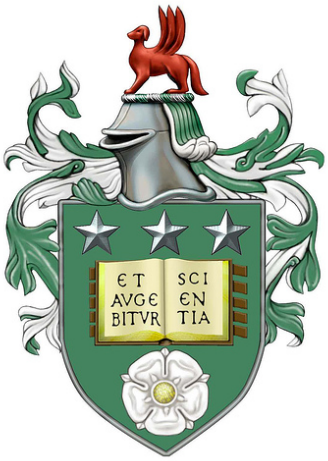
Learning node attributes in graphs with application in social networks



Fatimah A Almulhim

A thesis submitted in accordance with the requirements for the degree of Doctor of Philosophy

University of Leeds

Department of Statistics

2019

The candidate confirms that the work submitted is her own, except where work which has formed part of jointly authored publications has been included. The contribution of the candidate and the other authors to this work has been explicitly indicated below. The candidate confirms that appropriate credit has been given within the thesis where reference has been made to the work of others.

Jointly authored publications:

Almulhim F.A., Thwaites P.A., and Taylor C.C. (2019) A New Approach to Measuring Distances in Dense Graphs. In: Nicosia G., Pardalos P., Giuffrida G., Umeton R., Sciacca V. (eds) Machine Learning, Optimization, and Data Science. LOD 2018. *Lecture Notes in Computer Science*, vol 11331. Springer, Cham

Chapter 3 and part of Chapter 4 are based on this publication.

This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

@2019 The University of Leeds and Fatimah A Almulhim

I would like to dedicate this thesis to my family for never doubting my abilities, even when I doubt myself.

Acknowledgements

All thanks, praise, and glory to Allah the Almighty, who alone, guided me to complete and achieve my goal.

Many thanks to my supervisors Dr. Peter Thwaites and Prof. Charles Taylor for their support, guidance, and patience which helped me to accomplish this work. Both have provided valuable knowledge in Statistics and helpful advice on programming with R and typesetting with LaTeX.

Many thanks to my mother Nawal for her support and prayers in everything in my life. I am also very grateful to my husband Abdullah and my lovely children Abdulaziz and Ziyad for their endless support and faith in my abilities - without whose love and encouragement, I would not have been able to complete this work.

Finally, many thanks to my brothers, sisters, aunts, and friends for their unconditional love throughout my life. This work was financially supported by Princess Nora bint Abdulrahman University.

Abstract

This thesis is initially concerned with measuring distances in undirected and unweighted graphs. The problem of computing distances and shortest paths between nodes in graphs is one of the fundamental issues in graph theory. It is of great importance in many different applications, for example, transportation, and social network analysis. Basically, the majority of dense graphs have ties between the shortest distances. Therefore, we consider a different approach and introduce a new measure to solve all-pairs shortest paths for undirected and unweighted graphs. This measures the distance between any two vertices by combining the length and the number of all possible paths between them. The main aim of this new approach is to break the ties between equal shortest paths (SP), and distinguish meaningfully between these equal distances. Using the new distance measure for clustering produces higher quality results compared with SP over different simulated graphs and real data networks. We also present a modification of the k-means algorithm which is based on a new strategy of selecting the initial centroids. Clustering results improve by using this strategy compared to the classic k-means. We are also interest in classifying and predicting graph node attributes using nonparametric techniques. Therefore, we simulate different graphs from different models, where each node has a label (class) and a real value attribute. In node classification, we adapt current techniques to be suitable for distances between nodes in graphs. We also introduce several new ideas regard choosing the smoothing parameter in the kernel estimator. The simulation results show that using the support vector machine (SVM) with kernel trick produces more accurate classification accuracy rates compared with other classifiers. In node prediction, we apply the Nadaraya-Watson estimator to predict the real value attributes. We investigate the relationship between node features by considering a weight parameter between nodes' classes in the distance matrix. The prediction results vary between different networks due to the differences in the graph size and the node features structure.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Many of our real-life systems consist of objects or collections of objects, which have links between them of different types. These can be described by networks, and the topology of these networks usually depends on the nature of the links and interactions between objects. Some of them have some patterns in the structure, which describe the behaviour of the system, and some of them can be considered as random networks where the links are connected randomly between objects.

Network structures have been studied by mathematicians and represented as graphs, where the nodes are represented by dots and the edges are represented by lines. Early work on graph theory appeared in the eighteenth century and was done by Leonhard Euler in 1736, with the famous bridge problem which is called the *Seven Bridges of Königsberg.* The problem concerns four different regions separated by the Pregel river and connected by seven bridges in Prussia, and the question is how to move between the lands using each bridge once. To solve this problem, Euler represented the problem as a graph where the lands are represented by four nodes and the bridges are represented by seven edges. The problem is impossible unless none or two nodes have an odd number of edges. This problem's solution is considered as the first theorem in graph theory and the basis of other theorems in graph theory and topology (Douglas, 2001). Graph theory has been developed and got its statistical foundation in the twentieth

century. In particular, random graphs have had a lot of interest as they can model most complex networks.

A lot of real networks have received more interest in recent years, for example, the world wide web as one of the largest invisible networks, which reached one billion nodes by 1999 (Albert and Barabási, 2002). In this network, the nodes correspond to the web-pages and the edges correspond to the hyperlinks between these websites.

In addition, there are different biological systems which can be represented by networks and have rich topological structures. One of the most valuable biological networks is the network of metabolic reactions, where the nodes are substrates, for example, H_2O, and the edges describe the chemical reactions between these substrates. Also, protein networks, where the nodes correspond to proteins and the edges describe the interactions between proteins (Dorogovtsev and Mendes, 2002).

Another type of interesting network is the social network where the nodes are people who have interactions between them in different relationships (as edges), for example, family and friend networks in schools or neighbourhoods. The nodes could also be groups of people in companies and the edges represent the business relationships between them. The most common and largest social networks are in social media, for example, Facebook and Twitter, which are growing dramatically over time (Newman, 2010).

Most real networks are considered as complex networks as their topological structures have different features, for example, different types of nodes or different types of edges at the same time. For example in social networks, the nodes could be associated with a variety of features such as gender (men or women), ages, or different nationalities. The edges could represent different links, such as friend relationships, neighbour relationships or many other relationships (Newman, 2003).

The prevalence of data-driven networks in our lives leads to a lot of questions about this data and how we could analyse and understand the structures and properties. The simplest and most common analysis and optimization techniques come from machine learning (ML), which can be considered as the interaction between computer science and statistical analysis. ML can be used to help computers to find a pattern in a data set by using some prior knowledge about

the data from a training set. This can help a researcher to analyse and predict features of new observations and make decisions.

Machine learning tasks of this type are mainly divided into three branches. First, unsupervised learning, where the input data is unlabelled and the goal is to explore and visualize the data structure, for example, clustering. We discuss this branch in Chapter 4. Second, supervised learning where the training data set has labels and the machine builds a model or pattern to predict the labels of the test data set. This has a variety of applications: for example, classification and regression. We discuss this branch in Chapters 5, and 6. Third, semi-supervised learning, which is considered as a class of supervised learning in which the training set has few labelled data but the majority is unlabelled. This is useful in applications where finding data labels is expensive and slow, for example, in web page classification, where there is a huge number of web pages and where the cost of reading and recognizing labels is expensive.

1.2 Motivation and contribution

The first factor which motivates the author to doing this study is that the existing work in graphs and networks is not keeping pace with rapid growth in real world networks, and in particular, social networks. Also, most of the existing research focuses on descriptive statistics of these networks, and on investigation of the existing graph models and their links to real networks. Little work extends to doing inference and analysing the behaviour of the network elements.

Furthermore, most real networks have a unique structure as a result of the heterogeneity of the data. A natural way of discovering this structure is often by grouping similar objects into disjoint subsets based on similar features, which is known as clustering. As the clustering is free from any prior information of how the clustering should be done, it is a common current research area in network analysis especially in social network analysis.

Clustering requires a distance between each pair of objects in order to group the closer objects together. But we notice that there is a dearth of research on the distances between nodes in graphs, where most of the studies are still based on the simple distance which calculates the number of edges between nodes. In dense networks such as Facebook, the corresponding

graph shows large numbers of edges between users. This means that there are different paths connecting most of the network's users, which should indicate the strength of the relationships between users. The strength of a relationship between two users has typically been measured by the distance between them (i.e., smaller distance means stronger relationship between pair of users).

However, existing distance measures are not adequate to distinguish between similar paths' lengths in the case of a variety of paths between two users. This discrete aspect results in lots of ties between shortest paths, which makes the clustering process more complicated. One of the main contributions of this study is to introduce a new distance metric which allows us to break ties between equal distances by taking account of the number of paths between each pair of users.

Further analysis could be done by applying classification and prediction in social networks after doing clustering in these networks. We notice that it is rarely done in this field, as most of the available work in the literature is related to link prediction. We aim instead to predict the node characteristics and real-valued attributes using the information from nearby nodes and the graph topological structure.

In this study, we apply our ideas to examples taken from the Facebook network and the Amazon network, whose nodes are rich in information, which makes the analysis more interesting.

This study has various technical contributions:

- In Chapter 3, we introduce a new distance metric for undirected and unweighted graphs which is based on the adjacency matrix and matrix multiplication. It resolves the limitations of the known algorithms, and has the ability to distinguish between shortest paths of equal length.
- In Chapter 4, we show that our new metric works well for node clustering, and shows an improvement in clustering results compared with the classical shortest path distances.
- In Chapter 4, we introduce an adaptation of the K-means clustering algorithm by considering two ways of choosing the initial centroids in graphs. This produces solutions as

good as the best solution from random centroids and is less time consuming.

