Signed Visibility Graphs of Time Series and their application to Brain Networks

by

Gunjan Soni

B.Tech., Mody University of Science and Technology, India, 2016

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The College of Graduate Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Okanagan)

April 2019

© Gunjan Soni 2019
The following individuals certify that they have read, and recommend to the College of Graduate Studies for acceptance, a thesis/dissertation entitled:

**Signed Visibility Graphs of Time Series and their application to Brain Networks**

submitted by Gunjan Soni in partial fulfilment of the requirements of the degree of Master of Science.

Dr. Yong Gao, Irving K. Barber School of Arts & Sciences
**Supervisor**

Dr. Heinz Bauschke, Irving K. Barber School of Arts & Sciences
**Supervisory Committee Member**

Dr. Bowen Hui, Irving K. Barber School of Arts & Sciences
**Supervisory Committee Member**

Dr. Ying Zhu, Faculty of Management
**University Examiner**
Abstract

Time series have been extensively studied and used in many fields to describe time-dependent observations such as rainfall levels, stock market, and even our brain signals. In this thesis, we extend previous network based approach to the analysis of time series, by introducing signs to the edges that describe the visibility graph. The resulting signed visibility graph contains more information about the properties of time series than standard visibility graphs.

We designed a dynamic programming algorithm that creates optimal, non-overlapping and time-respecting clusters/partitions in signed visibility graphs. Our method differs from existing approaches of standard network partitioning or community detection methods that do not take into consideration the time-order in a visibility graph. Our signed visibility graph approach can also be used to define mutual information based measure for comparing the correlation between two time series. By taking into consideration the information of a node due to both positive and negative edges in the graphs, our measure is more accurate than the existing correlation measure.

As an application, we applied our framework of signed visibility graph to brain networks and studied the clustering of brain time series for various sets of patients suffering from ADHD, Schizophrenia, Bipolar Disorder, and Epilepsy (with & without seizures), compared with normal human brain signals. For fMRI data, these comparisons are done on various levels: starting from individual Regions of Interest (ROIs) of the brain, to groups of ROIs, and lastly to Resting State Networks (RSNs; brain sub-networks/regions). These comparisons gave some interesting results showing differences in various metrics of brain network. For instance, differences were found in the average number of clusters, cluster size, average mutual information measure, average jaccard index values of clusters, with extreme values in certain sub-networks and in certain disease-sets, etc.
Lay Summary

Time series is a sequence of values that change with respect to time. Graphs are represented in the form of points and lines, and a line is used to connect two points. In order to study the characteristics of time series, they are converted to data structures called visibility graphs in which points are the time-series values and lines are drawn based on the type of visibility graph used. We introduced signs to the lines depending on the location of two points with respect to each other. We also designed an algorithm to group points in a visibility graph, based on their positive and negative lines, making sure that points within a group are continuous and groups follow time-constraint. We also created a formula to measure relation between two visibility graphs. We applied these concepts on the analysis of brain-network data (fMRI and EEG).
Preface

The work in this thesis has been accepted for a poster presentation at the International School and Conference on Network Science (NetSci) 2019, Burlington, Vermont, USA.

Two copyrighted figures have been reprinted in this thesis with permission of [25] and [46] respectively.
# Table of Contents

Abstract ................................................... iii
Lay Summary ................................................. iv
Preface .......................................................... v
Table of Contents ............................................. vi
List of Tables ................................................ viii
List of Figures ............................................... ix
List of Programs .............................................. x
Acknowledgements ............................................ xi
Dedication ...................................................... xii

1 Introduction ................................................ 1

2 Preliminaries ............................................... 4
   2.1 Graphs & Networks ....................................... 4
   2.2 Time Series .............................................. 5
   2.3 Visibility Graphs ......................................... 6
   2.4 Signed Networks ......................................... 8
   2.5 Clustering ............................................... 8
   2.6 Brain Networks .......................................... 10
   2.7 fMRI ..................................................... 11
   2.8 EEG ...................................................... 12

3 Signed Visibility Graphs ................................. 13
   3.1 Definition and motivation ............................... 13
   3.2 Algorithms for creating visibility graphs ............ 15
# Table of Contents

3.2.1 Basic Algorithm ........................................... 15
3.2.2 Horizontal Signed Visibility Graph Algorithm .......... 17
3.2.3 Divide and Conquer Algorithm ............................ 20

4 Partitioning Algorithm ......................................... 23
  4.1 Time-respecting Clustering .................................. 23
  4.2 Algorithm for Time-respecting Clustering ...................... 24
    4.2.1 Dynamic Programming .................................. 24
    4.2.2 The Algorithm ........................................... 25
  4.3 Comparison with Louvain Algorithm .......................... 27
    4.3.1 Louvain Algorithm ...................................... 27
    4.3.2 Methods and Results of Comparison ....................... 29
  4.4 Importance of Signed Edges and Time-respecting Constraint 33

5 Multivariate Correlation ....................................... 36
  5.1 Multivariate Time Series .................................... 36
  5.2 Interlayer Mutual Information ............................... 37

6 Application to Brain Network Analysis ........................ 44
  6.1 Terminologies .............................................. 44
    6.1.1 Data and Preprocessing Techniques ....................... 44
    6.1.2 Neurological Disorders .................................. 47
  6.2 Analysis of fMRI Data ....................................... 48
    6.2.1 fMRI data .............................................. 48
    6.2.2 Results .................................................. 49
  6.3 Analysis of EEG Signals ..................................... 58
    6.3.1 EEG Data ............................................... 58
    6.3.2 Results .................................................. 59

7 Conclusion ..................................................... 62

Bibliography ...................................................... 63

Appendix

A Implementation Programs ....................................... 68
# List of Tables

5.1 Comparative results of multivariate correlation equations . . 40  
5.2 Comparative results of values by $MI$ equations . . . . . . . . . 43
List of Figures

3.1 Signed Visibility Graph ........................................ 14
3.2 Signed Natural Visibility Graph example .................. 16
3.3 Signed Horizontal Visibility Graph example ............... 19
3.4 Signed Visibility Graph by Divide and Conquer algorithm . 20
4.1 Example of Partitioning ........................................ 28
4.2 Sample fMRI time series ........................................ 29
4.3 Comparison between Louvain algorithm and our algorithm .. 30
4.4 Louvain Algorithm clustering examples ...................... 31
4.5 Weak clusters by Louvain algorithm .......................... 32
4.6 4 cases of clustering ............................................ 34
5.1 Multiplex visibility graph from multivariate time series .... 37
5.2 Mutual Information values scatter plots ..................... 39
5.3 Mutual Information between 4 time series .................... 42
6.1 Yeo’s parcellation ............................................... 45
6.2 ROI level plots of avg. number of clusters .................. 49
6.3 Groups of ROIs level plots of avg. number of clusters ...... 50
6.4 Comparison with Louvain algorithm (groups of ROIs) ....... 52
6.5 Groups of ROIs level histogram of Jaccard Index values .... 53
6.6 Groups of ROIs level violin plots of MI values .............. 54
6.7 Groups of ROIs level violin plots of MI values (unsigned) .. 56
6.8 RSN level pirate plots of number of clusters ................. 57
6.9 RSN level violin plots of mutual information values ......... 58
6.10 EEG bar-scatter plot of number of clusters ................. 59
6.11 EEG box-scatter plot of Jaccard Index ....................... 60
6.12 EEG box plot of average cluster-size ......................... 61
List of Programs

A.1 Program for Natural Signed Visibility Graphs . . . . . . . . . . . . 68
A.2 Program for Horizontal Signed Visibility Graphs . . . . . . . . . . 70
A.3 Program for Divide and Conquer algorithm . . . . . . . . . . . . 71
A.4 Program for Partitioning algorithm . . . . . . . . . . . . . . . . 73
Acknowledgements

First of all, I would like to thank my supervisor, Dr. Yong Gao for his esteemed guidance and support from the very beginning of my graduate studies. His immense knowledge, motivation, enthusiasm and patience has been extremely helpful all the time in the research and writing of this thesis. I am deeply grateful to have him as the advisor and mentor for my masters study.

I am thankful to Dr. Daniele Marinazzo, Ghent University, for being so responsive and helping me learn the aspects of neurological data-processing.

I would also like to thank every faculty member in the university who have taught or guided me in any way. The knowledge given by them has been reflected in this thesis in one way or another.

I am also grateful to my fellow graduate students who I have interacted with, for any help and contribution in my masters program.
Dedication

I would like to dedicate this thesis to my parents Dr. Rekha Soni and Dr. Suresh Chand Soni, and my fiancé Krishna Gupta. Without their support, I wouldn’t have been able to pursue my dreams and interests. They have been the wind beneath my wings and I can never thank them enough.

I would also like to thank my friends in Kelowna and India for being there whenever I needed them.
Chapter 1

Introduction

In our interconnected world, almost anything within or around us, or any technology that we interact with involves a network in one way or another. For instance, there are basic networks such as electricity and roads, social networks, communication networks such as telephones and advertising, organizational networks such as the government, the internet, biological networks such as our blood vessels, and even our brains [31]. The history of the study of networks goes back to the 18th century when they were used for global trade and commercial purposes. Since then there has been an exponential growth in the discovery and invention of different types of networks. The field of network science combines ideas from mathematics, physics, biology, computer science, statistics, social sciences, and many other areas using more complex networks in real world.

In addition to networks, time series are also frequently used to model phenomena where properties change over time. A time series is a sequence of data values taken at successive equally spaced points in time; it is a sequence of discrete-time data, for example, rainfall levels, stock market, and our brain signals. In order to analyze and extract meaningful statistics and other characteristics of time series data, certain methods of analysis are applied. One of the recent developments of new methods in exploitation and analysis of the time series is visibility graphs [26, 28] by which a time series is characterized as and mapped to a network. This approach helps to use the tools of network science for describing the structure and underlying dynamics of time series.

The task of grouping a set of nodes or vertices in such a way that nodes in the same group or cluster are more similar to each other (according to some node metric or feature) than to those in other groups, is termed as clustering or community detection. This problem is computationally hard in general and is extensively used in many fields, including machine learning, pattern recognition, image analysis, bioinformatics, etc. Cluster analysis can be achieved by various algorithms that differ significantly in their understanding of what constitutes a cluster and how to efficiently find them. Some such popular algorithms are Newman’s Network Modularity algorithm [30] that
defines modularity based on edge-density, taking a difference of fraction of edges within a partition and fraction of edges incident to a partition; Girvan-Newman’s method [12] that involves iterative removal of nodes with high vertex-betweenness value to form communities; Neighbour-Joining method [40] which follows a bottom-up approach and creates clusters between nodes with least distance; and Louvain algorithm [5] that will be discussed later. All these algorithms can be used for partitioning visibility graphs but they do not take into account the time-dependency of each node in the network, and therefore do not provide with a solution specifically for visibility graphs.

Based on graph theory, there have been many recent developments in the analysis of complex networks quantitatively and these have been rapidly translated to studies of both anatomical and functional brain network organization. Human brain networks, derived from MRI and electrophysiological data, follow complex network characteristics like small-worldness, have high-degree cortical ‘hubs’ and modular & hierarchical properties [7]. The existence of these hubs have resulted in the division of the whole brain network into several sub-networks that are spatially non-overlapping and functionally distinct. Data from the brain, taken in the form of magnetic imaging or electrical signals, are in the form of time series, and therefore, is used to apply the aforementioned notions to gain some fruitful results.

The main contribution of this thesis is the proposal of signed visibility graphs that generalize the notion of a visibility graph. The concept of a signed visibility graph is also used to remodel the mutual-information measure for multiplex networks [25]. With the signed-visibility-graph-based measure, we observe differences in the obtained value compared with original measure value, due to the consideration of negative edges. We designed and implemented a dynamic programming algorithm for partitioning a signed visibility graph into clusters that respect the temporal constraint imposed by the time series. The implementations and algorithm are then applied to data taken from brain signals (fMRI and EEG) of healthy persons and patients suffering from ADHD, Schizophrenia, Bipolar Disorder and Epilepsy. The results gathered from each patient-sets are then compared with the results of healthy persons as well as with other patient-sets to generate some interesting observations.

In Chapter 2, we give an overview of the preliminary terminologies required for understanding the ideas discussed in the thesis. In Chapter 3, we discuss the concept of signed visibility graph along with an explanation of the methods involved in its creation and their properties. In Chapter 4, we discuss the visibility graphs partitioning algorithm and its applicability. Chapter 5 sheds light on an application of the signed visibility graphs: mu-
tual information in multivariate visibility graphs along with a comparative discussion of the difference in the correlation value of signed and unsigned visibility graphs. In Chapter 6, we put forward all the results of the research and apply the partitioning algorithm and mutual information on brain signals data with a detailed discussion of differences in cluster numbers, cluster sizes, mutual correlation measure, and jaccard index values between the clusters of healthy persons and patients with considered mental and neurological disorders. In Chapter 7, we summarize our results and mention the future direction.
Chapter 2

Preliminaries

In this chapter, we explain some background material and notions for understanding the discussions of the chapters to follow.

