its weight with zero and then re-optimising the other edge weights. Lastly, we showed that substituting a negative edge weight with zero is better than substituting with

Table 5.2: SAD of the NNT^m and $SLCA^m$ in Figures 5.5 and 5.6 respectively

Sector	Size	$SLCA^m$ SAD	NNT^m SAD
E	5	9.800	4.000
HC	9	18.000	5.111
CD	11	16.982	8.145
F	7	4.857	4.571
U	4	3.167	3.333
CS	7	16.286	5.809
M	4	23.000	11.667
IT	11	17.818	6.982
I	9	18.833	8.944
RE	3	11.333	11.333

Table 5.3: SAD of the NNT^m and $CLCA^m$ of a random stock sample of size 40. The columns are defined the same as those in Table 5.2.

Sector	Size	$CLCA^m$ SAD	NNT^m SAD
CD	6	6.933	7.133
CS	5	7.200	4.800
E	2	7.000	9.000
F	6	4.000	4.000
HC	3	2.667	5.333
I	7	6.952	7.238
IT	5	6.800	5.600
M	2	7.000	14.000
RE	2	2.000	13.000
U	2	2.000	2.000

any other non-negative value—whether we re-optimise the other edge weights or not. Accordingly, when stuck in a local minimum, we proposed the heuristic algorithms NNI^- and *subzero* to get a tree with all-non-negative edge weights. Based on ten parallel runs of our scheme on different random samples with different

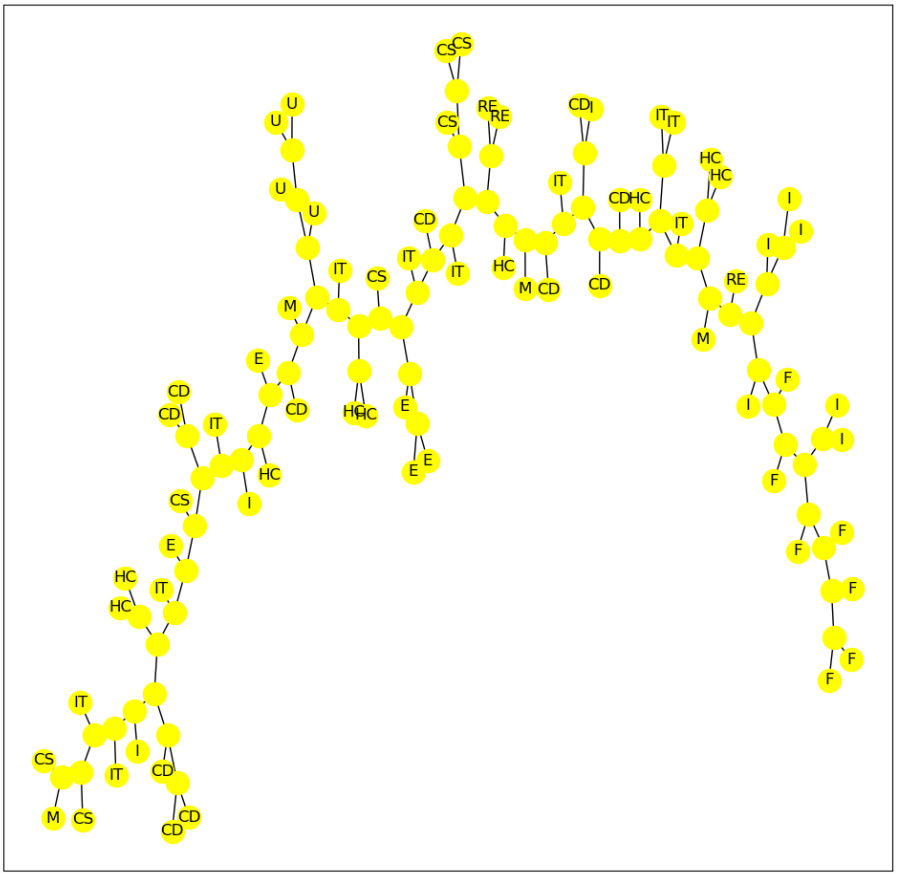


Figure 5.6: The SLCA^m of a random sample of 70 stocks. The leaves have been labelled with the sector of the stocks.

sizes, we saw that our scheme demonstrates a decent performance.

We applied our scheme on different random samples of stocks to find a better clustering of stocks with regards to their economic sectoral classification. Our method was able to recognise the clusters better than SLCA, but failed to do so compared to CLCA and ALCA. That means our scheme does not have an evident edge over CLCA and ALCA. Since these two algorithms—CLCA and SLCA—are much faster and simpler than our algorithm, our analysis gives no reason to prefer NNT over them.