- In Chapter 5, we apply some nonparametric techniques in graph node classification, an area with little attention in the literature. For example, in kernel density classification, most of the existing studies focus on classifying point labels in Euclidean space rather than in graphs. So, we apply the kernel classifier to graph nodes, and introduce a new strategy of choosing the smoothing parameter. This strategy is not exclusive to graphs, but also could have different applications in other areas.

- In Chapter 5, we apply an adaptive kernel classifier in the classification of a graph node labels, and introduce a novel way of choosing the smoothing parameter based on the node degree.

- In Chapter 5, we apply the support vector machine technique in the classification of graph node labels. As the nodes do not exist in Euclidean space and do not have coordinates, we use a multidimensional scaling of the distance matrix to represent the nodes' positions in a low-dimensional coordinate system.

- In Chapter 6, we modify the distances between nodes with respect to similar and different classes, and use the modified distances in predicting the node label values by using nonparametric regression. We also introduce a new practical technique to predict the relationships between node features and how these features are related to, and distributed over the graph nodes, based on some combined tests.

1.3 Thesis outline

This thesis is divided into seven chapters. Chapter 2 summarises the main concepts in graphs and networks, and discusses some useful properties in graphs. We also focus on four different graph models that we use in this study, followed by a general description of our Facebook network and Amazon network data.

Chapter 3 introduces an overview of the shortest distance path algorithms in graphs, and motivates our new distance measure in undirected and unweighted graphs, with a description

in detail, followed by a comparison with the existing measures. Chapter 4 focuses on node clustering methods, and discusses the efficiency of the new distance measure on the resulting clusters. We also present a modification of the K-means clustering algorithm by introducing a new way of choosing the initial centroids. The final part of this chapter is a comparison study with simulations to show the differences in the accuracy rate and the time taken in clustering when using our proposed measure and the traditional distance measure.

Chapter 5 introduces the common methods in classification with a goal of node label classification. We also discuss several new ideas in this chapter. We end this chapter by a comparison study between four different classification methods on simulated graphs and real networks. Chapter 6 is concerned with node label prediction, with consideration of the relationships between nodes features. We apply a Nadaraya-Watson type estimator on four different graph examples from the Amazon network and simulated graphs. We also discuss different strategies for distributing features on simulated graphs which result in different structures and relationships between nodes. The prediction results of these different graphs help us to accept or reject several hypotheses about the relationships between nodes' features. Chapter 7 provides the final conclusion and future work discussion.

Chapter 2

Introduction to graphs and networks

2.1 Introduction

In this chapter, we shall briefly introduce the important concepts and terminology in graph theory, which are related to this thesis, and discuss common properties of the graphs. In Section 2.3, we cover different graph models and focus on four different models which we use in this research. In Section 2.4 we talk about real networks and the structural properties of them, with a focus on social networks, followed by a full description of our data: the Facebook network and the Amazon network. All results in this thesis are calculated by using some functions in the `igraph` package in `R` software with our own codes. The `igraph` package is designed for graphs and networks analysis (Csárdi and Nepusz, 2006).

2.2 Concepts and Characterizations

There are many definitions and concepts in graphs and networks analysis, however, in this section, we introduce the most relevant concepts to this study which are taken from Douglas (2001). The following definitions introduce the graph elements.

Definition 2.2.1 *A* **graph** G *is a triple consisting of a* **vertex set** $V(G)$*, an* **edge set** $E(G)$*, and a relation that associates with each edge two vertices (not necessarily distinct) called its*

endpoints.

Definition 2.2.2 *A* **loop** *is an edge whose endpoints are equal.* **Multiple edges** *are edges having the same pair of endpoints. A* **simple graph** *is a graph having no loops or multiple edges. We specify a simple graph by its vertex set and edge set, treating the edge set as a set of unordered pairs of vertices and writing $e = uv$ (or $e = vu$) for an edge e with endpoints u and v. When u and v are the endpoints of an edge, they are* **adjacent** *and* **neighbours**.

The next three definitions describe the structure settings.

Definition 2.2.3 *The* **complement** $\bar{G}$ *of a simple graph G is a simple graph with vertex set $V(G)$ defined by $uv \in E(\bar{G})$ if and only if $uv \notin E(G)$. A* **clique** *in a graph is a set of pairwise adjacent vertices. An* **independent set** *(or* **stable set***) in a graph is a set of pairwise non-adjacent vertices.*

Definition 2.2.4 *A* **path** *is a simple graph whose vertices can be ordered so that two vertices are adjacent if and only if they are consecutive in the list. A* **cycle** *is a graph with an equal number of vertices and edges, whose vertices can be placed around a circle so that two vertices are adjacent if and only if they appear consecutively along the circle.*

Definition 2.2.5 *A* **subgraph** *of a graph G is a graph H such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$ and the assignment of endpoints to edges in H is the same as in G. We then write $H \subseteq G$ and say that "G* **contains** *H". A graph G is connected if each pair of vertices in G belongs to a path; otherwise, G is* **disconnected**.

The following two concepts introduce the graph matrices which represent the graph mathematically.

Definition 2.2.6 *Let G be a loop-less graph with vertex set $V(G) = v_1, ..., v_n$, and edge set $E(G) = e_1, ..., e_m$. The* **adjacency matrix** *of G, written $A(G)$, is the n-by-n matrix in which entry a_{ij} is the number of edges in G with endpoints v_i,v_j. The* **incidence matrix** *$M(G)$ is the n-by-m matrix in which entry $m_{i,j}$ is 1 if v_i is an endpoint of e_j and otherwise is 0. If vertex v is an endpoint of edge e, then v and e are* **incident**.

Remark 2.2.1 *An adjacency matrix is determined by a vertex ordering. Every adjacency matrix is* **symmetric** *($a_{ij} = a_{ji}$ for all i, j). An adjacency matrix of a simple graph G has entries 0 or 1, with 0s on the diagonal.*

The following part gives more descriptions of the connections in graphs.

Definition 2.2.7 *A* **walk** *is a list $v_0, e_1, v_1, \ldots, e_k, v_k$ of vertices and edges such that, for $1 \leqslant i \leqslant k$, the edge e_i has endpoints v_{i-1} and v_i. A* **trail** *is a walk with no repeated edge. A $(u, v)-$walk or $(u, v)-$trail has first vertex u and last vertex v; these are its* **endpoints**. *A $(u, v)-$* **path** *is a path whose vertices of degree 1 (its* **endpoints***) are u and v; the others are* ***internal vertices***. *The* **length** *of a walk, trail, path, or cycle is its number of edges. A walk or trail is* **closed** *if its endpoints are the same.*

Definition 2.2.8 *A graph G is* **connected** *if it has a (u, v)-path whenever $u, v \in V(G)$ (otherwise, G is* **disconnected***). If G has a (u, v)-path, then u is* **connected to** *v in G. The* **connection relation** *on $V(G)$ consists of the ordered pairs (u, v) such that u is connected to v.*

Definition 2.2.9 *The* **components** *of a graph G are the maximal connected sub-graphs. A component (or graph) is* **trivial** *if it has no edges; otherwise it is* **non-trivial**. *An* **isolated vertex** *is a vertex of degree 0.*

Proposition 2.2.1 *Every graph with n vertices and k edges has at least $n - k$ components.*

More concepts describe the graph parameters such as the vertex degree and counting the edges in a graph.

Definition 2.2.10 *The* **degree** *of vertex v in a graph G, written $d_G(v)$ or $d(v)$, is the number of edges incident to v, except that each loop at v counts twice. It is also calculated by the sum of the entries in the row for v in either $A(G)$ or $M(G)$. The maximum degree is denoted by $\Delta(G)$, the minimum degree is denoted by $\delta(G)$, and G is* **regular** *if $\Delta(G) = \delta(G)$. It is k-***regular** *if the common degree is k. The* **neighbourhood** *of v, written $N_G(v)$ or $N(v)$, is the set of vertices adjacent to v.*

Definition 2.2.11 *The* **order** *of a graph G, written $n(G)$, is the number of vertices in G. An n-***vertex graph** *is a graph of order n. The* **size** *of a graph G, written $e(G)$, is the number of edges in G.*

Proposition 2.2.2 (Degree-Sum Formula) *If G is a graph, then $\sum_{v \in V(G)} d(v) = 2e(G)$.*

Proof: Summing the degrees counts each edge twice, since each edge has two ends and contributes to the degree at each endpoint.

Some basic properties and distance of tree graph models are contained in the following definitions and theorem;

Definition 2.2.12 *A graph with no cycle (no closed walk) is* **acyclic**. *A* **tree** *is a connected acyclic graph. A* **leaf** *(or* **pendant vertex***) is a vertex of degree 1.*

Theorem 2.2.1 *For an n-vertex graph G (with $n \geq 1$), the following are equivalent (and characterize the trees with n vertices).*

1. *G is connected and has no cycle.*
2. *G is connected and has $n - 1$ edges.*
3. *G has $n - 1$ edges and no cycles.*
4. *G has no loops and has, for each $u, v \in V(G)$, exactly one (u, v)-path*

The proof of this theorem is in Douglas (2001).

Definition 2.2.13 *If G has a $u, v-$path, then the* **distance** *from u to v, written $d_G(u, v)$ or simply $d(u, v)$, is the minimum length of a (u, v)-path which is also called the geodesic distance. If G has no such path, then $d(u, v) = \infty$. The distances between nodes in a graph are represented by distance matrix D where the matrix entries d_{ij} correspond to the distances between nodes v_i, v_j. The* **diameter** *(diam G) is $max_{u,v \in V(G)} d(u, v)$.*

An important theorem in graph theory that we use in our new distance metric in Chapter 3 is:

Theorem 2.2.2 *If A is the adjacency matrix of a graph G with $V(G) = \{v_1, v_2, \ldots, v_n\}$, then the (i, j) entry of $A^k, k \geq 1$, is the number of different $v_i - v_j$ walks of length k in G.*

(Chartrand et al., 2011)

In this study, we consider graphs with undirected edges, so we shall introduce the concept of directed graphs in the following definition.