2.1 Graphs & Networks

A network represents mutual interactions/connections among different objects/members. These objects are called vertices/nodes while the interactions are indicated by edges/links. A well-known example is Zachary’s Karate Club, as mentioned in [47], which is a social network with members of the club represented by nodes and the links between them representing their friendship or interactions.

Graphs are mathematical abstractions of networks. A graph $G = (V, E)$ is a structure with a set of vertices $V$ and connecting those vertices is a set of edges $E$ [10, 11]. An edge $e \in E$ between two nodes $u, v \in V$ can be written as $e = (u, v)$. An edge can have a number of attributes that define the relation between the nodes it is linking, like sign, direction, weight etc. For example, a positive edge between two nodes in a graph used to model social network may indicate friendly relations between the two members, and a negative edge may indicate enmity. Depending on the type of network, there may be multiple interpretations of the same attribute. Edges can have weights too, which signify either a parameter or the degree of link between the two nodes an edge is connecting, depending upon their usage. For example, in a highway network, the weight of an edge may represent distance, cost to travel that distance or the travel time between two cities. A well-known problem is to minimize either the cost or the time, or to find shortest path between two given cities.

Based on the attribute of direction, we have two types of graphs: undirected and directed. An undirected graph is simply defined by $G = (V, E)$, where the edges in $E$ are represented as $\{u, v\}$, have no direction, form an unordered set, and represent symmetric relationships. In a directed graph each edge $(u, v) \in E$ has an added direction attribute with $u$ being the source vertex, $v$ being the target vertex and each edge represents an asymmetric
2.2 Time Series

A time series is a series of data points that are listed or indexed according to time order. Basically, any value that can change with time would form a time series. The analysis of time-series applies to many fields such as economics (unemployment, consumer price index, population, gross national product, and production), natural sciences (water level index of a lake, air temperature, size of natural population, etc.), physical sciences and mathematics (designs for airplanes and rockets, improvement of radar and other electronic devices, and investigation of certain production and chemical processes) [16], and biological sciences (blood sugar levels, growth of foetus, and brain signals).

There are two types of time series: discrete and continuous. Discrete time series values are recorded or observed at fixed time intervals, for example, monthly data of lake water levels. Continuous time series are observed continuously within a specific time interval, for example, recording of EEG
values for a span of 60 seconds [6].

Based on the number of variables involved, time series are of two types: univariate, and multivariate. A univariate time series has just one variable, that has a changing value with respect to time, for example, the stock value of a company. A multivariate time series has more than one variables that are represented in the same series, where each variable has a particular value at each time point. Multivariate time series can be seen as a combination of more than one univariate time series. For example, changing weather conditions (involving dew, temperature, pressure, pollution, rain, wind speed, etc.) for a particular location [22].

2.3 Visibility Graphs

Visibility graphs have been largely used to check “visibility” in polygons: the visibility graph of a simple polygon is a graph whose vertices correspond to the vertices of the polygon and two vertices have an edge between them if they can “see” each other [14], according to some criteria. The polygon visibility graphs have been used in solving a number of real-world problems such as architecture and urban planning (art-gallery problem). Visibility graphs can also be defined on time series in which the nodes are time points, and two nodes are linked if they have a visibility between each other according to the type of visibility graph used. Multilayer networks are networks that incorporate multiple types of interactions between the same nodes. Univariate time series are mapped to a single visibility graph, but for mapping multivariate time series, multilayer visibility graphs are used, which will be discussed more in Chapter 5.

There are different types of visibility graphs and the following is a brief description of a few of them.

Natural Visibility Graph

Given a time series of $N$ data, we take any two time points $i$ and $j$ for which the measured quantity values are $y_i$ and $y_j$, respectively. These two points are said to be visible to each other and will become two connected nodes in the associated Natural Visibility Graph (NVG) if any other data point $y_k$ placed between them fulfills [26]:

$$y_k < y_i + (y_j - y_i) \frac{k - i}{j - i}.$$  \hspace{1cm} (2.1)
A natural visibility graph is defined on time points if they have visibility according to the above condition. The visibility graph associated with a time series is always connected (as each node has a visibility with at least its nearest neighbors), undirected, and invariant (under several transformation of the time series either by rescaling of the axes or under horizontal and vertical translations).

**Horizontal Visibility Graph**

Two time points \(i\) and \(j\), in which the measured quantity takes values \(y_i\) and \(y_j\), respectively, will have Horizontal Visibility if any other data point \(y_k\) placed between them is smaller [28]:

\[
y_k < \min\{y_i, y_j\}, \forall i < k < j
\]  

(2.2)

In the horizontal visibility graphs (HVGs), if between \(y_i\) and \(y_j\) a horizontal line is drawn and it does not intersect with any intermediate data height, then the nodes \(i\) and \(j\) are connected. The resulting HVG is simply a subgraph of its corresponding NVG.

**Divide-and-Conquer Visibility algorithm**

A visibility graph can be constructed by checking inequality (2.1) for each pair of time points. A more efficient algorithm is based on the idea of divide and conquer technique [27]. Within each time series \(y_1, \ldots, y_n\), a node \(k\) with maximum value is found which divides the time series into two separate parts - left and right. In each step, the connections of \(k\) with other nodes are created based on its “visibility” with other nodes and there is a recursion of these steps until no further division can happen. Following the above two steps, all nodes in the time series are considered, and the time complexity of visibility graph creation is highly reduced.

**Sort-and-Conquer Visibility algorithm**

A more recent iterative visibility algorithm is proposed in [17] which consists of two stages. In the first stage, data elements are sorted in a list in descending order according to their \(y\) values. In each iteration, one node \(k\) taken in order from the list is considered and is joined with other nodes based on an interval. This interval includes nodes that are positioned between the nodes \(k\) is already joined with (due to previous iterations; null for first iteration). The visibility between \(k\) and elements of the interval is then checked, edges are drawn if it exists, and network is updated accordingly.
2.4 Signed Networks

The concept of signed networks, or graphs with positive and negative edges, has been popular in the field of social networks, with the ties of friends denoting positive edges and ties of enemies denoting negative edges. The same concept can be applied to other networks too with different denotations, while the fundamental idea remains the same - adding signs to edges. In these networks, the term switching-a-vertex means negating the signs of all the edges incident to that vertex and switching-an-edge means negating the sign of that edge.

There are various notions to define balance of a signed graph, and some of them are quite contrasting. The definition of “strong” balance says that $G$ is a balanced network if, and only if, there are even number of negative edges in every cycle of $G$. The “weak” definition of balance, by contrast, says that a network is balanced if, and only if, there is no cycle that has only one negative edge [23]. Another definition given by Harary [20] says that any network that is clusterable, displays perfect strong balance. Which means the network’s nodes can be divided into some number of non-overlapping sets such that all edges within the sets are positive and all edges between the sets are negative. Thus, a network possessing insularity or cliquishness can be characterized as a strongly balanced network. We would work on the same principle of dividing a signed network into clusters to form a weakly balanced visibility network in Chapter 4.

2.5 Clustering

Community detection, or clustering, is the term used for organizing vertices to clusters, with many edges joining vertices of the same cluster and comparatively few edges joining vertices of different clusters. Such clusters, or communities, can be considered as fairly independent entities of a graph, being functionally different from each other. Detecting communities is of great importance in sociology, biology and computer science, disciplines where systems are often represented as graphs [15]. There exist many forms of clustering in our day to day life - clustering of players to form teams, clustering in protein-protein interactions, clustering of hyperlinks in webpages, etc.

Clustering is a partition of the vertex-set $V$ into subsets $C_1, C_2, \ldots C_n$, where $n$ is the number of clusters in $V$, such that each positive edge joins two nodes in the same subset and each negative edge joins two points from
2.5. Clustering

different subsets. Since real-world examples do not showcase this strong partitioning, there is a slight modification to this definition stating, the graph is divided into subsets such that most of the edges within a subset are positively signed and most of the edges between two subsets are negatively signed. In our work, we have used this modified or weakly balanced definition of clustering.

Let us start with a subset \( C \) of a graph \( G \). The following details are mentioned in [15] but we have modified it a little to suit the explanation for our work. We define 4 parameters to indicate the internal and external degrees of vertex \( v \in C \), \( k_{\text{int}}^v \), the number of positive edges \( v \) has internal to the cluster \( C \), \( k_{\text{ext}}^v \) the number of positive edges \( v \) has external to the cluster \( C \) or with other clusters, \( k_{\text{int}}^v \) the number of negative edges \( v \) has internal to the cluster \( C \), and \( k_{\text{ext}}^v \) the number of negative edges \( v \) has external to the cluster \( C \). Now, for \( v \) to be a part of the cluster \( C \), it should fulfill the following:

1. \( k_{\text{int}}^v > k_{\text{ext}}^v \), i.e., the number of positive edges of \( v \) should be greater within the cluster

2. \( k_{\text{ext}}^v > k_{\text{int}}^v \), i.e., the number of negative edges of \( v \) should be greater outside the cluster

Now considering the cluster \( C \) as a whole, we define a few more terms, \( \text{intra-cluster positive density} \), \( \delta_{\text{int}}(C) \) which is the ratio between the number of internal positive edges of \( C \) and the number of all possible internal edges, and \( \text{intra-cluster negative density} \), \( \delta_{\text{int}}(C) \) which is the ratio between the number of internal negative edges of \( C \) and the number of all possible internal edges, i.e.

\[
\delta_{\text{int}}(C) = \frac{\#\ \text{positive internal edges of } C}{n_c(n_c-1)/2}
\]  

\( \delta_{\text{int}}(C) = \frac{\#\ \text{negative internal edges of } C}{n_c(n_c-1)/2} \)  

Similarly, \( \text{inter-cluster positive density} \), \( \delta_{\text{ext}}(C) \) which is the ratio between the number of positive edges from the vertices of \( C \) to the rest of the graph and the maximum number of inter-cluster edges possible, and \( \text{inter-cluster negative density} \), \( \delta_{\text{ext}}(C) \) which is the ratio between the number of negative edges from the vertices of \( C \) to the rest of the graph and the maximum number of inter-cluster edges possible, i.e.

\[
\delta_{\text{ext}}(C) = \frac{\#\ \text{positive inter-cluster edges of } C}{n_c(n-n_c)}
\]  

\( \delta_{\text{ext}}(C) = \frac{\#\ \text{negative inter-cluster edges of } C}{n_c(n-n_c)} \)
2.6 Brain Networks

The average positive link density $\delta_p(G)$ is the ratio between number of positive edges of $G$ and maximum number of possible edges $n(n-1)/2$. Similarly for negative edges we have the average negative link density $\delta_n(G)$. Now we have four conditions for $C$ to be a community or cluster,

1. $\delta_{p\text{-int}}(C) > \delta_p(G)$, $\delta_{n\text{-int}}(C) < \delta_n(G)$
2. $\delta_{p\text{-ext}}(C) < \delta_p(G)$, $\delta_{n\text{-ext}}(C) > \delta_n(G)$

The goal of most clustering algorithms, implicitly or explicitly, is searching for the best tradeoff between a large $\delta_{p\text{-int}}(C)$ and a small $\delta_{p\text{-ext}}(C)$ and/or a small $\delta_{n\text{-int}}(C)$ and a large $\delta_{n\text{-ext}}(C)$. A simple way to do that is, e.g., maximizing the sum of the differences $\delta_{p\text{-int}}(C) - \delta_{p\text{-ext}}(C)$ and/or $\delta_{n\text{-ext}}(C) - \delta_{n\text{-int}}(C)$ over all clusters of the partition. More discussion on this is done in Chapter 4.

By observation, visibility graphs have been found to have a modular structure in which subnetworks are constituted by time points that are mainly adjacent. After partitioning the visibility graph there is a natural decomposition of the time series into smaller time intervals.

2.6 Brain Networks

The central organ of the human nervous system is the brain. Despite having a fixed anatomy, most of the physical and mental actions of the body, processing, integrating, etc. are done by the brain. It also acts as a coordinator of the information it receives from the sense organs, deciding all the instructions sent to the rest of the body as a response to the information or action-reaction pair. These innumerable functionings may be a result of brain’s network architecture being hierarchical and modular, which organizes local interactions and ensures adaptability, damage-resilience, efficient message passing, and diverse functionality from a fixed structure [34].

The structure of brain is represented by a network, with the nodes being neurons or brain-regions, depending upon the usage, and the edges being their communication channel or signal-interaction. Our brain is divided into various sub-regions that are organized and interconnected as communities. These brain regions are spatially separated, are functionally distinct and each of them has a relationship of integrated functioning with all others.

Studies have shown that brain networks typically have a small-world topology with similarities like dense local clustering or cliquishness, short
path lengths and relatively few long-range connections. The characteristic also explains for both segregated/specialized and distributed/integrated information processing of the brain. Moreover, brain networks have been found to be economical, tending to minimize wiring costs while supporting high dynamical complexity, similar to small-world networks [2].