CHAPTER 6

Concluding Remarks

In this thesis, we investigated the area of stock-correlation networks. In studying the dynamics of the stock market, it is crucial to find stocks that demonstrate a similar behaviour. In the context of stock market, behaviour refers to the increase and drop in stock prices. To quantify this similarity in behaviour, Pearson correlation coefficient is often used in this area which gives a value in the interval $[-1, 1]$. If the correlation coefficient of the *log*-returns of two stocks is 1, they have the exact same price pattern. If this value is -1, their price action is the exact opposite of one another, and a value of 0 denotes no correlation. In order to extract meaningful information from the pairwise correlation of stocks, one way is to illustrate the most important interactions between the stocks in form of a graph, and that is what a stock-correlation network refers to. That is because the complete weighted graph of stocks cannot give us a direct insight (at least visually) due to its highly interwoven structure.

6.1 Overview of the Previous Algorithms

We mentioned that several algorithms have been put forward to build stock-correlation networks—or equivalently to extract the most important stocks interactions. The following is a recap of theses algorithms and their pros and cons. Afterwards, we review how this thesis is related to them, the results we got, and some potential topics for prospective researchers in this area.

The most widely used and first ever proposed of these algorithms is MST which also corresponds to a specific hierarchical clustering of stocks—SLCA. It has been argued in the literature that the output of SLCA matches the economic sectoral classification of stocks fairly well. It has also been discussed in the literature that dynamics of the market can be tracked by MST. Take for example when there is a market crisis, the MST shrinks and it looks more star-like. The downside of this algorithm mentioned in the literature is the structural limitation imposed on the network which means that we have to exclude many edges that connect highly correlated stocks only for the sake of maintaining the tree structure. Also, the chain phenomenon associated with SLCA is another shortcoming mentioned regarding the clustering of MST. To address this shortcoming, another algorithm that has been proposed in the literature is the average linkage MST (ALMST). This tree is derived from the ALCA of stocks, so it does not have the disadvantage of SLCA. However, it has been discussed that the edges in MST are more reliable than those in ALMST based on the bootstrap technique.

The second mostly used algorithm is arguably PMFG. It has been mentioned in the literature that one of the pros of PMFG is that it always contains MST. Since it also has more edges than MST, it contains more information. Moreover, having MST as its subset means that it also demonstrates the hierarchical structure of stocks to a good extent. Lastly, it has 3 and 4-cliques, the analysis of which can show highly connected stocks. These cliques have been investigated in the literature, and it has been found that they demonstrate a high homogeneity: a good proportion of stocks in them belong to the same economic sector. The downside of PMFG has been argued to be the same as that of MST. In other words, PMFG cannot include heavy edges in the network merely in order to maintain its planar structure.

Lastly, the threshold method algorithm has been proposed which only includes edges with a larger correlation weight than a threshold in the network—or edges with a distance edge smaller than a particular threshold. The threshold is usually set such that the degrees of vertices follow a power law distribution. The advantage of this algorithm is that the clusters can be analysed by investigating the cliques components. It also includes the anti-correlated stocks which can help in portfolio optimisation and asset diversification.

The above mentioned algorithms are the most widely used ones in the literature. However, other algorithms have also been proposed for generating stock-correlation networks including, but not limited to, the directed bubble hierarchical tree (DBHT), p-median problem, triangulated maximally filtered graph (TMFG),

and maximum likelihood. Moreover, correlation coefficient has not been the only measure used to quantify the similarity in the behaviour of stocks. Its variants such as partial correlation coefficient has also been utilised. The downside to the correlation coefficient is that it only accounts for the linear correlation between stocks, and it has been argued in the literature that the relationship between stocks has a nonlinear component. In light of this, other measures such as mutual information and its variants have been applied to factor in the nonlinear relationship between stocks to account for their similarity.

6.2 Our Contribution

All in all, for each algorithm suggested in this area, some pros and cons have been mentioned. Of all the pros, a good clustering seems to have got the most attention. Therefore, we set the goal of this thesis in coming up with algorithms for generating stock-correlation networks such that they provide a better clustering of stocks compared to the most widely used algorithms—MST and PMFG—in the literature. In Chapter 3, we came up with an algorithm to generate a stock-correlation network. In Chapter 4, we tried to use a tree that approximates a complete weighted graph as a stock-correlation network. Lastly, in Chapter 5, we tried to come up with a better clustering of stocks compared to the previous works in the literature. Below, we give a summary of these chapters and their corresponding results.