Definition 2.2.14 *A* **directed graph** *or* **digraph** *G is a triple consisting of a* **vertex set** *$V(G)$, an* **edge set** *$E(G)$, and a function assigning each edge an ordered pair of vertices. The first vertex of the ordered pair is the* **tail** *of the edge, and the second is the* **head***; together, they are the* **endpoints***. We say that an edge is an edge* **from** *its tail* **to** *its head.*

2.2.1 Common measurements in graphs

An important and principal step in network analysis is discovering the network topology, by measuring some useful quantities which help to capture the network features and patterns. In this section, we introduce some common measures which are used in this study, with a brief summary about them. They are taken from Newman (2003); Kolaczyk and Csárdi (2014); Wasserman and Faust (1994); Newman (2010).

Centrality, closeness, and betweenness

The node centrality is measured by the node degree (the number of edges that connect it to the rest of the nodes), which reflects the importance of a node. For example, in social networks, if a node has high degree in a graph, this means it has more connections to its neighbours, and could be considered as a more influential node than small degree nodes.

Another measure of the node centrality is the closeness centrality, which measures the average distance from each node to the other nodes in the graph. Let $d(v_i, v_j)$ be the geodesic

distance between nodes v_i and v_j in graph G of size n, then the closeness of node v_i is

$$\ell_i = \frac{1}{n-1} \sum_{i \neq j}^{n} d(v_i, v_j).$$

Betweenness centrality is an important concept in graph theory, which measures the amount of information flowing through a node in the graph. In another way, it measures the probability of a node to be in a path in a graph. Mathematically, let n_{st}^i be the number of shortest paths between nodes v_s and v_t which pass through node v_i, and b_{st} be the total number of shortest paths between nodes v_s and v_t. The betweenness centrality of node v_i is

$$x_i = \sum_{s,t=1}^{n} \frac{n_{st}^i}{b_{st}}.$$

Graph density

Graph density measures the density of the edges over the graph nodes and the extent to which the graph resembles a clique. It is determined by the proportion of the actual edges in the graph to the maximum number of the possible edges

$$\text{den}(G) = \frac{2|E(G)|}{|V(G)|(|V(G)|-1)}.$$

It has a range from 0 in the case of no edges in the graph, to 1 in the case of a fully connected graph (or clique). In directed graphs, the denominator is replaced by $2|V(G)|(|V(G)|-1)$.

Transitivity

One of the most important properties of networks topology is the transitivity $C|_T$, which is also called the clustering coefficient. This quantity measures the density of triangles (three nodes connect by three edges) in a graph G. It has the form

$$C|_T(G) = \frac{3\tau_\triangle(G)}{\tau_3(G)}, \tag{2.1}$$

where $\tau_{\triangle}(G)$ is the number of triangles in the graph G, and $\tau_3(G)$ the number of connected triples which means a single vertex connected with pair of vertices by pair of edges. The factor 3 in the numerator is due to the fact that any triangle has 3 different connected triples of vertices, and so $C|_T$ is in the range $[0, 1]$.

Social networks usually have high transitivity as the friend of a friend is also likely to be a friend. In sociology literature, transitivity is usually referred to as "the fraction of transitive triples ". Watts and Strogatz (1998) introduced the local value of transitivity, which is calculated for each node in the graph as

$$c|(v) = \frac{3\tau_{\triangle}(v)}{\tau_3(v)},$$

where $\tau_{\triangle}(v)$ is the number of triangles in G which contains node v, and $\tau_3(v)$ is the number of connected triples where the degree of v is greater or equal to two. If v has degree 0 or 1, $c|(v) = 0$.

Mean geodesic distance

The mean of the geodesic distances ℓ between all vertices in a graph is defined by

$$\ell = \frac{1}{\frac{1}{2}n(n+1)} \sum_{i \geqslant j} d_{ij},$$

where $\frac{1}{2}n(n+1)$ is the number of upper (or lower) values in the distance matrix, and d_{ij} is the geodesic distance between v_i, and v_j.

Small-world effect

The small-world effect is a property of networks which have dense connections between their vertices. Most pairs of vertices are connected by a short path through the graph, which is known as the small world effect. This phenomenon appears in most real-world published networks, and can be checked by measuring the average of geodesic distances between nodes.

Newman (2003) summarized the mean geodesic distance ℓ of different social, information, technological and biological networks, and among 27 networks where their number of vertices

range between hundreds to millions, the ℓ measurement did not exceed 5 for more than half networks. Moreover, small values of ℓ could have a direct effect on the dynamics of processes on the networks. For example, spreading of any information or news is much faster in networks with small world effect than any other networks. As in social networks, for example, rumors spread faster with a maximum of six steps between any pair of people than if it passes more than hundred steps.

Degree distributions

The degree distribution of a network is the degree distribution of the nodes in the network. This is one of the most common properties in graph analysis. It shows the spread of the degrees over the graph nodes and gives the probability of a degree k for any randomly selected node.

2.3 Graph Models

In this section, we give an overview of some common random graph models in graph theory that we use in our simulations and experiments in the rest of the chapters. There are a huge number of models; too many to cover in this study, so we choose models which have a variety of different structures, which make the analysis and comparison more compelling. The most common models are the classic random graph, small world model, and growing network models. The descriptions are taken from Aggarwal and Wang (2010); Kolaczyk and Csárdi (2014).

2.3.1 Classical Random Graphs

Erdős and Rényi (1959) introduced a simple undirected graph model which is called the Erdős-Rényi graph model (ER model) and has two distinct features: first, it has a fixed number of nodes n, and the second is that it has a random distribution of edges over the graph nodes with a probability p, so the links between nodes are independent from any node features.

For n nodes, there are $\binom{n}{2} = n(n-1)/2$ node-pairs. A graph $G(n, e)$ will have some node

pairs joined by edges with probability p, and some not joined by edges with probability $1-p$. So the probability of assigning e edges between specified node pair (out of $E = \binom{n}{2}$), and hence producing a particular graph $G(n, e)$, is

$$P(G) = p^e(1-p)^{E-e},$$

Erdős and Rényi found that, given this graph type, the degree distribution of the graph nodes is

$$P(\text{degree}(v_i) = k) = \binom{n-1}{k} p^k (1-p)^{n-1-k},$$

This is bin$(n-1, k)$, so the average node degree is $\bar{k} = (n-1)p$. The binomial distribution of the node degrees can be approximated with Poisson distribution for large n (because $(n-1 \approx n)$) and small p, and np is constant, which has a formula

$$P(k) = \frac{(np)^k e^{-np}}{k!}.$$

The graph model structure depends on the value of p, the model produces small components (connected graph) with a small probability p, and high density large components with high probability p values. If $p = 1$, the transitivity equals 1, so have a complete graph (fully connected).

More details can be found in Dorogovtsev and Mendes (2002); Albert and Barabási (2002); Newman (2003, 2010).

We can simulate a graph from the ER model by using `erdos.renyi.game` function in `igraph` package in `R` (Csárdi and Nepusz, 2006). Figure 2.1 shows three graphs simulated from the ER model with different p values, which controls the density of the edges over the graph nodes.

2.3.2 Small-World Model

This model was introduced by Watts and Strogatz (1998), when they observed one of the major features of large real networks, that they combine small average geodesic distance between