Due to the well arranged and orchestrated communication between the functional connectivity networks, all the complex cognitive processes of the human brain take place. Explanation about differences in the functioning of the brains of patients with mental disorders is a topic with many approaches and we will be discussing one approach in our work of finding the functioning of the brain-regions in these patients, then comparing it with controls (humans with healthy or normal functioning brains) and finally establishing some inter-group differences. These differences will be observed on the basis of differences that occur in the study of signals like fMRI and EEG of patients and controls.

2.7 fMRI

Functional magnetic resonance imaging or functional MRI (fMRI) is a measure of brain activity that detects changes associated with cerebral blood flow based on which it measures activation of neurons. When a region of the brain is active or is getting used, there is an increased blood flow to that region. The primary form of fMRI uses the blood-oxygen-level dependent (BOLD) contrast, which is a type of specialized brain and body scan of humans or other animals used to map neural activity by showing the change in blood flow related to energy use by brain cells. When there is a task to be performed, the brain area(s) responsible for performing the task become active, the blood flow in that active area increases, thus increasing the supply of oxygen required for creating the energy required by the cells in that area for performing the task. This blood flow is then imaged to form fMRI images [39]. fMRI data is represented in the form of 2D, 3D and 4D images.

One of the types of fMRI imaging is a Resting state fMRI (r-fMRI) which is used in brain mapping to evaluate interactions within brain regions that occur in a resting or task-negative state, a state when an explicit task is not being performed. Because brain activity is intrinsic, i.e., it is present even when an externally prompted task is not performed, any brain region will have spontaneous changes in BOLD signal. The resting state fMRI approach is mainly useful in exploring the brain’s functional organization and in examining the alterations in this organization in case of neurological
2.8 EEG

Electroencephalography (EEG) is an electrophysiological monitoring method that is used for recording electrical activity of the brain, more precisely, for measuring voltage fluctuations that arise within neurons from ionic current used for message passing within brain. EEG records brain’s spontaneous electrical activity over a period of time from multiple non-invasive electrodes, which are small metal discs with thin wires, that are placed on a person’s scalp, and these signals are recorded in the form of wave patterns. The EEG values track brain wave patterns with respect to time and therefore are a time series. EEG is a popular monitoring test for observing abnormal patterns in patients that indicate seizures and other problems. It is often used to diagnose epilepsy, sleep disorders, depth of anesthesia, coma, etc. EEG is one of the oldest and still continues to be a valuable tool for research and diagnosis that offers millisecond-range temporal resolution. Apart from clinical uses, EEG is also very popular in the field of artificial intelligence in various technologies like artificial brain and neo-humanity. In our work, we have used EEG time series for gaining knowledge of brain functioning.
Chapter 3

Signed Visibility Graphs

In this chapter, we will introduce the concept of signed visibility graphs, discuss the motivation and present methods for constructing them using time-series.

3.1 Definition of Signed Visibility Graphs and its motivation

One of the ways in which signs have been introduced into graphs is through social networks wherein, each person belonging to a society is a node. If two people are friends then their nodes in the network have a positive link and if two persons are foes then they are connected through a negative link. A group of people with only positive links between them, may form a social group or a team. The signs of edges may have different meanings depending on the purpose of the network. For team-formation, positive edges may denote similar interests or complementing knowledge-areas whereas negative edges may denote dissimilar interests or conflicting knowledge-areas.

An important problem in the study of social networks is to understand the tension between the two signs, which is termed as structural balance. There are many theories of structural balance, but we would focus on one of them to define balance in our work in next chapter. Weak structural balance theory states that if a graph can be divided into groups such that nodes within each group are friends, or positively tied, and nodes between groups are enemies, or with negative ties, then the graph is said to have a weak balance [11].

Signed Visibility Graphs is a concept of the combination of signed networks with visibility graphs. We add signs (or weights) to the edges between two value points or nodes based on the visibility between them. If two nodes can “see” each other, they have an edge with positive sign or +1 weight. For adding the negative edges (or edges with -1 weight) we look for inverted visibility, i.e., those nodes that are not visible to each other in the “general” visibility graphs but will have a visibility with each other when the graph is
3.1. Definition and motivation

Figure 3.1: An example of a signed visibility graph.

inverted or made upside-down, will have a negative edge.

We first create a visibility graph of a given time series using any of the visibility algorithms - we call this graph the “general” graph - and assign a +1 weight or a positive sign to all the edges of visibility in this graph. We then invert the graph, or make it upside-down without removing the positive edges - we will call this graph the “inverted” graph. Now all the nodes that did not have a visibility in the general graph, or don’t have a positive edge between them, but can see each other in the inverted graph, will have an edge between them that will have a negative sign or a -1 weight.

In Figure 3.1, we have a sample time series and have edges to denote a visibility graph. Green edges are for positive visibility and red edges are for negative or inverted visibility. Almost all the possible edges have been drawn in the figure.

In this work, we have only worked with undirected graphs but this concept would work for directed graphs too in a similar fashion, the only difference would be that the edges will have 2 attributes: sign and direction. Now that we have explained the abstract design of signed visibility graphs, we will see how they are implemented using three types of visibility graphs: natural visibility graphs, horizontal visibility graphs, and divide-and-conquer algorithm.
3.2 Algorithms for creating visibility graphs

The methods discussed in Chapter 2 have been modified to create signed visibility graphs, and the algorithms are discussed below.

3.2.1 Basic Algorithm

The algorithm is a natural way to create a visibility graph based on the definition and Equation (2.1) given in Chapter 2. Below is the description of a customized method to create a signed visibility graph. Each node $v \in V$ has been initialized based on $y_v$, sorted by $t_v$.

---

**Algorithm 1 Natural Signed Visibility Graph**

1: Input: A time series $y[0..(n-1)]$
2: Result: The visibility graph for given time series $G(V, E_p, E_n)$
3: for each $v_a \in V$ do
4:   for each $v_b \in V \mid t_b > t_a$ do
5:     // Adjacent data points are neighbours
6:     if $t_b = t_a + 1$ then
7:       $E_p = E_p \cup \{v_a, v_b\}$;
8:     continue;
9:   end if
10:  $visible \leftarrow true$;
11:  $visible2 \leftarrow true$;
12:  for each $v_c \in V \mid t_a < t_c < t_b$ do
13:    value $\leftarrow y_b + \frac{(y_a - y_b)(t_b - t_c)}{t_b - t_a}$;
14:    if $y_c > value$ then
15:      $visible \leftarrow false$;
16:    else if $y_c < value$ then
17:      $visible2 \leftarrow false$;
18:    end if
19:    if $visible = false$ and $visible2 = false$ then
20:      break;
21:  end if
22: end for
23: if $visible2 \leftarrow true$ then
24:   if $(v_a, v_b) \notin E_p$ then
3.2. Algorithms for creating visibility graphs

25: \[ E_n = E_n \cup (v_a, v_b); \]
26: \[
\text{end if}
\]
27: \[
\text{else if visible } \leftarrow \text{ true then}
\]
28: \[ E_p = E_p \cup (v_a, v_b); \]
29: \[
\text{end if}
\]
30: \[
\text{end for}
\]
31: \[
\text{end for}
\]
32: Return \( G; \)

Figure 3.2: (a) Original time series of the water levels of river Amazon. (b) Signed natural visibility graph obtained from the time series in (a).
3.2. Algorithms for creating visibility graphs

In the above algorithm, the output is a visibility graph $G$ with $E_p$ as the positive edges and $E_n$ as the negative edges. Each node has a value $y$ (on y-axis) and a time point $t$ (on x-axis). For every pair of nodes we check for their visibility by checking the value of the nodes between them and generate a positive, negative or no edge accordingly. The python implementation of this algorithm is in the Appendix [Program A.1].

The visibility graph [Figure 3.2b] has been generated with a time series of high and low water levels of the river Amazon at Iquitos for the years 1962-1995 [21]. Positive edges are denoted by colour green and negative edges are denoted by colour red. The original time series has been shown in Figure 3.2a. The worst case time complexity of this algorithm is $O(n^3)$, where $n$ is the number of data or value points in the original time series.

3.2.2 Horizontal Signed Visibility Graph Algorithm

In this section, we will discuss the horizontal visibility graph algorithm, remodelled for creating signed visibility graphs. The algorithm mentioned below was referenced from the paper [48] that gave a faster version of constructing a horizontal VG based on the same concept as original HVG [Equation (2.2)], but with a few changes. The key concept of this “fast” horizontal VG algorithm is that a maximal value point is always visible to its neighbors maximal value point, even if the distance between those two maximal points is more than two nodes. All the maximal values belonging to the time series are stored in a sorting list $SortList$ in the algorithm, which is empty when the index is zero. The explanation is more lucid in the algorithm below.

Algorithm 2 Horizontal Signed Visibility Graph

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td><strong>Input:</strong> A time series $y[0..(n - 1)]$</td>
</tr>
<tr>
<td>2.</td>
<td><strong>Result:</strong> A visibility graph for the given time series $G(V, E_p, E_n)$</td>
</tr>
<tr>
<td>3.</td>
<td>// Creation of positive visibility graph: LEVEL 1</td>
</tr>
<tr>
<td>4.</td>
<td>$SortList$.add($y[0]$)</td>
</tr>
<tr>
<td>5.</td>
<td>for $i = 1$ to $(n - 1)$ do</td>
</tr>
<tr>
<td>6.</td>
<td>$E_p = E_p \cup (v_{y[i]}, v_{y[i-1]})$;</td>
</tr>
<tr>
<td>7.</td>
<td>if $y[i] &gt; y[i-1]$ then</td>
</tr>
<tr>
<td>8.</td>
<td>Search $Loc$ while $SortList[0..Loc] &lt; y[i]$</td>
</tr>
<tr>
<td>9.</td>
<td>if $Loc &gt; -1$ then</td>
</tr>
<tr>
<td>10.</td>
<td>for $x = 0$ to $Loc$ do</td>
</tr>
<tr>
<td>11.</td>
<td>$E_p = E_p \cup (v_{y[i]}, v_{SortList[x]})$;</td>
</tr>
<tr>
<td>12.</td>
<td>Remove $SortList[x]$ from $SortList$</td>
</tr>
</tbody>
</table>
In its Level-1, iteratively a maximal point is found and is positively connected to all the points in SortList lower in value than itself. All these lower values are then removed from SortList and the new maximal value is added to the list. In Level-2, we create a new array inv_y for storing the new value points after the old value points have been subtracted from max_y and inv_SortList acts as the sorting list for the inverted graph. Similar steps are repeated for creating negative edges. Finally, the graph G is returned containing both positive and negative edges. The Python code implementation for the algorithm is in Appendix [Program A.2] and below is the generated graph [Figure 3.3].
3.2. Algorithms for creating visibility graphs

The worst case time complexity to generate a horizontal visibility graph using Equation (2.2) is $O(n^3)$, where $n$ is the number of nodes in the time series. But Algorithm 2 follows a faster approach, and as mentioned in [48], has a time complexity of $O(n)$.

NVG vs. HVG

It is evident from figures 3.2b and 3.3 that horizontal visibility graphs are just subgraphs of natural visibility graphs. But there exist certain differences that make these two unique. Degree distributions in a graph inherit correlations. Natural visibility graphs have more connections, more edges and a higher degree distribution as compared to the horizontal visibility graphs and the former naturally allow for development of hubs, which turn out to be typically associated with extreme events in the data and can correlate with data at all scales. NVGs are in principle better suited to handle and extract long-range correlations whereas HVGs have been found to possess exponentially decreasing degree distributions and hence they are good to be used in processes involving short-range correlations. Horizontal visibility algorithms geometric criterion is more visibility restrictive than the natural VG, which implies that the nodes within horizontal visibility graph
3.2. Algorithms for creating visibility graphs

Figure 3.4: Signed visibility graph created by the Divide and Conquer algorithm obtained from the same “water level of river Amazon” time series.

will have less visibility with other nodes as compared to the nodes in NVG.

Despite the differences, in both the cases, the $n$ nodes (or time-points) are connected to each other by a trivial Hamiltonian path that originates a natural ordering in the degree sequence. Both the algorithms produce graphs that are (i) connected, (ii) undirected, and (iii) invariant under affine transformations of the series. Both the algorithms originally have the same time complexities of $O(n^3)$, and their faster versions have similar complexities as well - $O(n)$ for HVG and $O(n\log n)$ for NVG (to be discussed in next section).