In Chapter 3, we proposed the proportional degree (PD) algorithm where the degree of each vertex is proportional to the total sum of its corresponding stock correlation with all other stocks. We can set the total number of edges in this algorithm to any value, and for the sake of comparing with PMFG, we set it at $3n - 6$ in which n is the number of vertices (or stocks). We also used the normalised mutual information (NMI) to account for the nonlinear relationship between stocks. PD does not have the structural limitations of PMFG, and unlike the latter, the size of cliques is not bound to four. We also showed that the cliques in this algorithm have a significantly higher homogeneity as to the economic sectoral classification of stocks compared to PMFG. This higher homogeneity also holds when using correlation coefficient rather than NMI. We also used Louvain community detection and normalised spectral clustering (NSC) to compare the clustering properties of PD and PMFG. Louvain did not provide us with a meaningful evaluation. However, based on NSC, we saw that the clusters of PD match those achieved through applying NSC on the complete weighted graph of stocks, better than PMFG. Last but not least, we showed that PD maintains the clustering structure better than

PMFG in the absence of some random vertices. As such, its clusters are more robust than those in PMFG. We should point out that PD may find applications as means of analysis other than clustering in the area of stock-correlation networks (discussed below in Section 6.3).

In Chapter 4, we devised an algorithm to build a tree to approximate a complete weighted graph. We used this tree as a stock-correlation network since it uses more information from the pairwise correlation between stocks compared to MST in the following sense. Let K_n represent the complete edge weighted graph of distances between stocks. (These distance values have been derived from the pairwise correlation between stocks.) We want to come up with a tree such that the distances between every two vertices in this tree best estimate the distances (or the edge weights) between them in K_n. We have used the residual sum of squares (RSS) as the optimality criterion, because of which we call this algorithm the RSS optimal tree (RSSOT). We have come up with efficient methods to calculate the edge weights on a given tree and to switch between different structures to find the optimal tree. We used two metaheuristics for this optimisation problem: Simulated Annealing (SA) and Iterated Local Search (ILS). It turns out that when the dispersion of distance values in K_n is large, both metaheuristics have a favourable performance. In contrast, when the dispersion of distance values is small, ILS outperforms SA. As mentioned above, we applied RSSOT on pairwise stock correlations, but it did not give a useful structure and clustering in the end.

Finally, in Chapter 5, we devised another algorithm as with Chapter 4 to estimate K_n. However, this time the edge weights have a non-negativity constraint and the tree is binary such that the leaves represent the stocks, and the internal vertices are unlabelled. We called this algorithm the non-negative tree (NNT). We proposed NNT as an improvement over SLCA in the same manner that we proposed RSSOT as an improvement over MST. We came up with methods to avoid negative weighted edges and efficiently search through different tree structures to find the optimal structure using an ILS scheme. We compared the resulting tree with that of the output of SLCA for different random samples of stocks, and we argued that NNT offers a better clustering of stocks than SLCA. However, the same is not evident when we compare the clustering performance of NNT with CLCA and ALCA. As such, since these two algorithms are much simpler and faster than NNT, we do not see any reason to use NNT over them.

We saw that PD resulted in a stock-correlation network with advantages over PMFG. However, RSSOT could not be favoured over MST as a stock-correlation

network. Also NNT could not be preferred over the output of ALCA or CLCA as a means to clustering stocks. These two algorithms, RSSOT and NNT, were the result of suggesting a solution to interesting problems, but they failed to outperform some previously suggested algorithms of generating stock-correlation networks and clustering.

6.3 Future Topics of Research

Finally, we offer some potential topics for future research before finishing this final chapter as follows.

- We used $3n - 6$ edges to build the PD network for the whole purpose of comparing its performance with its PMFG counterpart. It is not clear that this number of edges in the PD network is the optimal one considering the criteria of a superior stock-correlation network (see Section 2.7). Since our method is versatile—in terms of the number of edges as input—a potential topic for research can be varying the sparsity of edges in the PD algorithm and comparing the resulting networks.
- Other measures of correlation and dependence across stocks such as the Spearman's rank correlation coefficient (see Corder and Foreman [2014, Chapter 7] for definition) could be used to build a stock-correlation network and compare that with other stock-correlation networks according to the criteria used in the literature.
- As discussed in Chapter 2, stock-correlation networks—especially the MST and threshold types—have been employed to study the behaviour of stocks in financial crises [Lee and Nobi, 2018; Nobi et al., 2014; Majapa and Gossel, 2016; Kumar and Deo, 2013; Junior and Franca, 2012]. Our study has been mostly from a clustering standpoint, and the performance of PD compared to other methods in studying stock-correlation networks during financial crises and market crash remains to be seen.
- We did not encounter any study in this area where returns were calculated taking into account the properties of economic-sector-specific noise. We suggest studying stock-correlation networks with this consideration.
- The two of the most commonly used algorithms for generating stock-correlation networks, MST and PMFG, have arisen in other areas (in particular, graph theory) unrelated to the area of stock-correlation networks. Other algorithms for filtering information in graph theory such as graph sparsification (see

Spielman and Teng [2004, Section 1.3] for a brief introduction) can potentially be used for building stock-correlation networks and evaluating the results.