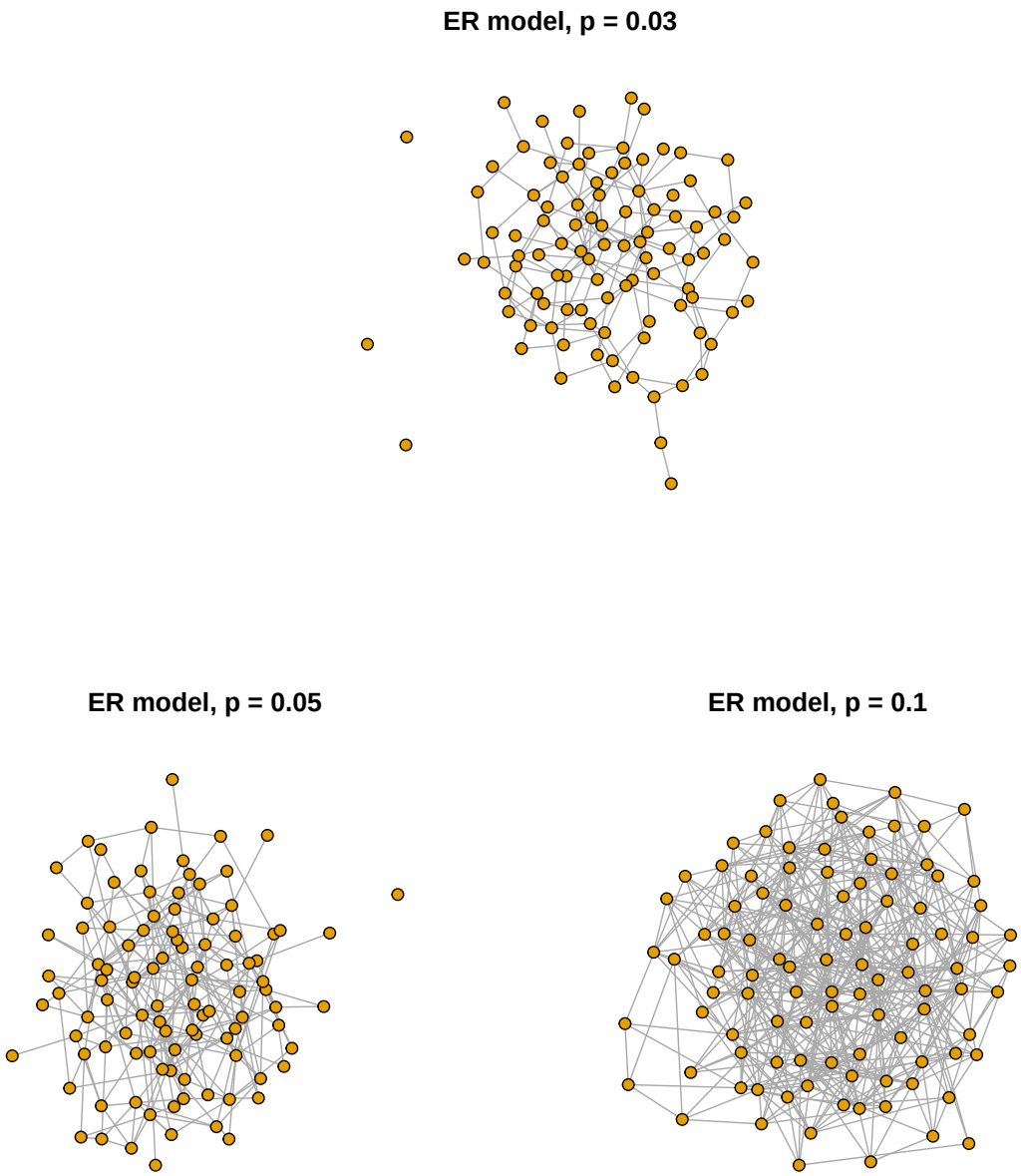


Figure 2.1: Plots of three simulated graphs from the ER model with different probability values, $p = 0.03, 0.05, 0.1$, and $n = 100$.

nodes with large clustering coefficient, compared to random graphs. Their model starts with any regular graph with n nodes, for example, a ring, where each node connects with k neighbours as in Figure 2.2 (left), which has $nk/2$ edges. Then modify this graph by introducing a small probability p for rewiring some edges, by moving one end of randomly chosen edges to join another nodes, or add extra edges. This should exclude loops and multiple edges. If p is chosen to be small, the small world model appears as in Figure 2.2 (centre). By increasing the value of p, the model is random graph.

Watts and Strogatz concluded that there is a critical region in between these limits as shown in Figure 2.3. We can see that for small values of p, the transitivity dose not change much with large average geodesic distance. While for large values of p the average geodesic distance and transitivity are decreased. So there is a region in the middle which achieves high transitivity and small geodesic distance (approximately in the range for $p \in [10^{-2}, 10^{-1}]$ (Newman, 2003; Dorogovtsev and Mendes, 2002; Albert and Barabási, 2002).

The `igraph` package can be used to produce graphs from the Watts and Strogatz model by using the function `watts.strogatz.game`(Csárdi and Nepusz, 2006).

Figure 2.4 shows different graph structures of Watts and Strogatz model with different values of p. As p increases from 0.05 to 0.1 and 0.2, the number of triangles in the graph decreases compared with the number of connected triples of nodes, which means less transitivity in the graph. The transitivity of the graphs is $0.4, 0.25, 0.1$ respectively. Also, the average of the geodesic distances decreases by increasing p, which has values $5.5, 4.2, 3.6$ respectively.

2.3.3 Models of Network Growth (Scale-free Networks)

Most real world networks have a variation in the node degree, and have a heavy tailed degree distribution. It usually follows the power law distribution $P(k) = ck^{-\gamma}$, where γ is a power law exponent, and c is a normalization constant.We know that the edges in random graphs are randomly distributed, and the nodes have similar degrees with small variation which make the random graphs inappropriate to model real world networks.

For this reason, Barabási and Albert (1999) generalized the random graph models by

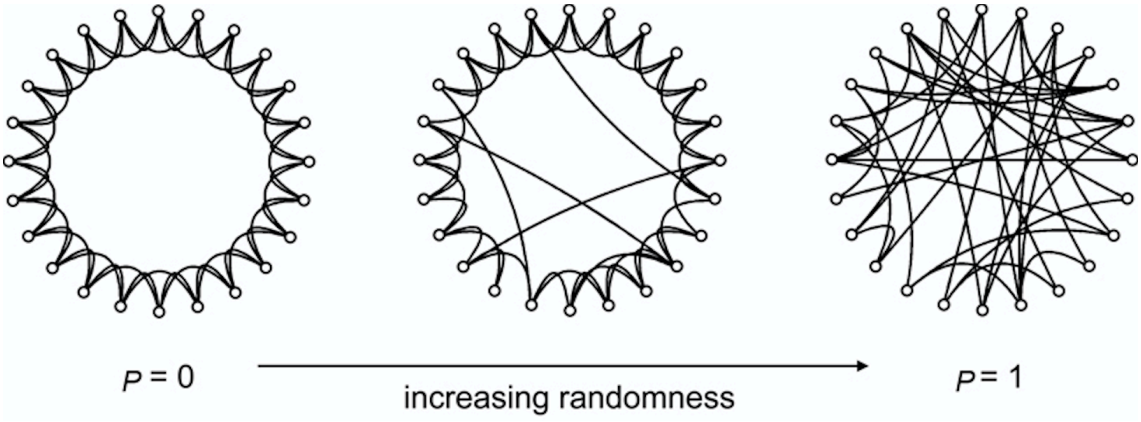


Figure 2.2: Plots show the behaviour of different probability value p, there are three graphs showing the effect of p from 0 which produces a regular graph to 1 which produces a random graph (Watts and Strogatz, 1998).

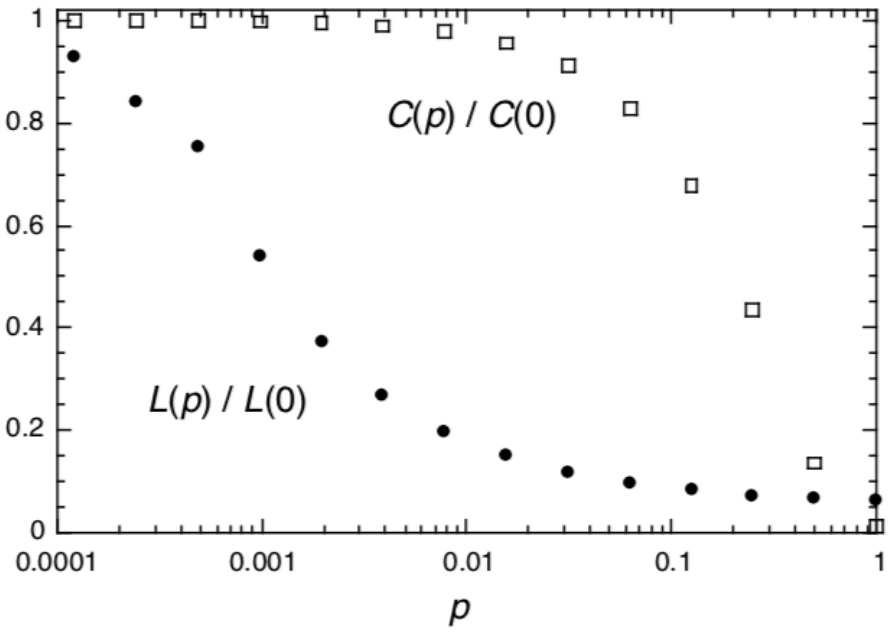


Figure 2.3: Plot shows two curves of the effect of p on the transitivity C, and the average geodesic distance L (Watts and Strogatz, 1998).