3.2.3 Divide and Conquer Algorithm

In this section, we will discuss a divide and conquer method, which has been proposed in [27] for creating a visibility graph. As described in the previous chapter, the method first finds a maximum value node of the whole time series, connects it to all the nodes it can see and divides the time series into two parts: “left” and “right”. There is a recursive call on both the parts until the series cannot be divided anymore. The algorithm is described below and has been modified from its original version to best suit our work.
3.2. Algorithms for creating visibility graphs

**Algorithm 3** Divide and Conquer Algorithm

1. **Input**: A time series \( y[0..(n-1)] \)
2. **Result**: The visibility graph for given time series \( G(V, E_p, E_n) \)
3. //For the first run \( left = 0 \) and \( right = (n-1) \)
4. TimeSeries2VisibilityGraphFast\( (y, 0, (n-1)) \)

5. **function** TimeSeries2VisibilityGraphFast\( (y, left, right) \)
6. if \( left < right \) then //There must be atleast two nodes in the time series
7. \( k \leftarrow \) the index of maximum value of \( y[left...right] \)
8. TimeSeries2VisibilityGraphFast\( (y, left, k-1); // \) the left side of \( k \)
9. TimeSeries2VisibilityGraphFast\( (y, k+1, right); // \) the right side of \( k \)
10. for each \( i \) from \( left \) to \( right \) do
11. \( visible \leftarrow true; \)
12. \( visible2 \leftarrow true; \)
13. if \( i < k \) then
14. \( a \leftarrow i; b \leftarrow k \)
15. else
16. \( a \leftarrow k; b \leftarrow i \)
17. end if
18. for each \( v_c \in V \mid t_a < t_c < t_b \) do
19. \( value \leftarrow y_b + (y_a - y_b) \frac{t_b - t_c}{t_b - t_a}; \)
20. if \( y_c > value \) then
21. \( visible \leftarrow false; \)
22. else if \( y_c < value \) then
23. \( visible2 \leftarrow false; \)
24. end if
25. if \( visible = false \) and \( visible2 = false \) then
26. break;
27. end if
28. end for
29. if \( visible2 \leftarrow true \) then
30. if \( (v_a, v_b) \notin E_p \) then
31. \( E_n = E_n \cup (v_a, v_b); \)
32. end if
33. else if \( visible \leftarrow true \) then
34. \( E_p = E_p \cup (v_a, v_b); \)
35. end if
36. end for
37. end if
38. End function
3.2. Algorithms for creating visibility graphs

In the above algorithm, the function TimeSeries2VisibilityGraphFast is responsible for generating the visibility graph by taking three parameters, the time series, the index of the leftmost value of the time series, i.e., 0, and the index of the right most value of the time series, i.e., \((n-1)\). We assign the index of the maximum value to variable \(k\). Then from index left to right the visibility of the node at \(k\) is checked with every other node in this window and signed edges are drawn according to the criteria. At the end of the function there is a recursive call to the function for node \(k\)'s left and right hand side parts with updated left and right index values. The implementation of this algorithm in Python [Program A.3] is in the Appendix. Figure 3.4 below shows the graph created after the implementation. Notice how Figure 3.2b and Figure 3.4 are structurally the same, the only difference being the time complexity to create them. It was observed empirically [27] that this algorithm has an average case time complexity of \(O(n \log n)\).
Chapter 4

Partitioning Algorithm

Our main work in this thesis is that of putting forth the idea of forming a clustering algorithm specially for time series (i.e. an algorithm that respects the time-feature of a series), using the technique of dynamic programming. As mentioned in Chapter 2, clustering (or partitioning or community detection) is a process of identifying certain collections of objects/nodes/members that possess similar properties/characteristics/interests within their collection and have dissimilarities with the objects belonging to other collections. In the case of signed visibility graphs, clustering can be done based on edge sign or weight, i.e. two nodes would belong to the same cluster if they have more positive ties within the cluster as compared to their ties outside the cluster or if they have more negative ties with other nodes outside the cluster than the nodes within the cluster. In this chapter, we will first briefly describe time-respecting clustering, followed by dynamic programming and an explanation of our algorithm.

4.1 Time-respecting Clustering

A partition of a visibility graph is time-respecting if all the nodes inside it are positioned in the network based on their time-points, and thereby the whole cluster follows time-constraint imposed by the structure of visibility graph. Two partitions are non-overlapping if the intersection between them is null, i.e., they do not have any node common or shared between them. In this work, we have divided a visibility graph $G$ in time-respecting non-overlapping clusters $[P_0, \ldots, P_{n-1}, P_n]$ in which $[t_0, \ldots, t_{(n-1)}, t_n]$, or the sets of time-points of nodes belonging to respective clusters, is in the form of consecutive time-intervals. For the given definition, we will define a measure of the quality of a time-respecting partition in the next section.

The basic scheme behind formation of clusters is that if within a set of nodes, there are more number of positive edges than negative edges joining the nodes and if outside the set, there are more negative edges than positive edges joining the set with other nodes, then the set forms a cluster. The aim is to find a solution of non-overlapping clusters such that the number of
positive edges inside and the number of negative edges outside any cluster are maximum values possible for the set of nodes that forms the cluster. Similarly, the number of positive edges outside and the number of negative edges inside any cluster must be the minimum values possible for the nodes that the cluster consists of.

Similar to the description of inter and intra-cluster positive densities in Chapter 2, we would now define a measure for finding the best-possible set of clusters. Let us take a visibility graph $G$ with clusters $(P_0, \ldots, P_{(n-1)}, P_n)$, then the overall quality, $Q$ of the clusters can be calculated as,

\[
Q = \text{Positive edges inside clusters}(P_0, \ldots, P_{(n-1)}, P_n) \\
+ \text{Negative edges between clusters}(P_0, \ldots, P_{(n-1)}, P_n) \\
- \text{Positive edges between clusters}(P_0, \ldots, P_{(n-1)}, P_n) \\
- \text{Negative edges within clusters}(P_0, \ldots, P_{(n-1)}, P_n)
\]

And our aim would be to maximize this overall “quality” for creating most efficient clusters, following conditions similar to the ones mentioned in Chapter 2 for creation of a community.

### 4.2 Algorithm for Time-respecting Clustering

The main idea behind the algorithm is to exploit the scheme of signed edges of a visibility graph to form clusters, by following dynamic programming. We will first briefly describe what dynamic programming is, followed by the algorithm.

#### 4.2.1 Dynamic Programming

The technique of Dynamic Programming has been used to solve numerous computing problems such as Weighted Interval Scheduling, 0-1 Knapsack problem, Segmented least-squares, etc [24]. In dynamic programming, the space of all possible solutions for a problem are explored by decomposing the problem into a series of subproblems, and then the correct solutions to larger and larger subproblems are gradually built up. Each of these subproblems is solved once and their solutions are stored using a memory-based data structure (array, map, etc). This technique is used in both mathematical optimization and computer programming. One of the main features of Dynamic Programming is maintaining an array for storing the results of subproblems, that can be iteratively computed, so that we do not have to
re-compute them later when the need be, which is mainly optimization over plain recursion. In our algorithm, as described in the next section, we use the same concepts to build the clusters.

4.2.2 The Algorithm

We will first define a measure of quality of the partition in the form of a recursive optimal solution,

For any node $i$ (such that $t_i < t_j$),

$$OPT[j] = \max_{0 \leq i \leq j} (q_{i,j} + OPT[i - 1])$$

and the partition $p_i, ..., p_j$ is used in an optimal solution for the subproblem if and only if the maximum is obtained using index $i$.

$OPT[j]$ is the maximum optimal quality value of time-respecting partitions on the subgraph of the first $j$ nodes, $q_{i,j}$ is the value of “quality factor” (calculated later in this section) for the cluster $(i, j)$ and $OPT[i - 1]$ is the optimal quality measure for clusters before node $i$, which is needed in this equation because the clusters in the subgraph of first $i$ nodes would have negative and positive edges with the cluster $(i, j)$ which are needed to be considered. These facts form the first crucial component of forming a dynamic programming solution: a recurrence equation that produces the optimal solution in terms of the optimal solutions to smaller subproblems.

For every node $j$ in the visibility graph, we take all the nodes to the left of it one by one (say, $i$) and check if there can be a cluster formation with all the nodes between $i$ and $j$ (including these two nodes). The decision for cluster formation is made by checking for the cumulative edge signs within the cluster which is done by computing the quality, $q_{i,j}$. The equation is,

$$quality = (pos_{in} + neg_{out}) - (neg_{in} + pos_{out})$$

Here, $pos_{in}$, the number of positive edges between the nodes within the cluster,

$neg_{out}$, the number of negative edges between the nodes in the cluster and the nodes outside of the cluster,

$neg_{in}$, the number of negative edges between the nodes within the cluster,

$pos_{out}$, the number of positive edges between the nodes in the cluster and the nodes outside of the cluster.

As per the discussion in previous section, our motto is to maximize $(pos_{in} + neg_{out})$, but for the quality measurement we need to minimize
4.2. Algorithm for Time-respecting Clustering

\( (\text{neg.in} + \text{pos.out}) \) too and therefore we aim at maximizing Equation (4.2) that considers all the edges involved with a cluster. The higher the value of quality measure of a cluster, the more favourable are the chances for the formation of that cluster. If a cluster has negative quality value, its formation would be very unlikely.

For each pair \((i, j)\) in the graph \([t_i < t_j]\), we calculate the quality, or \(q(i, j)\). If the value of \(q(i, j)\) is greater as compared with the quality values of other nodes to the left of \(j\), then we say that the node \(i\) gives us an optimal solution for the node \(j\). Hence, for the nodes explored so far in the visibility graph (till \(j\)), one cluster must be formed between \(i\) and \(j\), until we explore the graph more and find either \(i\) or \(j\) or both to belong to a cluster of a node with higher quality value.

Following the two equations is not the only way to be followed in finding the correct clusters - we need trace-backing, as done in dynamic programming, to actually form the clusters. Also, we maintain a globally accessible array that stores the values of optimal solutions of partitions that have been formed, to be used to find optimal solutions for partitions yet to be formed.

For creating the array, say \(M\), we start with \(M[0] = 0\), and iteratively compute the optimal solution for each node in \(V\) and store these solutions to use them in all future recursive calls. For a node \(j\), we use the value of optimal solutions to subproblems on interval \(\{1, 2, \ldots, j\}\) to define the value of \(M[j]\) using Equation (4.1). Below is the pseudo-code for the above explanation. Here, \(\text{Find_clusters}\) is used to trace back through \(M\) efficiently and return the list of partitions.

**Algorithm 4** Partitioning Algorithm

1. **Input:** A signed visibility graph \(G(V, E_p, E_n)\)
2. **Result:** List of clusters
3. **for** all pairs \(i \leq j\) such that \(i, j \in V\) **do**
   4. Compute maximum “quality” \(q(i, j)\) for partition \(p_i, \ldots, p_j\) using Equation (4.2)
5. **end for**
6. **for** \(j = 1, 2, \ldots, n\) **do**
   7. Use Equation (4.1) to compute \(M[j]\)
8. **end for**
9. Return \(M[n]\)
10. //Trace-back
4.3. Comparison with Louvain Algorithm

As an example, we ran the code with a sample data of the fMRI time series of a person (more details in Chapter 5) and got the results as shown in Figure 4.1. The Python code [Program A.4] for this implementation is mentioned in the Appendix. In Figure 4.1a the nodes with same colour-map belong to the same cluster and there are total 17 clusters generated by the algorithm for a graph of 236 nodes [details in Fig. 4.1b].

4.3 Comparison with Louvain Algorithm

In this section we will see if the approach of signed visibility graph and partitioning with our algorithm is more accurate as compared to a clustering algorithm, the Louvain Algorithm. We have compared our algorithm with Louvain algorithm because this algorithm was used in [41] for clustering in visibility graphs. We believe that any other standard community detection measure would give the same results of comparison as given by Louvain because other community detection algorithms work only on the network structure, without considering any other information such as the time constraint. We first give a brief description of the Louvain algorithm followed by the comparison.

4.3.1 Louvain Algorithm

In [4], the Louvain algorithm is used for community detection which is based on greedy technique and works on weighted graphs. It has a high modularity and runs in $O(n \log n)$ time complexity. It has a number of iterations and in each iteration, it calculates the modularity gain ($\Delta Q$), where modularity is simply a measure of the density of edges inside of the communities compared to the edges between communities and has a value between -1 and 1. The gain is the change in modularity when a node is removed from its own community and moved to the community of each of its neighbors. The algorithm mainly runs in two phases: in first level, all the vertices of the graph
4.3. Comparison with Louvain Algorithm

Figure 4.1: (a) An example visibility graph showing different clusters formed. The time series of this graph is Figure 4.2. (b) The output of details of clusters and number of clusters for the above example.

are put into separate communities, so the total communities are equal to the number of nodes. Then the $\Delta Q$ is calculated for each node and the node is moved into the community of a neighbour that has highest $\Delta Q$ value. This last step is looped until no further gains can be achieved in the system. In the second phase, partitions done in first phase are termed “super-vertices” and if there was at least one edge between the nodes of the corresponding partitions, then their super-vertices are connected, and the weights of all the edges between the nodes belonging to the 2 partitions are summed to denote the edge weight of a single edge between the corresponding super-vertices. These two phases are repeated until the system reaches stability.

Since the system may reach a different level of stability each time, this
4.3. Comparison with Louvain Algorithm

Algorithm may give different clusters in each run and therefore it is ran multiple times and an average clusters information from all these runs is taken into account.