- In PD, we determine the degree of each vertex in such a way that it is linearly proportional to the total sum of correlation between that vertex and all the other vertices. It would be interesting to see what happens if we set them such that they are polynomially proportional rather than linear, especially since the power law distribution of degrees is an integral part of many complex networks, and it has also been studied in stock-correlation networks. It would be of interest to study the clustering performance of such a network besides other aspects considered in the area.
- We used the ordinary least square (OLS) in both Chapters 4 and 5 as the optimality criterion. It would be interesting to see how the results would change if we used the weighted least squares (WLS) or generalised least squares (GLS)—as mentioned in Chapter 5—instead, especially in RSSOT. Would it result in useful tree structures and clusters in RSSOT?
- We applied RSSOT on stock-correlation networks but did not get clusters that fairly match the economic sectoral classification of stocks. What are other potential areas in which RSSOT can be applied to solve a problem?

Bibliography

Albert, R. and Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47.

Albert, R., Jeong, H., and Barabási, A.-L. (1999). Internet: Diameter of the world-wide web. *nature*, 401(6749):130.

Aloise, D., Deshpande, A., Hansen, P., and Popat, P. (2009). Np-hardness of euclidean sum-of-squares clustering. *Machine learning*, 75(2):245–248.

Amini, A. A., Chen, A., Bickel, P. J., Levina, E., et al. (2013). Pseudo-likelihood methods for community detection in large sparse networks. *The Annals of Statistics*, 41(4):2097–2122.

Arai, Y., Yoshikawa, T., and Iyetomi, H. (2015). Dynamic stock correlation network. *Procedia Computer Science*, 60:1826–1835.

Barabási, A.-L., Albert, R., and Jeong, H. (2000). Scale-free characteristics of random networks: the topology of the world-wide web. *Physica A: statistical mechanics and its applications*, 281(1-4):69–77.

Barbi, A. and Prataviera, G. (2019). Nonlinear dependencies on brazilian equity network from mutual information minimum spanning trees. *Physica A: Statistical Mechanics and its Applications*, 523:876–885.

Bastian, M., Heymann, S., and Jacomy, M. (2009). Gephi: An open source software for exploring and manipulating networks.

Birch, J., Pantelous, A. A., and Soramäki, K. (2016). Analysis of correlation based networks representing dax 30 stock price returns. *Computational Economics*, 47(4):501–525.

Blanc, J.-L., Pezard, L., and Lesne, A. (2011). Delay independence of mutual-information rate of two symbolic sequences. *Physical Review E*, 84(3):036214.

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008.

Blum, C. and Raidl, G. R. (2016). *Hybrid Metaheuristics: Powerful Tools for Optimization*. Springer.

Boginski, V., Butenko, S., and Pardalos, P. M. (2005). Statistical analysis of financial networks. *Computational statistics & data analysis*, 48(2):431–443.

Bonanno, G., Caldarelli, G., Lillo, F., and Mantegna, R. N. (2003). Topology of correlation-based minimal spanning trees in real and model markets. *Physical Review E*, 68(4):046130.

Boss, M., Elsinger, H., Summer, M., and Thurner 4, S. (2004). Network topology of the interbank market. *Quantitative finance*, 4(6):677–684.

Brandes, U., Delling, D., Gaertler, M., Gorke, R., Hoefer, M., Nikoloski, Z., and Wagner, D. (2007). On modularity clustering. *IEEE transactions on knowledge and data engineering*, 20(2):172–188.

Brida, J. G. and Risso, W. A. (2010). Dynamics and structure of the 30 largest north american companies. *Computational Economics*, 35(1):85.

Bryant, D. J. and Waddell, P. J. (1997). Rapid evaluation of least squares and minimum evolution criteria on phylogenetic trees.

Buccheri, G., Marmi, S., and Mantegna, R. N. (2013). Evolution of correlation structure of industrial indices of us equity markets. *Physical Review E*, 88(1):012806.