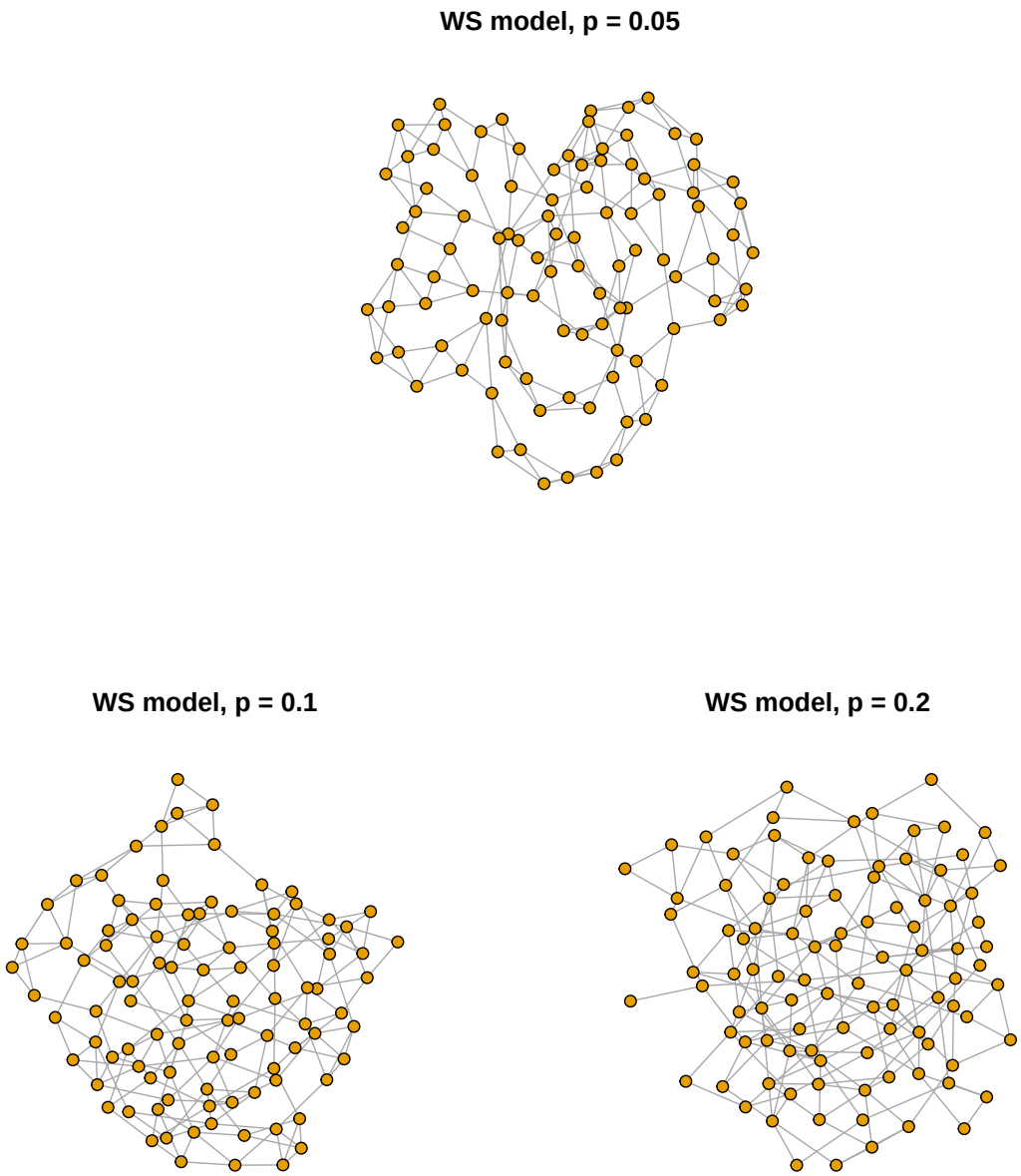


Figure 2.4: Plots of three simulated graphs from the WS model with different probability values, $p = 0.05, 0.1, 0.2$, and $n = 100$.

adding a restriction on the degree distribution to follow the power law distribution. They also called this model the scale-free network model, as the power law is free from characteristic scale. This means that, there is no scale limit on the node degrees, which allows hubs to appear in this model. Hubs are a small number of nodes whose degree are a lot higher than most of the graph nodes. The existence of hubs is one of the most common features in the scale free model compared with the random graph model. The hubs' degrees in the scale free model always exceed the highest degree in the random graphs (Newman, 2003; Albert and Barabási, 2002).

In this model, the networks grow through time by adding vertices and edges. Unlike the random graphs which have a fixed number of nodes. The growing process is similar to the real network dynamic, as the addition of the new edges is usually preferred between pairs of nodes which have a third node in between. In the social networks language this means that there is a high probability to be a neighbour of your neighbour's neighbours than unknown neighbours. In mathematics, this means increasing the transitivity of the network as the number of completed triangles increases as well.

The first model in growing networks was introduced by Price (1965), when he studied the network of citations between scientific papers. He showed that both out- and in- degree distributions (how many times paper A cites other papers and how many other papers cite paper A) follow the power law distribution which is a right skewed distribution with heavy tail.

His idea is based on a graph with a small set of nodes m_0, to which is added a new node every time step with m edges where $m \leqslant m_0$, which links this node with the old nodes with probability proportional to the existing (old) nodes degrees. In citation networks, the probability of a paper getting cited should be proportional to the number of its citations. Price called it a cumulative advantage method.

The values of m and m_0 determine the topology structure of the model. Choosing $m = m_0 = 1$ leads to the tree model structure which we consider in this study, while choosing m and $m_0 > 1$ lead to a nested graph structure.

Although Price's model proposed an explanation of network growth by the power-law degree distribution in citation networks and different real-world networks, it still was not well known until it got updated by Barabási and Albert (1999) who called it preferential attach-

ment. The updated model has the same idea of adding new nodes with probability proportional to the old node degree. So if $\Pi(i)$ is the probability of a new vertex joining vertex v_i, $\Pi(i) = k_i / \sum_j k_j$ where k_j is the degree of vertex v_j. This means that high degree nodes have higher probability of getting links than low degree nodes.

The model is undirected which means that there is no difference between in- and out-degree in the model. The simple version of this model is a tree which just adds one edge with every new node. More generalizations of Barabási and Albert (BA) model were introduced by researchers and can be found in Fazekas and Pecsora (2015).

In the `igraph` package in `R`, there are different functions to produce graphs from the BA Model, for example, `barabasi.game` and `aging.prefatt.game`. Figure 2.5 shows three BA graphs which describe the effect of different power values on the graph structures.

Further details on the mathematics of these models can be found in Newman (2003), Albert and Barabási (2002), and Dorogovtsev and Mendes (2002).

Leskovec et al. (2005) introduced a new growing network model called the forest fire model, which mimics the behaviour of fire spreading between trees in a forest. It is more dense than preferential attachment with no explicit communities, and its diameter decreases as the network grows, as shown by example in Figure 2.6.

The forest fire model is considered as a directed graph model and has two parameters: the forward burning probability which controls the in-degree distribution, and the backward burning probability which controls the out-degree distribution. Both probabilities control the density of the graph. However, in this study, we ignore the direction of the edges, so we use it as an undirected graph.

The noticeable thing is that by increasing the graph model density, it looks like a random graph model (ER model), however, by decreasing the graph model density, the model becomes close to the BA model with tree structure. This can be seen in Figure 2.6. The `igraph` package can be used to produce graphs from FF model by using the function `forest.fire.game`.

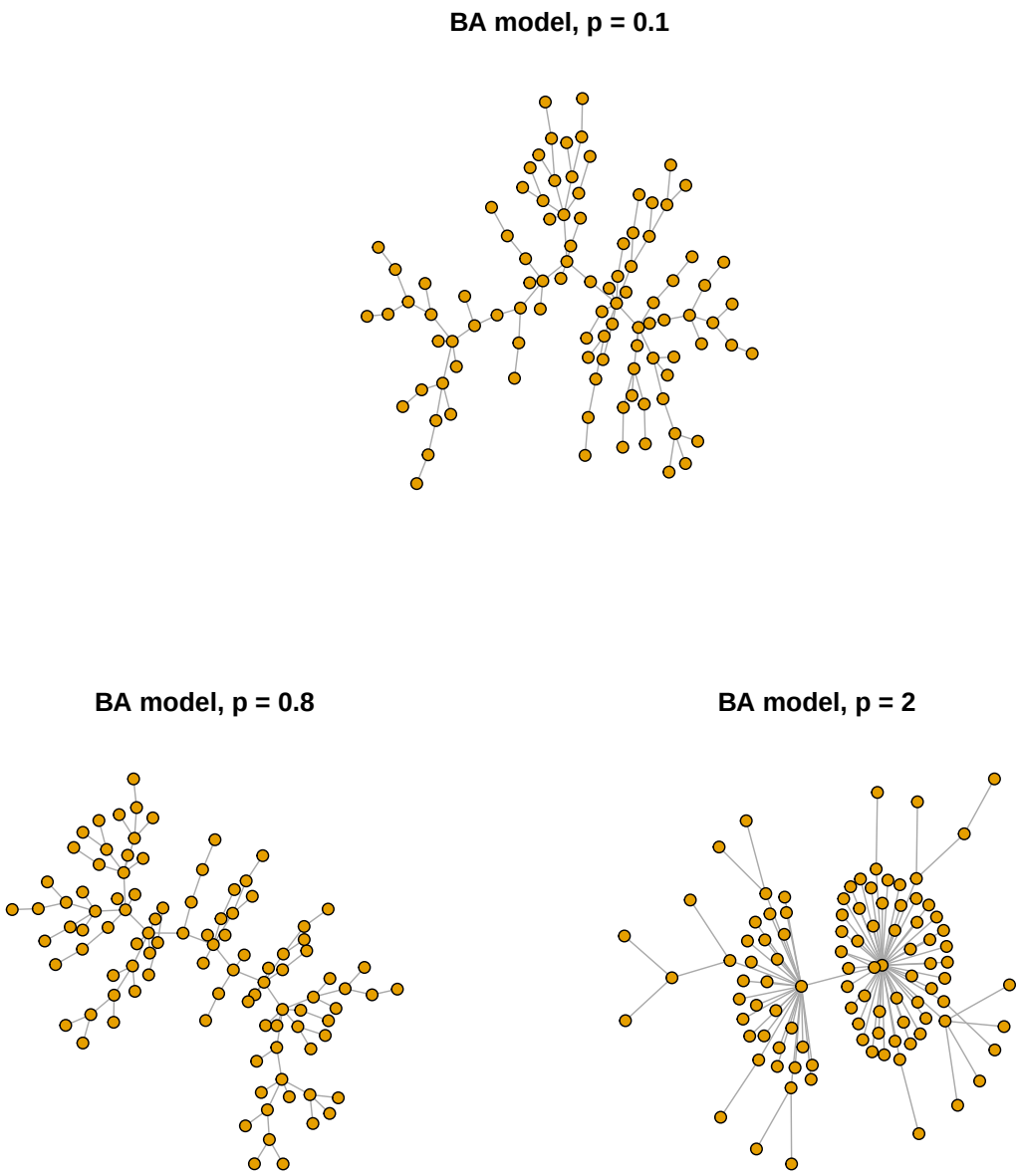


Figure 2.5: Plots of three simulated graphs from the BA model with different power values, $p = 0.1, 0.8, 2$, where p is the power of the preferential attachment probability $\Pi(k) \sim k^p$, and $n = 100$.