4.3.2 Methods and Results of Comparison

To compare the performance and effectiveness of our algorithm with Louvain, we processed and extracted 1300 time-series from a fMRI dataset (more details in Chapter 5). A sample time series from the dataset is shown in Figure 4.2. For testing the Louvain algorithm, we used unsigned visibility graphs because Louvain algorithm works with positively-weighted graphs and since negative edges in signed visibility graph have negative weights, the algorithm doesn’t take those edges into account. For each graph, Louvain was ran 50 times and all the observations are the average of 50 runs.

Figure 4.3 shows the results of the comparison. The first point of noticeable difference was of the range of number-of-clusters values given by both the algorithms. While our algorithm gives a bigger range of number of clusters (lowest value being 2 and highest being 28), the Louvain algorithm doesn’t show much extreme variation in this number (lowest being 5 and highest being 12).

After a detailed examination of the clusters formed by Louvain, we have the following observations, in comparison with our method:
4.3. Comparison with Louvain Algorithm

Figure 4.3: The figure shows 2 line-plots for both the algorithms, blue line indicating the number of clusters for our algorithm and red line indicating number of clusters for Louvain algorithm.

- **Louvain algorithm does not respect the time-restraint as imposed by time-series and the structure of visibility graphs.** For proving this, we have a look at Figure 4.4 which shows two examples of how clustering is done in Louvain algorithm. Figure 4.4a is a visibility graph of one of the samples taken from the dataset used for comparison, that shows clusters indicated by different colors. In this figure, the nodes are placed according to their time points, i.e., they follow time-restrained positioning. When we observe the cluster indicated by color pink, we notice that it is discontinuous and is divided into three parts over the graph (the nodes of discontinuous clusters have been slightly enlarged for better visual observation).

A similar observation can be made in Figure 4.4c with the cluster in color pink, which is disintegrated in two parts in the visibility graph. Another discontinuous cluster is indicated by the color orange whose nodes a and b have a cluster (blue colored) between them. If the nodes in a visibility graph follow a time-order in their positioning, then their clusters are expected to be continuous (or whole or unbroken) and under the time-positioning restriction, but it is not followed in the Louvain algorithm. Figures 4.4b and 4.4d show the general network view of the graphs in figures 4.4a and 4.4c respectively. In these networks, nodes are not positioned according to their time-points but
4.3. Comparison with Louvain Algorithm

Figure 4.4: Two examples of clustering by Louvain algorithm: (a) and (c) show clustering in the visibility graph structure; (b) and (d) show the respective generic network views of graphs (a) and (c), without time-constraint positioning.

have been placed according to their connections. The Louvain algorithm basically clusters the nodes according to this structure.

- *The Louvain Algorithm forms weak clusters.* After observations of the quantity and integrity of the clusters formed by Louvain algorithm, we observed a weakness in the quality of the clusters too. The fundamental property of a cluster is that the nodes or elements within a cluster should be highly connected to each other, or must have more number of connections within the cluster than outside the cluster. A cluster
4.3. Comparison with Louvain Algorithm

Figure 4.5: Formation of weak clusters by Louvain Algorithm.

that doesn’t follow this property is not a strong cluster. Figure 4.5 shows a zoomed-in image of the graph in Figure 4.4a. In this graph, we first observe the cluster $X$ (or, cluster $(a, c)$) containing nodes $a$, $b$ and $c$. For it to be a strong cluster, the nodes between $(a, b)$ must have high number of connections with the nodes between $(b, c)$ but the only node from $(a, b)$ connected with the nodes in $(b, c)$ is the node $b$ itself. This doesn’t indicate a good cluster and therefore $(a, b)$ and $(b, c)$ must form two separate clusters (which happens in our algorithm). Another example of a weak cluster is the Cluster $Y$ wherein no nodes in $(d, e)$ except the node $e$ are connected with the nodes in $(e, f)$, therefore making $(d, f)$ a weak cluster.

The above two observations show that Louvain or any other community detection algorithm, that don’t take into consideration the time constraint, are not the best options for clustering of VGs. They also explain the reason why Louvain algorithm has such a low modularity, with an average number of clusters 9.1 for the dataset taken. Such low modularity is not a good measure for time series that have so many highs and lows in their values. Clustering in Louvain algorithm is only based on the weights of the edges, which isn’t a very useful factor to consider while partitioning nodes in a
4.4. Importance of Signed Edges and Time-respecting Constraint

time-series. Whereas our algorithm strictly follows the time-order of the series and therefore the clusters are formed with a time-respecting factor in consideration, in-sync with the structure of the visibility graph, and not just based on connections.

We can thus see that our algorithm is more effective in clustering visibility graphs where vertices have an implicitly-defined order as compared with Louvain algorithm because our method takes care of the sole feature of the time-series while clustering, the time. Another good feature of this algorithm is that it can deal with dynamic data while forming clusters, i.e., we can use a visibility graph in which the nodes keep adding dynamically and still create accurate clusters using our algorithm as it takes one node at a time for making decisions on the formation of a cluster involving that node. Everytime we need to form and see updated clusters after addition of one or few new nodes to the graph, we just need to run the trace-backing code and the clusters would be generated.

4.4 Importance of Signed Edges and Time-respecting Constraint

We will discuss why the concept of signed edges in visibility graphs and our partitioning algorithm give correct results of clusters, and also how the idea of dynamic programming solves such problems correctly. Let us look at Figure 4.6, which shows 4 of the many possible cases involved in the formation of clusters and analyze these one by one. In all figures, we have indicated edges by green and red colors to show positive and negative edges respectively and the figures are just a part of a bigger graph, used to show how backtracking affects the accuracy. All the edges shown are only to indicate the shape of the sub-graph shown and not all connections between nodes have been drawn.

We first have a look at Figure 4.6a, where we consider $j$ to be the last node of this part of the graph. Now, we consider a node $j'$ that has its highest quality value with node $i$ (i.e. $start_{j'} = i$) then accordingly we should have a cluster $(i, j')$. But by observation, this cluster is a wrong formation because the number of positive edges outside the cluster are high. The value of quality for $(i, j'+1)$ would be higher than that for $(i, j')$ simply because the number of positive edges are more in the former and there are no negative edges in either. Now for node $j$, $start_j$ would be $i$ as the cluster $(i, j)$ has only positive edges inside of it and the value of quality with node $i$ is very high. Now when we trace-back, node $j$ is picked first, forming
4.4. Importance of Signed Edges and Time-respecting Constraint

Figure 4.6: In this figure we see 4 of many possible cases found in the formation of clusters

the cluster \((i, j)\). If trace-backing wasn’t considered, then node \(j\) would probably form a cluster with node \(j' + 1\) as node \(i\) would be unavailable to it, because of the formation of cluster \((i, j')\). When we sum the values of \textit{quality} for the two clusters \((i, j')\) and \((j' + 1, j)\), we would find it to be quite low as compared to the value for cluster \((i, j)\) because of all the positive edges outside of the cluster that are getting added up in the denominator of Equation (3.1), decreasing the respective values. Therefore in this case the algorithm gives us the best possible cluster \((i, j)\) which is a stable choice for \(j', j' + 1\) and all other nodes within \((i, j)\).

Now we consider Figure 4.6b and see that in this case \(start_j \neq i\) because of a lot of negative edges between them and therefore we would probably get a cluster \((j'', j)\), if \(j\) has the most positive edges or least negative edges in \((j'', j)\).

We now consider the figures with a little bend in the concave or convex shape of the subgraph so that the subgraph has some negative edges inside the cluster. In Figure 4.6c, considering the node \(j'\) again, \(start_{j'} = i\) but the value of \textit{quality} for \((i, j' + 1)\) would be less as compared to \textit{quality} for \((i, j')\) because of the negative edges within the cluster. We would again have two
4.4. Importance of Signed Edges and Time-respecting Constraint

clusters \((i, j')\) and \((j' + 1, j)\) but the cumulative value of these two clusters’ 
quality would still be lower than the quality value of the cluster \((i, j)\) because 
even if there are some negative edges in \((i, j)\), and none in the clusters \((i, j')\) 
and \((j' + 1, j)\), they are outnumbered by the high number of positive edges 
that this cluster has, still giving us a very high quality value and therefore 
the algorithm gives us the cluster \((i, j)\). The domination of positive edges 
in a cluster with one or more such “bends” (that introduce negative edges 
within the cluster) is because visibility of the nodes that form that bend 
increases thereby increasing positive edges within the cluster and giving a 
higher value of Equation (4.2). The number of negative edges increases 
too because of the bends but the positive visibility is comparatively higher 
because of the presence of nodes that are taller (higher \(y\) value) than the 
bend(s)’s height(s). This case would be true until the bend height becomes 
equal to or more than the height of the tallest node in the cluster, i.e., until 
and unless \(y_{j'} \geq y_j\), the best cluster formed would be \((i, j)\). If \(y_{j'} \geq y_j\) then 
the clusters formed by the algorithm are \((i, j')\) and \((j' + 1, j)\).

In Figure 4.6d \((i, j)\) would form the worst cluster because of a lot of 
negative edges even if there are some positive edges introduced because of 
the bump. In this case, \(j\) may form a cluster with some other node \(j''\), \(j'\) 
and \(j' + 1\) would be a part of the same cluster (the small cluster covering the 
whole bend) and \(i\) would be a part of some other cluster of more positive 
edges than negative edges within. Any other cases of cluster formation in 
graphs and subgraphs would revolve around these 4 cases.
Chapter 5

Multivariate Correlation

In this chapter we will have a look at one of the applications of signed visibility graphs, correlation in multivariate visibility graphs.

5.1 Multivariate Time Series

All the time series examples we have covered in this thesis till now are univariate time series. A graph involving these only takes two parameters: time as an independent variable, and a single dependent variable. On the other hand, a multivariate time series is composed of more than one time-dependent variables. In a multivariate time series, each observation at a time $t$ is considered to be a vector of values instead of a single value, since at each $t$ there are more than one variable under observation. Typically for these series, there is a close interrelation between the variables in the vector, and this is the reason why they are considered to be a single observation in vector form instead of distinct observations from separate time series. These variables could be of different types and units (for example, the weather conditions involving pressure, temperature, etc.) or could be of the same type with different values at a each time-point (we would see an example of this type later).

Mathematically, consider $M$ time series variables $s_{1t}, ..., s_{Mt}$, then a multivariate time series involving these variables would be a $(M \times 1)$ vector $S_t$ where its $i$th row is $s_{it}$.

As far as mapping of time series goes, univariate time series are mapped to a single visibility graph and a multivariate time series is mapped to a multilayer visibility graph $\mathcal{M}$ with $M$ layers, where $M$ is the number of variables in the multivariate time series. A layer $\alpha$ of the multiplex graph $\mathcal{M}$ corresponds to the NVG associated with time series of state variable $[x^{[\alpha]}(t)]_{i=1}^{N}$. For every node $i$ in the graph, there is a natural correlation or consistency across layers of various univariate time series, so the resulting visibility graph is called a Multiplex Graph. This multiplex visibility graph encodes the complex structure of each time series in the topology of each layer. Figure 5.1 shows an example of a multivariate time series with $M = 3$.
variables, which is transformed into a multiplex graph of \( M = 3 \) layers (figure taken from [25, 32]).

A multivariate series consists of properties of not only the individual series it is made of, but also cross-relations among those series. These individual series are analyzed jointly in order to understand the dynamic relationships shared among them over time and also for the purpose of improvement of accuracy of these individual series’ forecasts by making use of the additional information available from the joint interdependencies [38]. For extracting these complex interdependencies and correlations that might emerge in the series across variables, certain similarity measures are used across layers.

### 5.2 Interlayer Mutual Information

One of the measures for extracting correlations between layers of a multiplex VG is the Interlayer Mutual Information that characterizes and measures the information shared by every two layers based on the similarity of the degree distributions [25, 32]. Since the degree distribution of any graph is the most basic metric for capturing the structure of each graph, the measure of mutual information captures the information shared between two layers, which is the information shared across each time series component of the multivariate time series. We would now define the measure of correlation, which has been mentioned in [25] for unsigned multiplex visibility graphs as \( \text{inter-layer mutual information} \ I_{i,j} \) for a given pair of layers \( i \) and \( j \) of the
\section*{5.2. Interlayer Mutual Information}

The interlayer mutual information graph $M$, \[ I_{i,j} = \sum_{k^i} \sum_{k^j} P(k^i, k^j) \log \frac{P(k^i, k^j)}{P(k^i)P(k^j)}. \] (5.1)

Here, $P(k^i)$ and $P(k^j)$ are the degree distributions at layers $i$ and $j$ respectively, $P(k^i, k^j)$ is the joint probability to find a node with degree $k^i$ and $k^j$ at layers $i$ and $j$ respectively.