Bulmer, M. (1991). Use of the method of generalized least squares in reconstructing phylogenies from sequence data.

Carlsson, G. E. and Mémoli, F. (2010). Characterization, stability and convergence of hierarchical clustering methods. *J. Mach. Learn. Res.*, 11(Apr):1425–1470.

Chen, H., Mai, Y., and Li, S.-P. (2014). Analysis of network clustering behavior of the chinese stock market. *Physica A: Statistical Mechanics and its Applications*, 414:360–367.

Chi, K. T., Liu, J., and Lau, F. C. (2010). A network perspective of the stock market. *Journal of Empirical Finance*, 17(4):659–667.

Chopard, B. and Tomassini, M. (2018). *An introduction to metaheuristics for optimization*. Springer.

Coletti, P. (2016). Comparing minimum spanning trees of the italian stock market using returns and volumes. *Physica A: Statistical Mechanics and its Applications*, 463:246–261.

Corder, G. W. and Foreman, D. I. (2014). *Nonparametric statistics: A step-by-step approach*. John Wiley & Sons.

Coronnello, C., Tumminello, M., Lillo, F., Micciche, S., and Mantegna, R. N. (2005). Sector identification in a set of stock return time series traded at the london stock exchange. *arXiv preprint cond-mat/0508122*.

Cover, T. M. and Thomas, J. A. (2012). *Elements of information theory*. John Wiley & Sons.

da Silva, F. L., Pijn, J. P., and Boeijinga, P. (1989). Interdependence of eeg signals: linear vs. nonlinear associations and the significance of time delays and phase shifts. *Brain topography*, 2(1-2):9–18.

Desper, R. and Gascuel, O. (2004). Theoretical foundation of the balanced minimum evolution method of phylogenetic inference and its relationship to weighted least-squares tree fitting. *Molecular Biology and Evolution*, 21(3):587–598.

Dorogovtsev, S. N. and Mendes, J. F. (2001). Scaling properties of scale-free evolving networks: Continuous approach. *Physical Review E*, 63(5):056125.

Dorogovtsev, S. N. and Mendes, J. F. (2002). Evolution of networks. *Advances in physics*, 51(4):1079–1187.

Even, S. (2011). *Graph algorithms*. Cambridge University Press.

Everitt, B. S., Landau, S., Leese, M., and Stahl, D. (2011). Cluster analysis 5th ed.

Felsenstein, J. (2004). *Inferring phylogenies*, volume 2. Sinauer associates Sunderland, MA.

Fiedor, P. (2014). Networks in financial markets based on the mutual information rate. *Physical Review E*, 89(5):052801.

Fiedor, P. (2015). Mutual information-based hierarchies on warsaw stock exchange. *Acta Physica Polonica, A.*, 127.

Fortunato, S. (2010). Community detection in graphs. *Physics reports*, 486(3-5):75–174.

Fraley, C. and Raftery, A. E. (1998). How many clusters? which clustering method? answers via model-based cluster analysis. *The computer journal*, 41(8):578–588.

Galaskiewicz, J. and Wasserman, S. (1993). Social network analysis: Concepts, methodology, and directions for the 1990s. *Sociological Methods & Research*, 22(1):3–22.

Gan, S. L. and Djauhari, M. A. (2015). New york stock exchange performance: evidence from the forest of multidimensional minimum spanning trees. *Journal of Statistical Mechanics: Theory and Experiment*, 2015(12):P12005.

Garas, A. and Argyrakis, P. (2007). Correlation study of the athens stock exchange. *Physica A: Statistical Mechanics and its Applications*, 380:399–410.

Garey, M. R. and Johnson, D. S. (1979). Computers and intractability. w. h.

Giada, L. and Marsili, M. (2001). Data clustering and noise undressing of correlation matrices. *Physical Review E*, 63(6):061101.

Giada, L. and Marsili, M. (2002). Algorithms of maximum likelihood data clustering with applications. *Physica A: Statistical Mechanics and its Applications*, 315(3-4):650–664.

Goh, Y. K., Hasim, H. M., and Antonopoulos, C. G. (2018). Inference of financial networks using the normalised mutual information rate. *PloS one*, 13(2).

Gray, R. and Kieffer, J. (1980). Mutual information rate, distortion, and quantization in metric spaces. *IEEE Transactions on Information Theory*, 26(4):412–422.

Guo, X., Zhang, H., Jiang, F., and Tian, T. (2018a). Development of stock correlation network models using maximum likelihood method and stock big data. In *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 455–461. IEEE.

Guo, X., Zhang, H., and Tian, T. (2018b). Development of stock correlation networks using mutual information and financial big data. *PloS one*, 13(4):e0195941.