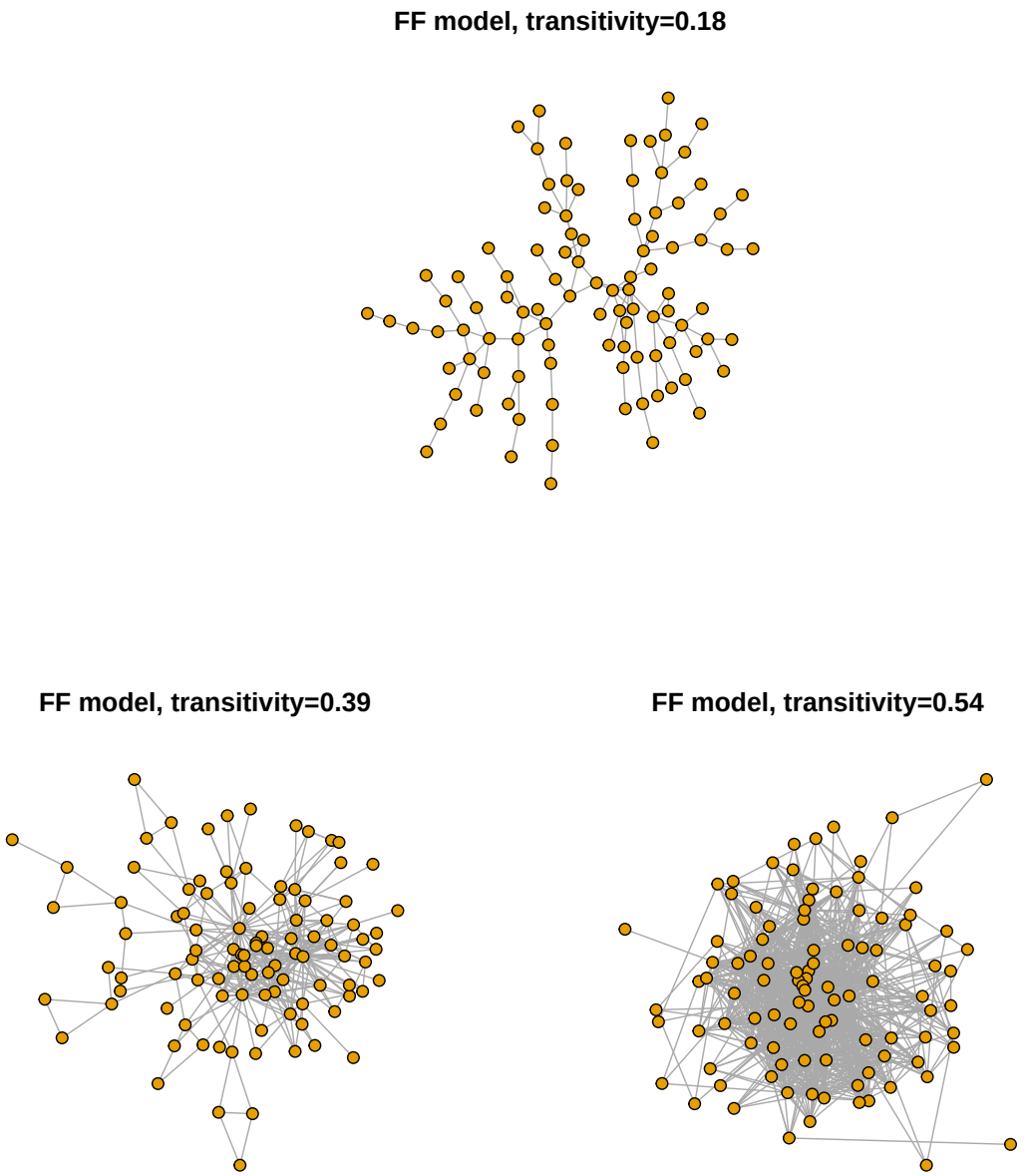


Figure 2.6: Plots of three simulated graphs from the FF model with different transitivity $0.18, 0.39, 0.54$, and $n = 100$.

2.4 Real world Networks - social networks

2.4.1 Introduction

Most real world networks are considered as dynamic networks as they grow and increase with time. The growing stages could consider new links or change in the existing links between nodes. One of the distinguishing properties of real world networks is, that the nodes do not just have relationships between them, but also could have different features. For example, in social networks, the nodes represent people who could be described by gender, age, or level of education. This results in a network's structure with a pattern which depends on these relationships and node features.

Detecting the pattern of the relationships and the network topology could be difficult as the network grows and changes with respect to time.

To study this type of network, we could have a "screen shot" at a specific time, and deal with it as a static network (which we consider in this study). The other possible way is analysing the network with respect to time as a connection between time series and network models.

Most recent work on social networks describes the topology structure and the behaviour of the nodes in the networks, and how the links propagate between nodes. Leskovec et al. (2008) studied the statistical properties of the communities in some large social networks, the community means a set of nodes which have links between them more than with the rest of the network's nodes. McAuley and Leskovec (2014) introduced an automatic tool to detect social circles in ego networks (Facebook, Twitter, and Google+), and identify the similar features between the users in the same circle (circles in social networks usually refer to group of users or friends sharing similar features). Leskovec et al. (2007) analysed the Amazon recommendation system which was available to four million customers and made about 16 million recommendations on more than half a million products. They described how the customers' behaviours changed and affected by their community behaviours, and how this network grows over time. Kim and Leskovec (2011) introduced a method of predicting the missing nodes and edges in real networks. Their method is based on fitting a model of the network structure based on the

known nodes and edges, then predicting the missing data using the fitted model.

In this study, we focus on two different social networks, the Facebook network and the Amazon network, giving a brief description of each in the next sections, and suggesting some machine learning techniques which we discuss it in later chapters.

2.4.2 Facebook Network

We decided to use the Facebook Network from the Stanford datasets website (Leskovec and Krevl, 2014) to analyse and discover the latent structure based on statistical analysis. The data were downloaded from http://snap.stanford.edu/data/ego-Facebook.html. Before starting the analysis process, it is useful to give a simple description of the Facebook Network. Facebook data was collected from a survey of ten users using the Facebook application. They were asked to identify (manually) all the circles to which their friends belonged. The average number of circles in each ego-network is 19 circles, with an average circle size of 22 friends. (McAuley and Leskovec, 2014)

The collected data has been anonymized by replacing the users' ID with a new value. Moreover, the features data are also obscured and replaced by distinct values, for example, for the feature (gender = male), the new value would be (gender = anonymized feature 1). Therefore, we focus on determining whether two users have the same gender without any information about their actual gender.

The Facebook Network is a combination of 10 ego-networks which consist of 4,039 users (nodes), 88234 edges and 193 circles (circle in this context means cycle). The ego user has edges to all nodes in his network. Each ego-network has five different files which describe the components of these sub-networks :

1. The *edges* file has a list of two columns of node ID, and each row represents an edge between two nodes.

   Note that the ego user is omitted from the data, so the links between the ego users and their network members disappear which results in some isolated vertices in the networks.

2. The *circles* file, represents each circle in a separate line, starting with circle number followed by node ID.

3. The *feat* file contains all node features anonymously coded by 0 or 1. Each line contains different features for a specific node. For example, birthday, education degree, education year, gender, language, and lots of different features.

4. The *egofeat* displays the features of the ego user separately from other nodes' features.

5. The *featnames* file which clarifies the names of each of the feature dimensions, if the feature has value 1 in node feature that means this node has this property and 0 otherwise.

We notice from the *feat* file that some features hava more than one line as shown in a screen shot in Figure 2.7, without any more details about them. The "gender" feature is the only feature which has two lines (one indicates to male and the other female). However, in the nodes' features file, most of the nodes have zero nomination for most of the features as shown in a screen shot in Figure 2.8. Therefore, we focus in our analysis on the gender feature because it is binary.

It is noticeable that the Facebook network is a complex and large network. Figure 2.9 shows a representation of the first ego of Facebook network. The graph layout function is fruchterman.reingold which plots the graph in two-dimensions with different edge lengths and less overlaps. The edge length is determined by the layout for visualization purposes.

2.4.3 Amazon network

The Amazon network data was collected by crawling Amazon website in summer 2006. It is based on Customers who bought this item also bought feature of the Amazon website. It is also called person-to-person recommendation network, as it constructed by running a recommendation program between customers. Each customer purchases an item has the option of sending emails recommending this item to friends. The first person who purchases the same item through a referral link in the email will get a 10% discount. The sender also will receive a 10% credit on the next purchase.

```
135 first_name ; anonymized feature 1075
136 first_name ; anonymized feature 1076
137 first_name ; anonymized feature 1077
138 first_name ; anonymized feature 1078
139 first_name ; anonymized feature 1079
140 first_name ; anonymized feature 1080
141 first_name ; anonymized feature 1081
142 first_name ; anonymized feature 1082
143 first_name ; anonymized feature 900
144 first_name ; anonymized feature 1083
145 first_name ; anonymized feature 1084
146 first_name ; anonymized feature 1085
147 gender ; anonymized feature 77
148 gender ; anonymized feature 78
149 hometown ; id ; anonymized feature 129
150 hometown ; id ; anonymized feature 1086
151 hometown ; id ; anonymized feature 1087
152 hometown ; id ; anonymized feature 280
153 hometown ; id ; anonymized feature 1088
154 hometown ; id ; anonymized feature 935
155 hometown ; id ; anonymized feature 1089
156 hometown ; id ; anonymized feature 176
157 hometown ; id ; anonymized feature 1090
```

Figure 2.7: A screen shot from the *featnames* file showing some features (lines 135-157). The anonymized feature numbers refer to the column number in the users *feat* file.

```
2662 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 1 0
0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
```

Figure 2.8: A screen shot of the nodes' features file which shows the features of node number 2662.