Note that the formula in the above equation doesn’t consider negative edges. Since in our work we are working with signed visibility graphs, we adapted the measure in Equation (5.1) to include the degree distributions of the nodes by negative edges, defined as,

\[ I_{i,j} = \sum_{pk^i} \sum_{pk^j} \sum_{nk^i} \sum_{nk^j} P^*_{i,j} \log \frac{P^*_{i,j}}{P(pk^i)P(pk^j)P(nk^i)P(nk^j)}. \] (5.2)

where, $P^*_{i,j}$ is the term,

\[ P^*_{i,j} = P(pk^i, pk^j, nk^i, nk^j) \]

Here, $I_{i,j}$ is the inter-layer mutual information or the inter-layer correlation measure for signed multiplex VGs,

- $pk^i, pk^j$ are the positive degrees (or degrees by positive links) of a node at layers $i$ and $j$ respectively,
- $nk^i, nk^j$ are the negative degrees (or degrees by negative links) of a node at layers $i$ and $j$ respectively,
- $P(pk^i), P(pk^j)$ are the positive degree distributions at layers $i$ and $j$ respectively (or the probability to find a node having degree $pk^i$ at layer $i$ and degree $pk^j$ at layer $j$ respectively),
- $P(nk^i), P(nk^j)$ are the negative degree distributions at layers $i$ and $j$ respectively,
- $P^*_{i,j}$ or $P(pk^i, pk^j, nk^i, nk^j)$ is the joint probability to find a node having degrees $pk^i$ and $nk^i$ at layer $i$ and degrees $pk^j$ and $nk^j$ at layer $j$ respectively.

In general, two layers in $M$ will have more correlated degree distributions, if they have higher value of $I_{i,j}$ and, therefore, the two associated time series will have more correlated structures. We implemented the Equations (5.1) and (5.2) on the same dataset we used for the implementation of our clustering algorithm, and came up with the results in Figure 5.2. We took a part of the dataset for evaluation with 122 time series, therefore making a multiplex graph with $M = 122$. Each value in the plots is the mutual
5.2. Interlayer Mutual Information

Figure 5.2: Results of implementation of the two equations: first scatter plot indicates Equation (5.1) and second scatter plot indicates Equation (5.2).

Information measure between a pair of visibility graphs (layers) from the multiplex graph, and the values from all the visibility graph pairs have been plotted. These visibility graphs were of time series taken from Regions of Interest (ROIs) of human brain (more discussion in next chapter).
5.2. Interlayer Mutual Information

Table 5.1: Comparative results of multivariate correlation equations.

<table>
<thead>
<tr>
<th>Factors</th>
<th>Equation 5.1</th>
<th>Equation 5.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degree Distributions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(per pair of layers or per I-value)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P^*_\text{avg}$</td>
<td>0.00861</td>
<td>0.00429</td>
</tr>
<tr>
<td>$D_{\text{avg}}$</td>
<td>0.00637</td>
<td>$8.87\times10^{-5}$</td>
</tr>
<tr>
<td>Number of iterations</td>
<td>504</td>
<td>238.8k</td>
</tr>
<tr>
<td>(per pair of layers or per I-value)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Range of values</td>
<td>0.55 - 1.27</td>
<td>6.1 - 7.68</td>
</tr>
<tr>
<td>Variance</td>
<td>0.010284</td>
<td>0.073642</td>
</tr>
<tr>
<td>$\Delta_{\text{avg}}$</td>
<td>0.089</td>
<td>0.195</td>
</tr>
</tbody>
</table>

In Figure 5.2 we notice the variation in values by both the equations. While Equation (5.1) gives us values within a range of 0.75 (maximum: 1.25, minimum: 0.5), Equation (5.2) gives a broader range of 1.7 (maximum: 7.7, minimum: 6). Equation (5.2) gives more varied and diverse values for exactly the same time points for which Equation (5.1) doesn’t show much variation. We carried out a break-down analysis of the equations, computing values of their components and noted the results in Table 5.1 for in-depth examination of the values obtained in the figure. These numerical results in the table are specifically for the dataset used, except for the first row of the table.

In Table 5.1, $P^*_\text{avg}$ is the average of joint probability value (or the numerator in the log function) and $D_{\text{avg}}$ is the average value of the products of degree distributions (or the denominator in log function), taken for each iteration and averaged for both the equations. $\Delta_{\text{avg}}$ is the average of difference between two I-values. The average of total number of summations (or the total number of iterations) for calculating I-value between each pair of adjacent layers is also noted. Notice that due to the consideration of positive and negative edges, Equation (5.2) has total 4 degree distribution parameters per pair of layers, whereas Equation (5.1) has 2 due to only the positive edges.

From Table 5.1 we have the following observations:

- The $P^*_\text{avg}$ is quite low (almost half) for Equation (5.2) as compared to (5.1), which means the joint probability that satisfies the degrees at a particular iteration is lower for the modified Equation. Equation (5.2) has 4 degrees in one run and it becomes rarer to find a node
5.2. Interlayer Mutual Information

that satisfies all the four degrees requirements, thereby decreasing the value of joint probability.

- The $D_{avg}$ is comparatively quite high for Equation (5.1) ($10^2$ times) than (5.2). Although the degree distributions due to positive edges will be the same for both the equations, the point of difference is introduced by low values of the degree distributions due to negative edges (degrees due to negative edges are lower in value than positive edges, giving low degree distributions).

- There is a huge difference in the number of iterations or number of times the calculated values are summed to get one $I$-value. This result is supported by the fact that there are 4 nested loops due to 4 degree summations in Equation (5.2) as compared to 2 nested loops in Equation (5.1).

- Equation (5.1) has a shorter range of values with smaller values as compared to (5.2). Although the $P_{avg}^*$ difference between the two equations is 0.00432, the major difference in the values is due to extremely low denominator of the log function ($D_{avg}$) in Equation (5.2), that gives a larger log function value. Along with this, Equation (5.2) undergoes 474 times more summations for each $I$-value, thereby giving us these results.

- Higher Variance (about 7 times) and higher $\Delta_{avg}$ (almost double) values of Equation (5.2), along with the visual picture in Fig. 5.2 give us an idea of how spread out the $I$-values are, due to the consideration of negative edges.

We ran an F-test to compare the two variances given by the two equations, getting F-value of 7.16 and p-value $< 2.2e-16$ for 95 percent confidence interval. For the same pair of visibility graphs, the magnitude of information about correlation given by (5.2) is higher as it has more diverse values compared with values given by (5.1) that are not very divergent.

To make it clearer, let’s take 4 sample time-series, as shown in Figure 5.3. For the comparison, $I$-values were calculated between time-series1 and each of the other three time series, using both the equations and the values are noted in Table 5.2. We can see that for the 3 same pairs of visibility graphs, Equation (5.1) gave almost similar values: 0.720, 0.719 and 0.721, whereas Equation (5.2) have much more varied values: 6.520, 6.444 and 6.243, respectively. The time-series 2, 3 and 4 are very different from each other in structure [Fig.5.3] but still, when correlated with time-series1, Equation
5.2. Interlayer Mutual Information

Figure 5.3: 4 sample time series used for calculating $I$-values using both the $MI$ equations.

Equation (5.1) gave very similar values for them. Equation 5.2 was able to give us a better idea about the amount of correlation between these time-series as it gave dissimilar values. Using Equation (5.1), we have to observe values till hundredth or even thousandth place of decimal to find any significant difference in the $I$ value, whereas values given by Equation (5.2) only need to be observed till tenth decimal place to see significant differences.

As compared to time series 4, time series 1 is structurally more similar to time series 2. The values of $I$ given by Equation (5.1) for mutual information of time series 1 with time series 2 and 4 respectively are extremely similar, suggesting similar structures, which is not the case. Equation (5.2) on the other hand, gives us a higher $I$-value between time series 1 and 2, and a lower $I$-value between time series 1 and 4, indicating the correct degree of their structural similarities.

It is evident that for the same pair of graphs, Equation (5.2) gives values with finer or more detailed information about the amount of correlation present between the graphs. This measure provides more help in detecting more subtle correlation structures. This difference supports the fact that considering both positive and negative edges is more informative in terms of the level of correlation than considering only the positive edges. The added
5.2. Interlayer Mutual Information

Table 5.2: Comparison between values given by both equations for the same pairs of visibility graphs.

<table>
<thead>
<tr>
<th>MI calculated between</th>
<th>Equation 5.1</th>
<th>Equation 5.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time-series1 - time-series2</td>
<td>0.7204</td>
<td>6.5206</td>
</tr>
<tr>
<td>Time-series1 - time-series3</td>
<td>0.7199</td>
<td>6.4442</td>
</tr>
<tr>
<td>Time-series1 - time-series4</td>
<td>0.7218</td>
<td>6.2433</td>
</tr>
</tbody>
</table>

structure feature, the negative edges, have helped in getting better insight into the structure correlation between two visibility graphs.
Chapter 6

Application to Brain Network Analysis

In this chapter, we have applied our framework to study brain networks represented in the form of brain images (fMRI) and electric signals (EEG) which are used for detection, diagnosis and study of neurological diseases and functioning of the brain. In this study, we will use the data generated by these techniques to find alterations or differences in brain-functioning of patients with different mental disorders, by using the concepts of networking and graphs.

6.1 Terminologies

Before we go further with the description of the fMRI data, there are several important concepts that need to be put under light.

6.1.1 Data and Preprocessing Techniques

- **Parcellations**: Defining brain structures of interest is an important preliminary step in brain-connectivity analysis. Brain Parcellations or Atlas are templates that divide the brain’s spatial domain into a set of non-overlapping regions (ROIs) or modules based on functional homogeneity. In a parcellation, the image voxels are grouped into structurally or functionally homogeneous regions, each of which is represented by a node in the corresponding network and connectivity is defined among all voxels in a pair of regions [44].

- **ROIs (Regions of Interest of Brain)**: Region of Interest (ROI) analysis in neuroimaging refers to selecting a cluster of voxels or brain region when investigating a region for effects. In most simple words, an ROI is simply a part of the brain (or a subnetwork) used for study of specific areas in brain having specific functionality.
6.1. Terminologies

- **RSNs (Resting State Networks):** These are brain areas that consist of many ROIs, divided based on functional similarity. According to Yeo’s networks, the brain is divided into left and right hemispheres, which are further divided into 7 non-overlapping functionally homogeneous sub-regions or RSNs namely Visual, Somatomotor, Dorsal Attention, Ventral Attention, Limbic, Frontoparietal control and Default, as shown in Figure 6.1 taken from [46].

- **Data Preprocessing:** Before the raw functional data can be put to use, a series of operations is typically performed on it, in order to reduce artifact and noise-related signal components. Some of these operations include slice timing correction, motion correction/realignment, normalization, interpolation, and smoothing, etc. along with some image preprocessing operations such as noise reduction, bias correction, brain extraction, and segmentation. Preprocessing is also responsible for standardizing the location of brain regions across different subjects to achieve validity and sensitivity in group analysis. There are various pipeline softwares available for this purpose like NIAK (NeuroImaging Analysis Kit) and fMRIPrep. NIAK is an open-source library of pipelines for the preprocessing and mining of connectomes extracted from large fMRI data, that performs all the previously mentioned preprocessing operations along with data-driven functional brain parcellation. Similarly, fMRIPrep automatically adapts a best-in-breed
6.1. Terminologies

- **BIDS** (*Brain Imaging Data Structure*): It is a data organization format especially for neuroimaging experiment data and is designed to facilitate its transfer, description, and storage. BIDS provides a disciplined way to arrange the different data types, is intuitive and easy to adopt. The specification is based on simple folder structures and file formats to reflect current lab practices and make it widely accessible to scientists coming from different backgrounds [19].

- **MNI-152** (*Montreal Neurological Institute-152*): For the purpose of analysis of fMRI data, registration of the data to a reference brain template is required in order to identify the activated brain regions. A brain template is an anatomical representation of the brain depicting finer anatomical details. The MNI-152 brain template was constructed by averaging 3D brain MRI images of 152 normal subjects. The advantage of MNI-152 brain template is that it provides a full head coverage, covering the top portion of the brain to the bottom portion of the cerebellum [29].

- **NIfTI-1** (*Neuroimaging Informatics Technology Initiative*): It is a very popular file format for storing the fMRI imaging data, which consists of a header file .hdr, to store meta-information, and the actual data file, with extension .img, the two files usually stored as a combined single file, with extension .nii. It also consists of an accompanying file describing the orientation in space, usually in .mat format. In the nifti format, the first three dimensions define the three spatial dimensions - x, y and z -, while the fourth dimension defines the time points - t.

- **Jaccard Index**: The Jaccard index, also known as Intersection over Union, is a statistical measure or comparison metric of similarity and diversity in two sample sets, to see which elements of the sets are shared and which are distinct. Higher the Jaccard index value between two sets, more similar they are. Mathematically, it is the size of the intersection divided by the size of the union of the sample sets:

\[ J(A, B) = \frac{|A \cap B|}{|A \cup B|} \]

- **Extracranial and intracranial EEG**: As mentioned in Chapter 2, EEG is a monitoring method used in recording electrical activity of
the brain. It is usually recorded by placing the device electrodes on the person’s scalp with a conductive gel, and this type of recording is called extracranial recording. The number of electrodes generally depends on the device can range from 2 to 256. When the required EEG recordings cannot be taken from the surface of the scalp (or, outside the skull), electrocorticography or intracranial EEG is used. In this, electrodes are placed directly on the exposed surface of the brain (or, inside the skull) for recording from the cerebral cortex.