Hagberg, A., Swart, P., and S Chult, D. (2008). Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

Han, D. et al. (2019). Network analysis of the chinese stock market during the turbulence of 2015–2016 using log-returns, volumes and mutual information. *Physica A: Statistical Mechanics and its Applications*, 523:1091–1109.

Heimo, T., Saramäki, J., Onnela, J.-P., and Kaski, K. (2007). Spectral and network methods in the analysis of correlation matrices of stock returns. *Physica A: Statistical Mechanics and its Applications*, 383(1):147–151.

Holmes, A., Illowsky, B., and Dean, S. (2017). *Introductory business statistics*. Rice University.

Hosseini, S. S. and Wormald, N. (2021). Semi-labelled binary tree optimisation subject to non-negativity. *Network*. Submitted.

Hosseini, S. S., Wormald, N., and Tian, T. (2020). On finding the optimal tree of a complete weighted graph. In *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*, pages 271–275. IEEE.

Hosseini, S. S., Wormald, N., and Tian, T. (2021a). Optimal tree of a complete weighted graph. To appear.

Hosseini, S. S., Wormald, N., and Tian, T. (2021b). A weight-based information filtration algorithm for stock-correlation networks. *Physica A: Statistical Mechanics and its Applications*, 563:125489.

Hsieh, D. A. (1991). Chaos and nonlinear dynamics: application to financial markets. *The journal of finance*, 46(5):1839–1877.

Huang, W.-Q., Zhuang, X.-T., and Yao, S. (2009). A network analysis of the chinese stock market. *Physica A: Statistical Mechanics and its Applications*, 388(14):2956–2964.

Junior, L. S. and Franca, I. D. P. (2012). Correlation of financial markets in times of crisis. *Physica A: Statistical Mechanics and its Applications*, 391(1-2):187–208.

Kaya, H. (2013). Eccentricity in asset management. *Available at SSRN 2350429*.

Kenett, D. Y., Huang, X., Vodenska, I., Havlin, S., and Stanley, H. E. (2015). Partial correlation analysis: Applications for financial markets. *Quantitative Finance*, 15(4):569–578.

Kenett, D. Y., Tumminello, M., Madi, A., Gur-Gershgoren, G., Mantegna, R. N., and Ben-Jacob, E. (2010). Dominating clasp of the financial sector revealed by partial correlation analysis of the stock market. *PloS one*, 5(12):e15032.

Kocheturov, A., Batsyn, M., and Pardalos, P. M. (2014). Dynamics of cluster structures in a financial market network. *Physica A: Statistical Mechanics and its Applications*, 413:523–533.

Kraskov, A., Stögbauer, H., Andrzejak, R. G., and Grassberger, P. (2005). Hierarchical clustering using mutual information. *EPL (Europhysics Letters)*, 70(2):278.

Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50.

Krzakala, F., Moore, C., Mossel, E., Neeman, J., Sly, A., Zdeborová, L., and Zhang, P. (2013). Spectral redemption in clustering sparse networks. *Proceedings of the National Academy of Sciences*, 110(52):20935–20940.

Kumar, S. and Deo, N. (2013). Analyzing crisis in global financial indices. In *Econophysics of Systemic Risk and Network Dynamics*, pages 261–275. Springer.

Kvalseth, T. O. (1987). Entropy and correlation: Some comments. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(3):517–519.

Le, C. M., Levina, E., and Vershynin, R. (2015). Sparse random graphs: regularization and concentration of the laplacian. *arXiv preprint arXiv:1502.03049*.

Lee, J. W. and Nobi, A. (2018). State and network structures of stock markets around the global financial crisis. *Computational Economics*, 51(2):195–210.

Li, M., Badger, J. H., Chen, X., Kwong, S., Kearney, P., and Zhang, H. (2001). An information-based sequence distance and its application to whole mitochondrial genome phylogeny. *Bioinformatics*, 17(2):149–154.

Lipschutz, S. and Lipson, M. (2001). *Schaum's outline of theory and problems of linear algebra*. Erlangga.

Lloyd, S. (1982). Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137.

Lourenço, H. R., Martin, O. C., and Stützle, T. (2019). Iterated local search: Framework and applications. In *Handbook of metaheuristics*, pages 129–168. Springer.

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

Majapa, M. and Gossel, S. J. (2016). Topology of the south african stock market network across the 2008 financial crisis. *Physica A: Statistical Mechanics and its Applications*, 445:35–47.

Mantegna, R. and Stanley, H. (2000). *An Introduction to Econophysics: Correlations and Complexity in Finance*, volume 53.