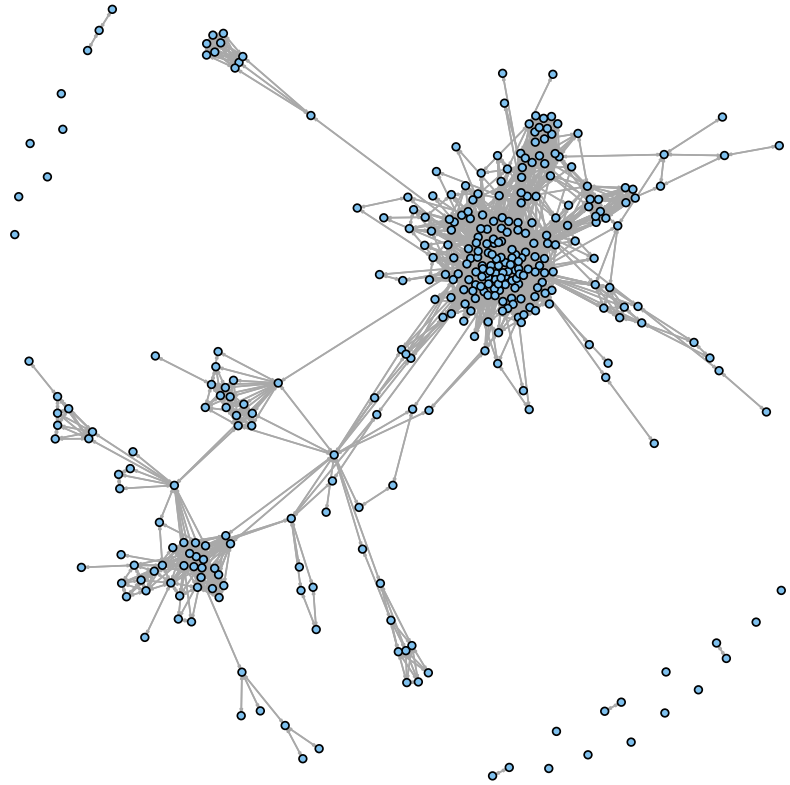


Figure 2.9: Visualization of the first ego-network of Facebook. It has 348 nodes, and $5,038$ edges. Note that the ego user is omitted.

The total number of recommendations reached $15\,646\,121$ which made among $3\,943\,084$ users. Also, the retailer's web site provides some information for all recommended items, for example, categories, reviews, and ratings. In total, there were $548\,552$ different recommended items, each item corresponds to one product type (book, DVD, video, music, toy, video games, software, baby product, CD and sports). The major common products are from books, music, DVD and video respectively. (Leskovec and Krevl, 2014; Leskovec et al., 2007)

For each item, there is a block example of the information as presented in Figure 2.10 :

```
Id:   10
ASIN: 0375709363
  title: The Edward Said Reader
  group: Book
  salesrank: 220379
  similar: 5  039474067X  0679730672  0679750541  1400030668  0896086704
  categories: 3
   |Books[283155]|Subjects[1000]|Literature & Fiction[17]|History & Criticism[10204]|
Criticism & Theory[10207]|General[10213]
   |Books[283155]|Subjects[1000]|Nonfiction[53]|Politics[11079]|History & Theory[11086]
   |Books[283155]|Subjects[1000]|Nonfiction[53]|Social Sciences[11232]|
Anthropology[11233]|Cultural[11235]
  reviews: total: 6  downloaded: 6  avg rating: 4
 2000-10-8  customer: A2RI73IFW2GWU1  rating: 4  votes:  12  helpful:   7
 2001-5-4  customer : A1GE54WF2WUZ2X  rating: 5  votes:  11  helpful:   8
 2001-8-27  customer: A36S399V1VC4DR  rating: 4  votes:   5  helpful:   3
 2002-1-26  customer: A280GY5UVUS2QH  rating: 3  votes:  12  helpful:   7
 2004-4-7  customer: A2YHZJIU4L4IOI  rating: 4  votes:  10  helpful:   2
 2004-4-27  customer: A1MB83EO48TRSC  rating: 4  votes:   5  helpful:   3
```

Figure 2.10: A screen shot from the Amazon-meta text file, which shows a block example of an item's features.

The "Id" is the product number in the file, "ASIN" is the Amazon standard identification number which consists of 10 numbers assigned by Amazon and its partners to identify any product in Amazon, "title" is title of the product, and "sales rank" is based on the recent and historical sales of each product in Amazon. All this information is unique for each item. Also, there is other information like "similar" items which are the ASIN of co-purchased items, "categories" indicate the groups of categories and the subcategories that the product belongs to separated by square brackets. "Reviews" include the product review information, for example, the total number of "reviews" and "downloads', the time of each review, the "customer" ID, the customer "rating" of the product, the total "votes" on each review, the total number of "helpful" votes on each product, and the average rating of all ratings rounded to the nearest half.

In our study we focus on similar items (the co-purchased products), group and average rating ("avg rating") of each product. In this example, the item's "ASIN" number is 0375709363

(let us call it x) was purchased with each of the similar items. This may be by the same customer or more likely to be by different customers. Also, this does not mean that the similar items of x were bought together with item x. If one of x's similar items appears as new item y with a full information block, then we can find that item x appears in the "similar" items of y. In other words, "similar" is not symmetric in the data.

In order to construct a network from this data, we pick up the ASIN entries with similar entries then join them to produce the adjacency list, so the nodes correspond to the items and the edges present the co-purchases between products. In the above example, item x has 5 edges connected with 5 similar items.

In this network, there are $548,552$ products belonging to different groups and we can classify them into three classes. The first class has $169,459$ products which just have an ID and ASIN numbers without any more information about the products, so we exclude them from the network as they considered as missing items. The second class has $83,701$ products which are considered as isolated products in the network because they were purchased alone and do not have any similar items. The third class has $295,392$ products with full information, and we construct our network and graphs based on this class of products and their relations. Note that the products from the third class have connections (co-purchased products) with known and some unknown products. This means that some of the co-purchased products are already from the third class of products and some of them do not exist. Therefore, in this study we focus on the known products and their information.

We refer to the groups by numbers as given in Table 2.1, and the groups' frequencies for the known nodes are in Table 2.2. We start to analyse the connected components by

Group	Baby products	Books	CD	DVD	Music	Software	Sports	Toys	Video	Video games
Symbol	1	2	3	4	5	6	7	8	9	10

Table 2.1: The symbols of product groups.

using the `clusters` function in `R`. The connected components have a variety of group labels among their nodes. We found that most of the components usually have books gathered in one component without any relation with other labels, or we could find occasional items from other

Group	frequency
Video games(10)	1
Books(2)	218551
DVD(4)	13558
Music(5)	49979
Software(6)	1
Toys(8)	4
Video(9)	13298
Total	295392

Table 2.2: The frequency of product groups.

groups in the network dominated by books as shown in Figures 2.11 and 2.12.

However, some other connected components usually gather DVDs and videos as common nodes followed by music then books as in Figure 2.13. Although most of the large connected components have a tree structure, there are a few small connected components with some triangles as in the graph structure in Figure 2.14.

Moreover, the original Amazon data network which contains the known and unknown products has a lot of dense connected components, as in Figure 2.15. Most of them consist of some known products and more than half of the products are unknown. These unknown products appeared frequently in the similar products of some known products as they were co-purchased with them. It would be interesting to analyse graphs such as Figure 2.15, however, the lack of information makes the analysis impossible.

In our study, we choose four different examples of connected components which have known items and cover most of the groups. We do our analysis and tests on them in the later chapters.

We also have the *avg rating* and frequencies for the known nodes of the Amazon network data in Table 2.3, and the *avg rating* frequencies of the known class nodes in Table 2.4. We will aim to predict the node's *avg rating* in some connected graphs from the Amazon data network.

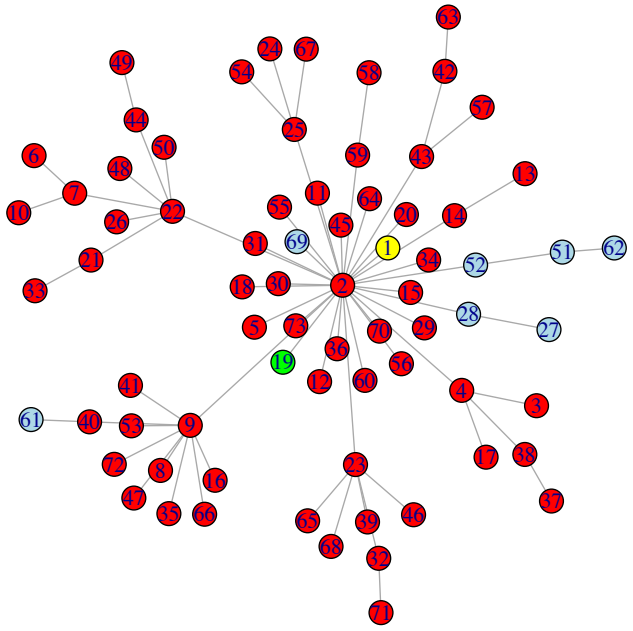


Figure 2.11: Connected component number 468 from the network, $n = 73$, $e = 72$, book(red)=64, music(yellow) = 1, DVD(green) = 1, video(blue) = 7.