6.1.2 Neurological Disorders

- **ADHD** (*Attention Deficit/Hyperactivity Disorder*): ADHD is the most commonly diagnosed mental disorder, found in both adults and children. Children with ADHD may be hyperactive and unable control their impulses, or they may have trouble paying attention. Adults with ADHD may have trouble managing time, being organized, setting goals, and holding down a job.

- **Bipolar Disorder**: Bipolar disorder is a brain disorder that causes unusual shifts in mood, energy, activity levels, and the ability to carry out day-to-day tasks. A person with this disorder can either be depressed and may feel sad or hopeless and lose interest or pleasure in most activities, or can suffer from mania (or hypomania) in which they may feel euphoric, full of energy or unusually irritable.

- **Schizophrenia**: Schizophrenia is a chronic and severe mental disorder that interferes with a person’s ability to think clearly, make decisions, manage emotions, and relate to others, along with impairing a person’s ability to function to their potential. Unfortunately, no single, simple course of schizophrenia treatment exists.

- **Epilepsy**: It is a chronic neurological disorder in which the person has recurrent, unprovoked seizures, or a brief disruption in normal activity of brain, interfering with brain function. It involves a loss of consciousness and rhythmic jerking movements. It could be caused due to a brain injury or genetic factors.
6.2 Analysis of fMRI Data

6.2.1 fMRI data

For fMRI implementations, we took the data of resting state fMRI of patients. The data was retrieved from 2 online repositories. The first dataset was taken from NeuroImaging Tools and Resources Collaboratory’s (NITRC) initiative ADHD200, which is a part of its 1000 Functional Connectome Project and is described in [9]. This initiative was started for understanding of the neural basis of ADHD through open data-sharing and discovery-based science, and is the source of unrestricted publically released resting state fMRI data of typically developing individuals with ADHD (ages: 7-21 years) [8]. For this study, the data contributed by Peking University and Beijing University was used. The preprocessed data of these samples was taken from the repository made available by the Neuro Bureau at [33]. Out of the available options of preprocessing strategies, we used the data from the NIAK on CBRAIN, as this technique preprocesses resting state fMRI data and extracts the time courses, which is our basic requirement. The data taken was “Beijing 1000 ROI extracted time courses” which is organized in the form of 954 ROIs for each patient, and each ROI has its own time series.

In the data, the term ADHD Index was defined to be a combination of the measures of “Attentiveness” and “Hyper/Impulsiveness” and each patient was given a score for these measures, maximum values being 36 and 30 respectively. Out of the 259 patients, we chose 10 patients who have the highest ADHD Index value (or the highest “inattentiveness” and “impulsiveness” values) and 10 patients that have the lowest ADHD Index values. All the selected patients have almost similar values for Verbal IQ (range 75-149) and Performance IQ (range 80-135) with an age range of 9-15 years.

The second dataset was taken from a large online repository openfMRI with accession number ds000030. This dataset has been contributed by the University of California, Los Angeles for their Consortium for Neuropsychiatric Phenomics (CNP) LA5c Study, as described in [37] and available at [35]. The dataset includes an extensive set of task-based fMRI assessments, resting fMRI, structural MRI, and other imaging data of 138 healthy subjects, as well as samples of individuals diagnosed with schizophrenia (58), bipolar disorder (49), and ADHD (45). For both healthy and patient groups, participants were of ages 21-50 years and the data is formatted according to the BIDS standard. The preprocessed rs-fMRI data, as described in [18] was
6.2. Analysis of fMRI Data

Figure 6.2: Comparison between the average number of clusters found in the patients with high (blue) and low (orange) ADHD index. Each point in the plot denotes a patient’s value, and each value is the average of number of clusters formed by 15 randomly chosen ROIs.

taken from [36] in the form of NIfTI-1 image files, which was preprocessed using fMRIprep pipeline with the MNI152 template. For our study, data of 10 healthy subjects as well as 10 subjects each from the 3 patients sets was taken, all subjects randomly chosen.

6.2.2 Results

ROI level

For the most basic ROI level, the data was taken from ADHD200 preprocessed database’s 1000 ROI time-course for 20 patients, as described before. First to check if any relevant differences exist between the high ADHD-index patients and low ADHD-index patients pertaining to our algorithm, we randomly chose 15 ROIs out of 954, and formed clusters using the visibility graphs of their time-series. We then averaged the total number of clusters in these ROIs for each patient.

In Figure 6.2, we plotted the average number of clusters for each patient belonging to the set of high ADHD-index (set-1, blue plot) and lower ADHD-index (set-2, orange plot), along with regression lines and LOWESS (Locally Weighted Scatterplot Smoothing) lines. The patients were indexed based on the values of ADHD-index - descending values for high ADHD patient set
6.2. Analysis of fMRI Data

and ascending values for low ADHD patient set. In the figure, relevant differences were noted in the average values of these patients. The average of these values is 15.33 for set-1 and 14.44 for set-2. Overall, the average number of clusters for patients with low ADHD are lower as compared to patients with high ADHD index. Welch’s t-test was applied for equal sample

Figure 6.3: Comparison between the average number of clusters found in the patients with high (green) and low (pink) ADHD index for the 7 RSNs. Each point in the plot denotes a patient’s value, and each value is the average of number of clusters formed by all the ROIs in the respective RSN.
sizes, unequal variances with a 95% confidence interval \((\alpha = 0.05)\) and the resulting t-value was 2.5177 with the p-value of 0.02436.

**Groups of ROIs level**

In the same dataset, ROIs were grouped together to form sub-networks or RSNs based on Yeo’s parcellation networks [46] with ROIs that are functionally similar belonging to the same RSN. For grouping, we chose the 7 networks approach with 1000 parcels. Here, the parcels are actually ROIs and they were divided into 7 RSNs by the method of matching as done in [42]. We matched our ROIs with the parcels of Yeo’s networks by matching the ROI number with the parcel number and assigned the network that the parcel is a part of, to the ROI, i.e., the RSN of each ROI was assigned based on its spatial overlap with the original 7-network parcellations. With the method of matching, we could group all the ROIs and got 7 groups with alias Cont, Dors, Limbic, Sal, Som, Vis and Default to carry out the evaluations.

For each patient, we had 954 time series (one for each ROI) divided into 7 RSNs. We considered 18 patients, 9 per set, and used 18×954 visibility graphs to generate the following results. Figure 6.3 shows the plots of average number of clusters for each RSN, along with regression lines. In each plot, one point denotes one patient, coloured according to the set it belongs to, and the plotted value is the average of the number of clusters formed by all the ROIs belonging to that RSN. The patients were indexed based on their ADHD-index values.

In RSNs Vis, Limbic, Dors, Sal, and Som, high ADHD patients showed higher average values, whereas in Default and Cont, the patient-sets did not show much difference. We now know which RSNs are majorly responsible for the differences found in the previous section. Dorsal Attention (Dors) and Salvent (Sal) are two attention systems of the brain [43], Limbic is responsible for learning, memory and emotion and these RSNs have shown differences in number of clusters. Welch’s t-test for these three RSNs gave p-values of 0.0055 for Dors, 0.0090 for Sal and 0.0011 for Limbic.

Figure 6.4 shows the comparison between the number of clusters generated by the Louvain algorithm and our algorithm for the RSNs Limbic, Dors and Sal - the RSNs responsible for learning, emotion and attentiveness in our brain. As we can observe, the results from Louvain don’t show much difference in the number of clusters of high ADHD and low ADHD patients. Welch’s t-test for the Louvain algorithm gave p-values of 0.551 for Dors, 0.0751 for Sal and 0.498 for Limbic.

To find out differences between contents of the clusters of 2 ROIs in
each patient, we applied Jaccard Index. We first took a patient’s RSN and

Figure 6.4: Comparison between the number of clusters generated by our algorithm and the Louvain algorithm for RSNs Limbic, Dors and Sal. Left column represents results from our algorithm, right column represents Louvain’s algorithm results.
6.2. Analysis of fMRI Data

Figure 6.5: Comparison between the Jaccard Index between clusters of patients with high (blue) and low (orange) ADHD index for the 7 RSNs. A value in the histogram denotes the average of number of cluster-pairs, between each ROI-pair, that have Jaccard Index values greater than equal to 0.7 (or 70%), calculated for all ROI-pairs of one RSN of one patient.

chose 2 ROIs from it. For the ROI-pair, we applied Jaccard index (JI) between the clusters of the ROIs (considering one cluster as one set) and noted the number of cluster-pairs for which the JI-value is greater than or equal to 0.7 (or the JI-percentage is greater than 70%). We calculated these numbers for all the ROI-pairs in the RSN, averaged it and plotted it in the histogram as one value. Same steps were applied to get the average number of cluster-pairs with ≥ 0.7 JI-value, for all the RSNs of the 9 patients.

Figure 6.5 represents the results of implementing above steps, for the two patient-sets. The $y$-axis is the average number of clusters with JI-value ≥ 0.7 per RSN per patient and the $x$-axis denotes the RSNs. Values of patients with high ADHD are colored in blue whereas orange color denotes values of patients with low ADHD. Regression or trendlines show that the average number is higher for patients with low ADHD, more specifically in Cont and Sal. Welch’s t-test generated p-value of 0.11 for RSN Cont and 0.099 for RSN Sal. It shows that there are clusters between 2 ROIs with high Jaccard indexes, and it is likely that during these segments the two ROIs reflect similar neural events [41]. Therefore, neural similarity is more in low ADHD index patients.

To check the differences in inter-ROI mutual information between the two patient-sets, we first took data from one RSN of one patient and calcu-
6.2. Analysis of fMRI Data

Figure 6.6: Comparison between the average inter-ROI mutual information between patients with (a) high and (b) low ADHD index for the 7 RSNs.

Figure 6.6: Comparison between the average inter-ROI mutual information between patients with (a) high and (b) low ADHD index for the 7 RSNs.
6.2. Analysis of fMRI Data

lated the interlayer mutual information between all the ROIs of that RSN using Equation (5.2). We then plotted the average of the calculated values, 9 values (one per patient) per RSN, for the two sets. The number of ROI pairs per RSN depends upon the number of ROIs the RSN has, for example, RSN Vis has 162 ROIs, RSN Som has 194, etc.

Figure 6.6 shows the results in the form of violin plots. For each set, each circle corresponds to \(<MI>\), the average mutual information value, of a given subject. Random average shifted histograms that show statistical measures like kernel-density estimates, median of Harrell-Davis estimator, and the 95% high density intervals using a Bayesian bootstrap are also shown in the figure, generated using ggplot’s violin plot in R programming language. The density estimator has higher value for RSNs of low-ADHD patients, and there is a clear differentiation in all the RSNs. T-test for the RSNs gave p-values of 0.0034 for Sal, 0.044 for Limbic and 0.010 for Som.

Figure 6.7 shows the results of average inter-ROI mutual information between patients with high and low ADHD index for the 7 RSNs, using Equation (5.1). There is not much difference that can be observed in the structure of violin plots between low and high ADHD data values. T-test for the RSNs gave p-values of 0.1495 for Sal, 0.2676 for Limbic and 0.3373 for Som.

RSN level

For carrying out implementation at RSN level, we used data from the UCLA CNP study of healthy individuals and patients suffering from Schizophrenia, ADHD and Bipolar Disorder. As mentioned earlier, the preprocessed data was in the form of NIfTI-1 image files, out of which the time series were extracted using Yeo atlas, as described in [46]. For the implementation, we first parcellated the data following Yeo parcellation into 7 subregions (or RSNs), using the python library “nilearn” which is a module for fast and easy statistical learning on NeuroImaging data. Then we used NiftiLabelsMasker in Python to generate the time series, one series per RSN. These 7 time-series were then used to generate the RSN-specific results, for each of the 40 individuals.

Our data has 4 sets: Controls, Schizophrenia, ADHD and Bipolar Disorder and we will observe how these sets differ from each other in terms of number of clusters and mutual information. Each patient has 7 visibility graphs (one per RSN), and for each graph, clusters were generated and their number was plotted in a pirate plot in Figure 6.8, in the order of each patient set and each RSN. A point in the plot corresponds to the number
6.2. Analysis of fMRI Data

Figure 6.7: Comparison between the average inter-ROI mutual information (using Equation (5.1) between patients with (a) high and (b) low ADHD index for the 7 RSNs.
6.2. Analysis of fMRI Data

Figure 6.8: Comparative pirate plot showing different number of clusters for each patient-set (A: ADHD, BD: Bipolar Disorder, C: Controls and S: Schizophrenia), per RSN.

of clusters in the visibility graph of one RSN of one patient. The plot has mean-value marker, full-densities and 95% Highest Density Intervals (HDIs) for each group. In most of the cases, the set Schizophrenia has the highest number of clusters, and Controls mostly has higher or almost similar values than Bipolar, based on the mean values. We got a p-value of 0.046 of values for controls and schizophrenics for RSN Dors.