Mantegna, R. N. (1999). Hierarchical structure in financial markets. *The European Physical Journal B-Condensed Matter and Complex Systems*, 11(1):193–197.

Markowitz, H. M. (1952). Portfolio selection. *Journal of Finance*, 7:77–91.

Marsili, M. (2002). Dissecting financial markets: sectors and states. *Quantitative Finance*, 2:297–302.

Marti, G., Nielsen, F., Bińkowski, M., and Donnat, P. (2017). A review of two decades of correlations, hierarchies, networks and clustering in financial markets. *arXiv preprint arXiv:1703.00485*.

McMillan, D. G. (2001). Nonlinear predictability of stock market returns: Evidence from nonparametric and threshold models. *International Review of Economics & Finance*, 10(4):353–368.

Mihaescu, R. and Pachter, L. (2008). Combinatorics of least-squares trees. *Proceedings of the National Academy of Sciences*, 105(36):13206–13211.

Miller, G. L. (1987). An additivity theorem for the genus of a graph. *Journal of Combinatorial Theory, Series B*, 43(1):25–47.

Musmeci, N., Aste, T., and Di Matteo, T. (2015). Relation between financial market structure and the real economy: comparison between clustering methods. *PloS one*, 10(3):e0116201.

Musmeci, N., Aste, T., and Matteo, T. (2014). Clustering and hierarchy of financial markets data: advantages of the dbht. *arXiv*.

Namaki, A., Shirazi, A., Raei, R., and Jafari, G. (2011). Network analysis of a financial market based on genuine correlation and threshold method. *Physica A: Statistical Mechanics and its Applications*, 390(21-22):3835–3841.

Narasimhan, G. and Smid, M. (2007). *Geometric spanner networks*. Cambridge University Press.

Newman, M. E. (2006). Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582.

Newman, M. E., Watts, D. J., and Strogatz, S. H. (2002). Random graph models of social networks. *Proceedings of the National Academy of Sciences*, 99(suppl 1):2566–2572.

Nishizeki, T. and Chiba, N. (1988). *Planar graphs: Theory and algorithms*. Elsevier.

Nobi, A., Maeng, S. E., Ha, G. G., and Lee, J. W. (2014). Effects of global financial crisis on network structure in a local stock market. *Physica A: Statistical Mechanics and its Applications*, 407:135–143.

Nobi, A., Maeng, S. E., Ha, G. G., and Lee, J. W. (2015). Structural changes in the minimal spanning tree and the hierarchical network in the korean stock market around the global financial crisis. *Journal of the Korean Physical Society*, 66(8):1153–1159.

Oh, K. J. and Kim, K.-j. (2002). Analyzing stock market tick data using piecewise nonlinear model. *Expert Systems with Applications*, 22(3):249–255.

Onnela, J.-P., Chakraborti, A., Kaski, K., and Kertesz, J. (2003a). Dynamic asset trees and black

monday. *Physica A: Statistical Mechanics and its Applications*, 324(1-2):247–252.

Onnela, J.-P., Chakraborti, A., Kaski, K., Kertesz, J., and Kanto, A. (2003b). Asset trees and asset graphs in financial markets. *Physica Scripta*, 2003(T106):48.

Onnela, J.-P., Chakraborti, A., Kaski, K., Kertesz, J., and Kanto, A. (2003c). Dynamics of market correlations: Taxonomy and portfolio analysis. *Physical Review E*, 68(5):056110.

Onnela, J.-P., Chakraborti, A., Kaski, K., and Kertiész, J. (2002). Dynamic asset trees and portfolio analysis. *The European Physical Journal B-Condensed Matter and Complex Systems*, 30(3):285–288.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2007). *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, USA, 3 edition.

Prim, R. C. (1957). Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401.

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

Redner, S. (1998). How popular is your paper? an empirical study of the citation distribution. *The European Physical Journal B-Condensed Matter and Complex Systems*, 4(2):131–134.

Rubinstein, M. (2002). Markowitz's" portfolio selection": A fifty-year retrospective. *The Journal of finance*, 57(3):1041–1045.

Rzhetsky, A. and Nei, M. (1992). Statistical properties of the ordinary least-squares, generalized least-squares, and minimum-evolution methods of phylogenetic inference. *Journal of molecular evolution*, 35(4):367–375.

Rzhetsky, A. and Nei, M. (1993). Theoretical foundation of the minimum-evolution method of phylogenetic inference. *Molecular biology and evolution*, 10(5):1073–1095.

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

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

Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905.