Figure 6.9 shows the difference in mutual information values within each of the RSNs, for the four groups of subjects considered, in the form of a violin plot. Each point in the plot corresponds to mutual information between visibility graphs of two patients belonging to the same set, considering the same RSN. As it can be observed, all the RSNs from all the sets show different densities and estimator values, with more differences shown in Limbic and Sal, and lesser differences shown in Cont.
6.3 Analysis of EEG Signals

We will now observe alterations or differences between patient-sets and controls using EEG signal time series.

6.3.1 EEG Data

The data was retrieved from [1] which is a study on healthy individuals and patients with Epilepsy, as mentioned in [3]. The data has 5 sets, and each set consists of EEG values from 100 individuals recorded for a duration of 23.6 seconds. Each time series has 4096 value points. Below is a brief description of the sets:

- **Set A**: Extracranial recording of healthy subjects with eyes open.
- **Set B**: Extracranial recording of healthy subjects with eyes closed.
6.3. Analysis of EEG Signals

Figure 6.10: Comparison between number of clusters for the 5 sets of EEG values.

- **Set C**: Intracranial recording of epileptic patients from the hippocampal part of the brain, without seizures.
- **Set D**: Intracranial recording of patients from within the epileptic zone of the brain, without seizures.
- **Set E**: Intracranial recording of patients from within the epileptic zone during seizures.

The data was already preprocessed and we used all the 500 time series for our implementations.

6.3.2 Results

The first implementation towards finding any differences between the patient-sets and healthy individuals was to create signed visibility graphs of the EEG time series and form clusters according to our methods, for all the time series belonging to the 5 sets. After creating the clusters, we noted their quantity in each time series and plotted the graph in Figure 6.10.

In the figure, the bar denotes mean values for each set with ± standard error of the mean. The mean values are also mentioned at the bottom of each bar for the particular set. Each point in the scatter plot denotes number of clusters in one time series belonging to the respective set. Set B has the
highest number of clusters, but set E has more spread-out or diverse values (higher variance). Higher the number of clusters, smaller the cluster size since the number of value points are same. Smaller the cluster size, more the number of high and low (or extreme) value points in the data. Patients with epilepsy (without seizures) showed lower number of clusters, suggesting that they have smaller variation in their value points. P-value of Welch’s t-test between set B and set D came out to be 2.2e-16, with a t-value of 18.835. Since set E was recorded during seizures, the EEG values would have been highly fluctuating, therefore giving these results.

To examine the similarities or differences within the clusters, we calculated Jaccard Index. For each pair of time series belonging to a set, the Jaccard Index values were calculated, so as to know which set has more similar elements in the clusters.

Figure 6.11 shows a comparative box-plot scatter chart for the Jaccard Indexes in the 5 sets. Each point in the plot denotes JI-value between two time-series belonging to same set. Set B has the highest average JI-value, showing that the time-series in this set have highest similarities with each other. Set E has high mean JI-value too, but it consists of a lot of outlier
6.3. Analysis of EEG Signals

values, probably due to the fact that seizures introduce a lot of randomness in the EEG values. Sets C and D have almost similar and low JI-values, indicating that the time series are more different in these sets.

For our final result, we calculated the cluster sizes for each time series in each set, expecting opposite results from Figure 6.10, i.e., higher the number of clusters, lower the clusters’ sizes. Figure 6.12 shows the results and as expected, we see that sets B and E, that had the highest number of clusters, have the lowest average cluster size values.

Figure 6.12: Comparison between average cluster sizes for the 5 sets of EEG values.
Chapter 7

Conclusion

In this thesis, we introduced a new concept of signed visibility graphs that can be used to improve the effectiveness of graph-theoretical approach to analyse time series data. We also studied about partitioning in visibility graphs and designed a new clustering algorithm based on signed visibility graphs and dynamic programming, that creates disjoint time-respecting clusters.

We compared our algorithm with the Louvain algorithm of community detection to find that our algorithm gives more reliable results in terms of clustering of visibility graphs. We also redefined the mutual correlation measure between two time-series by including the degree distributions due to negative edges. Comparison between the original mutual correlation measure and the redefined measure showed that our measure is more informative in terms of the amount of correlation between any two visibility graphs.

We then applied our methods to datasets of fMRI and EEG signals of healthy individuals and patients suffering from certain neurological disorders and found differences in the results of implementations.

For future study, the methods and concepts discussed in this work could be applied to many other fields to get some useful insights. The results noted in last chapter could be used to find reasons behind the differences observed, adding to the knowledge of these disorders for better identification and diagnosis. This work could also be a source of reference for creating more time-respecting clustering algorithms using other notions in network science.
Bibliography


Appendix A

Implementation Programs

We will list all the Python codes for the implementations done in this thesis here.

Following is the program for creating natural signed visibility graphs following Algorithm 1, mentioned in Chapter 3.

Program A.1 Program for Natural Signed Visibility Graphs

```python
import networkx as nx
import matplotlib.pyplot as plt
t = list(range(0, len(y)))
nodes = list(range(0, len(y)))
v1 = []  # v1 consists of first set of nodes
v2 = []  # v2 consists of the nodes which have an edge with nodes in v1, stored in the same order as v1
inv_v1 = []
inv_v2 = []
posn = {}
for a in range(0, len(y)):
    v3 = []
    for b in range(0, len(y)):
        visible = True
        visible2 = True
        if (t[b] > t[a]):
            if (t[b] == t[a] + 1):
                v1.append(a)
                v3.append(b)
            else:
                for c in range(0, len(y)):
                    if (t[c] < t[b] and t[c] > t[a]):
```


Appendix A. Implementation Programs

value = y[b] + ((y[a] - y[b]) * ((t[b] - t[c]) / (t[b] - t[a])))
if(y[c] > value):
    visible = False
elif(y[c] < value):
    visible2 = False
if(visible== False and visible2==False):
    break
if visible == True:
    if b not in v3:
        v1.append(a)
        v3.append(b)
    elif visible2 == True:
        if b not in inv_v3:
            inv_v1.append(a)
            inv_v3.append(b)
    v2.extend(v3)  #merge the newly found nodes
inv_v2.extend(inv_v3)
#Code for creating the graph
G=nx.Graph()
G.add_nodes_from(nodes)
for i in range(0, len(v1)):
    G.add_edge(v1[i], v2[i], weight = 1, color = 'g')
for i in range(0, len(inv_v1)):
    if(G.has_edge(inv_v1[i], inv_v2[i])==False):
        G.add_edge(inv_v1[i], inv_v2[i], weight = 0, color = 'r')
for node in nodes:
    posn[node] = (t[node], y[node])
edges = G.edges()
colors = [G[u][v]['color'] for u,v in edges]
position = nx.spring_layout(G, pos = posn, fixed = nodes)
nx.draw(G, pos=position, edge_color=colors, with_labels=True,
        node_color = 'b')
plt.show()
Appendix A. Implementation Programs

Following is the Python code for implementing Algorithm 2 as mentioned in Chapter 3, for creating a horizontal signed visibility graph.

**Program A.2** Program for Horizontal Signed Visibility Graphs

```python
code
sortList.append(y[0])
for i in range(1, len(y)):
    G2.add_edge(y[i], y[i-1], weight = 1, color = 'g')
    if(y[i]>y[i-1]):
        loc = -1
        for g in sortList:
            if(g<y[i]):
                loc = loc+1
            else:
                break
        if(loc>-1):
            counter = 0
            while(counter<=loc):
                G2.add_edge(y[i], sortList[0], weight = 1, color = 'g')
                del sortList[0]
                counter = counter+1
    sortList.append(y[i])
sortList.sort()
#Code for FHVG (Negative edges)
max_y2 = max(y)
inv_y2 = []
inv_sortList = []
for i in range(0, len(y)):
    inv_y2.append((max_y2+1)-y[i])
in
```
Appendix A. Implementation Programs

```python
y_val = (max_y2+1)-inv_y2[i]
sort_val = (max_y2+1)-inv_sortList[0]
if(G2.has_edge(y_val, sort_val)==False):
    G2.add_edge(y_val, sort_val, weight = 0, color = 'r')
del inv_sortList[0]
counter = counter+1
inv_sortList.append(inv_y2[i])
inv_sortList.sort()

#Code for creating the graph
for node in nodes:
    posn[y[node]] = (t[node], y[node])
edges = G2.edges()
for u,v in edges:
    colors = G2[u][v]['color']
    position = nx.spring_layout(G2, pos = posn, fixed = y)
xn.draw(G2, pos=position, edge_color=colors, with_labels=True, node_color = 'b')
plt.show()
```

Following is the program for implementing the divide and conquer algorithm (Algorithm 3) for creating a signed visibility graph.

**Program A.3 Program for Divide and Conquer algorithm**

```python
TS = list(y)
def TStoVGfast(TS2, left, right):
    if(left<right):
        k = TS2.index(max(TS2))
        k = k+left
        for i in range(left, right+1):
            if(k==i+1 or i==k+1):
                G.add_edge(k, i, weight = 1, color = 'g')
            else:
                visible = True
                visible2 = True
                for j in range(left, right+1):
                    if(k<i):
                        if(j<i and j>k):
                            value = y[i] + ((y[k] - y[i]) * ((t[i] -
                                                        t[j]) / (t[i] - t[k]))
```
Appendix A. Implementation Programs

```python
if(y[j] > value):
    visible = False
elif(y[j] < value):
    visible2 = False
if(visible== False and visible2==False):
    break
if(i<k):
    if(j<k and j>i):
        value = y[k] + ((y[i] - y[k]) * ((t[k] - t[j]) / (t[k] - t[i])))
        if(y[j] > value):
            visible = False
        elif(y[j] < value):
            visible2 = False
        if(visible== False and visible2==False):
            break
        if(visible==True and k!=i):
            G.add_edge(k, i, weight = 1, color = 'g')
        elif(visible2==True and k!=i and G.has_edge(k, i)==False):
            G.add_edge(k, i, weight = 0, color = 'r')
if(len(TS2)==2):
    return None
TS_left = TS[left:k]
TS_right = TS[k+1:right+1]
TStoVGfast(TS_left, left, k-1)
TStoVGfast(TS_right, k+1, right)
left1 = 0
right1 = len(y)-1
TS1 = list(y)
TStoVGfast(TS1, left1, right1)
```
Appendix A. Implementation Programs

Following is the Python implementation of the partitioning algorithm mentioned in Chapter 4. In the code the \texttt{quality_func} contains both the cases if $i = j - 1$ or not equal. The variable \texttt{clusters} is a list of lists that consists of all the clusters of the graph.

\textbf{Program A.4} Program for Partitioning algorithm

```python
weight=nx.get_edge_attributes(G,'weight')
mini = []
i_min = []
clusters = []
memo = []
memon = [] # memo numerator
memod = [] # memo denominator
for z in range(0, len(y)):
    memo.append(0)
memon.append(0)
memod.append(0)
mini.append(0)
i_min.append(0)
def quality_func(j2, i2, memon1, memod1):
    nin = 0
    pout = 0
    nout = 0
    pin = 0
    change = 0
    if(i2==j2-1):
        for a in range(0, len(y)):
            if(G.has_edge(a, i2) == True):
                if(a<i2):
                    edgeweight = weight[a, i2]
                else:
                    edgeweight = weight[i2, a]
                if(a==j2):
                    if(edgeweight == 0):
                        memod1 = memod1+1 #nin
                    else:
                        memon1 = memon1+1 #pin
                else:
                    if(edgeweight == 0):
                        memon1 = memon1+1 #nout
```
else:
    memod1 = memod1+1  # pout
for a in range(0, len(y)):
    if(a==i2 and i2==j2-1):
        continue
    if(G.has_edge(a, j2) == True):
        if(a<j2):
            edgeweight = weight[a, j2]
        else:
            edgeweight = weight[j2, a]
        if a in range(i2, j2+1):
            if(edgeweight == 0):
                nin = nin+1
                memon1 = memon1-1
            else:
                pin = pin+1
                memod1 = memod1-1
        else:
            if(edgeweight == 1):
                pout = pout+1
            else:
                nout = nout+1
    memon[i] = pin + nout + memon1
    memod[i] = nin + pout + memod1
    if(memon[i] == 0):
        change = memon[i]
    else:
        change = memon[i]-memod[i]
return change;
for j in range(1, len(y)):
    quality = 0
    for i in range(0, j):
        if(i==j-1):
            quality = quality_func(j, i, 0, 0);
        else:
            quality = quality_func(j, i, memon[i], memod[i]);
        memo[i]=quality
        if(i==0):
            mini[j] = quality
            i_min[j] = i
        if(quality>mini[j]):
            mini[j] = quality
            i_min[j] = i
Appendix A. Implementation Programs

j = len(y)-1 # last node in the graph
while(j>=0):
    i = i_min[j]
    clusters.append(list(range(i, j+1)))
    j = i-1
    if(j==1 and clusters[-1] != [0,1]):
        clusters.append([0,1])