Small, H. (1973). Co-citation in the scientific literature: A new measure of the relationship between two documents. *Journal of the American Society for information Science*, 24(4):265–269.

Sneath, P. H., Sokal, R. R., et al. (1973). *Numerical taxonomy. The principles and practice of numerical classification.*

Song, D.-M., Tumminello, M., Zhou, W.-X., and Mantegna, R. N. (2011a). Evolution of worldwide stock markets, correlation structure, and correlation-based graphs. *Physical Review E*, 84(2):026108.

Song, W.-M., Di Matteo, T., and Aste, T. (2011b). Nested hierarchies in planar graphs. *Discrete Applied Mathematics*, 159(17):2135–2146.

Song, W.-M., Di Matteo, T., and Aste, T. (2012). Hierarchical information clustering by means of topologically embedded graphs. *PloS one*, 7(3):e31929.

Soramäki, K., Bech, M. L., Arnold, J., Glass, R. J., and Beyeler, W. E. (2007). The topology of interbank payment flows. *Physica A: Statistical Mechanics and its Applications*, 379(1):317–333.

Spielman, D. A. and Teng, S.-H. (2004). Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90. ACM.

Szabó, G., Alava, M., and Kertész, J. (2003). Structural transitions in scale-free networks. *Physical Review E*, 67(5):056102.

Tabak, B. M., Serra, T. R., and Cajueiro, D. O. (2010). Topological properties of stock market networks: The case of brazil. *Physica A: Statistical Mechanics and its Applications*, 389(16):3240–3249.

Tsankov, P. (2021). Overview of network-based methods for analyzing financial markets.

Tumminello, M., Aste, T., Di Matteo, T., and Mantegna, R. N. (2005). A tool for filtering information in complex systems. *Proceedings of the National Academy of Sciences*, 102(30):10421–10426.

Tumminello, M., Coronnello, C., Lillo, F., Micciche, S., and Mantegna, R. N. (2007a). Spanning trees and bootstrap reliability estimation in correlation-based networks. *International Journal of Bifurcation and Chaos*, 17(07):2319–2329.

Tumminello, M., Di Matteo, T., Aste, T., and Mantegna, R. N. (2007b). Correlation based networks of equity returns sampled at different time horizons. *The European Physical Journal B*, 55(2):209–217.

Tumminello, M., Lillo, F., and Mantegna, R. N. (2010). Correlation, hierarchies, and networks in financial markets. *Journal of economic behavior & organization*, 75(1):40–58.

Von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416.

Wang, G.-J. and Xie, C. (2015). Correlation structure and dynamics of international real estate securities markets: A network perspective. *Physica A: Statistical Mechanics and its Applications*, 424:176–193.

Wang, G.-J., Xie, C., and Chen, S. (2017). Multiscale correlation networks analysis of the us stock market: a wavelet analysis. *Journal of Economic Interaction and Coordination*, 12(3):561–594.

Wang, G.-J., Xie, C., and Stanley, H. E. (2018). Correlation structure and evolution of world stock markets: Evidence from pearson and partial correlation-based networks. *Computational Economics*, 51(3):607–635.

Warne, D. (2013). On the effect of topology on cellular automata rule spaces.

Wasserman, S. and Faust, K. (1994). *Social network analysis: Methods and applications*, volume 8. Cambridge university press.

Watts, D. J., Dodds, P. S., and Newman, M. E. (2002). Identity and search in social networks. *science*, 296(5571):1302–1305.

Wen, F., Yang, X., and Zhou, W.-X. (2019). Tail dependence networks of global stock markets. *International Journal of Finance & Economics*, 24(1):558–567.

Wood, D. R. (2007). On the maximum number of cliques in a graph. *Graphs and Combinatorics*, 23(3):337–352.

Yang, I., Jeong, H., Kahng, B., and Barabási, A.-L. (2003). Emerging behavior in electronic bidding. *Physical Review E*, 68(1):016102.

You, T., Fiedor, P., and Hołda, A. (2015). Network analysis of the shanghai stock exchange based on partial mutual information. *Journal of Risk and Financial Management*, 8(2):266–284.

Zhang, J., Chen, Y., and Zhai, D. (2010). Network analysis of shanghai sector in chinese stock market based on partial correlation. In *2010 2nd IEEE International Conference on Information Management*

and Engineering, pages 321–324. IEEE.

Zhang, Y., Lee, G. H. T., Wong, J. C., Kok, J. L., Prusty, M., and Cheong, S. A. (2011). Will the us economy recover in 2010? a minimal spanning tree study. *Physica A: Statistical Mechanics and its Applications*, 390(11):2020–2050.