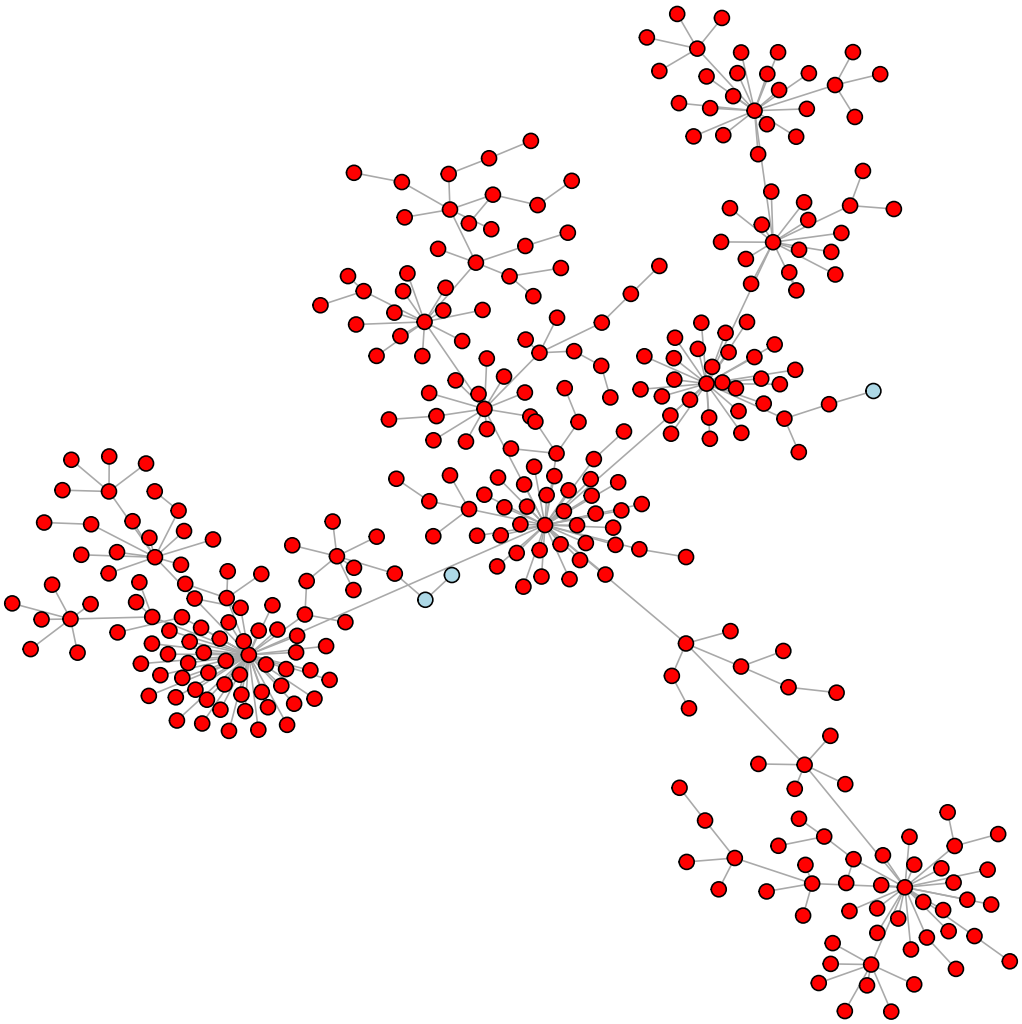


Figure 2.12: Largest connected component in the network which has number 1255 from the network, $n = 332$, $e = 331$, book(red) = 329, video(blue) = 3.

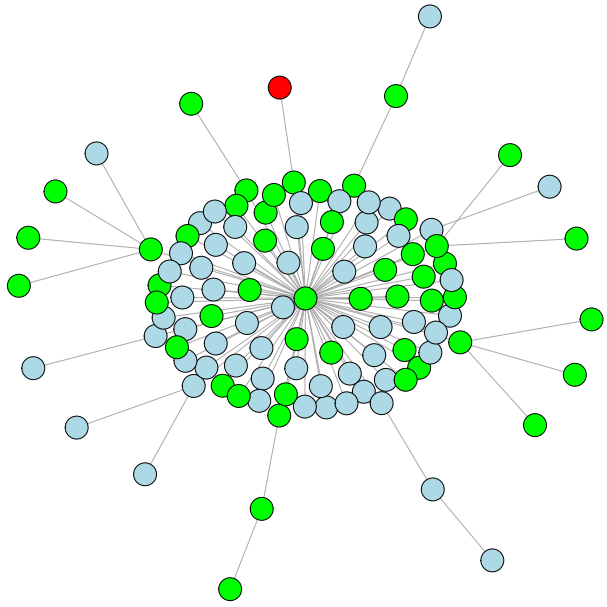


Figure 2.13: Connected component number 3577 from the network,$n = 110$, $e = 109$, book(red)=1, DVD(green)=50, video(blue)=59.

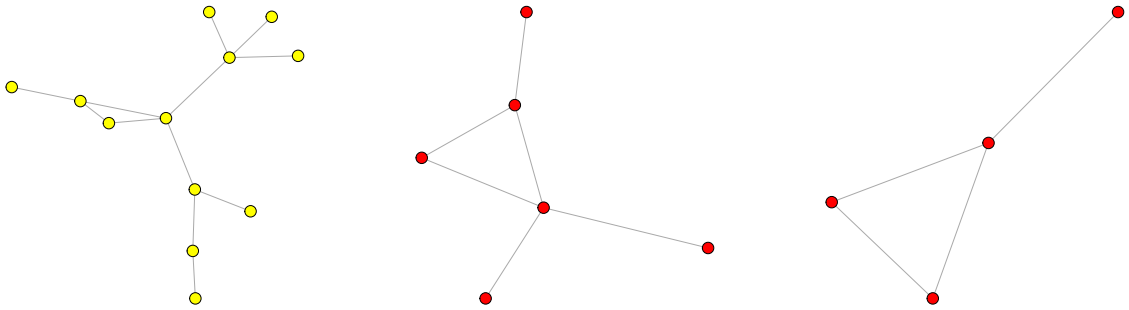


Figure 2.14: Left: connected component number 79 from the network, $n = 12$, $e = 12$, music (yellow) = 12. Middle: connected component number 474 from the network, $n = 6$, $e = 6$, book(red) = 6. Right: connected component in the network which has number 13144 from the network, $n = 4$, $e = 4$, book(red) = 4.

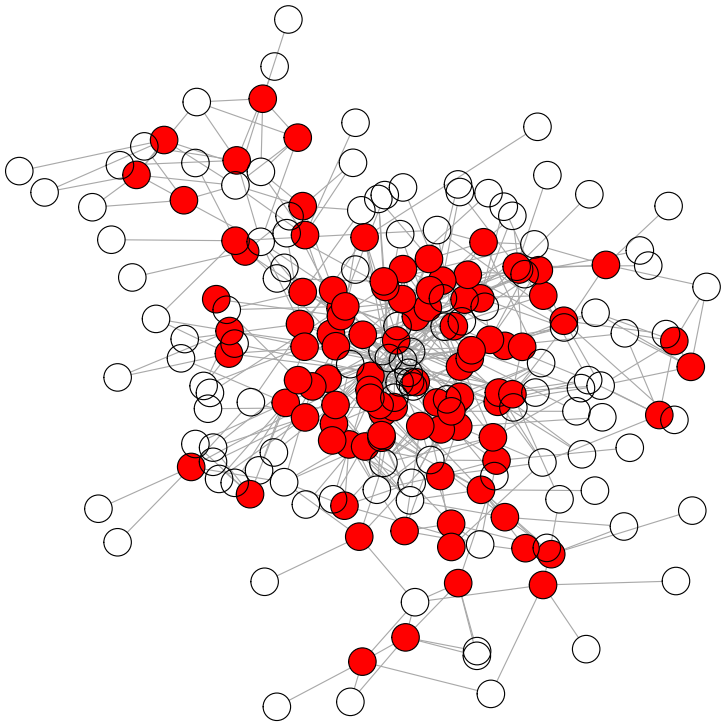


Figure 2.15: Connected component number 33 from the original network, $n = 217$, $e = 479$, book(red) = 101, 116 nodes unknown items.

Avg rating	Frequency
0	39457
1	877
1.5	377
2	1802
2.5	3611
3	11900
3.5	22970
4	54291
4.5	74487
5	85620
Total	295392

Table 2.3: The frequency of nodes *avg rating*, The zero values means that no rating has been made on this product.

Avg rating	B freq	B%	M freq	M%	D freq	D%	V freq	V%
0	35747	16.4%	2581	5.1%	483	3.6%	646	4.8%
1	721	0.3%	60	0.1%	53	0.4%	43	0.3%
1.5	290	0.1%	20	0.4%	48	0.3%	19	0.1%
2	1432	0.7%	86	0.2%	196	1.4%	88	0.7%
2.5	2726	1.2%	235	0.5%	426	3.1%	223	1.7%
3	9154	4.2%	898	1.8%	1088	8%	760	5.7%
3.5	16790	7.7%	2476	5%	2109	15.6%	1594	12%
4	39284	18%	8209	16.4%	3561	26.3 %	3236	24.3%
4.5	48925	22.4%	17378	34.8%	4012	29.6%	4172	31.4%
5	63482	29%	18036	36.1%	1582	11.7%	2517	19%

Table 2.4: The *avg rating* frequency table of nodes classes (B = book, D = DVD, M = music, V = video) and percentage for each *avg rating*.

Comments on the Amazon network

- The data is described in Leskovec et al. (2007) as a directed graph, this is because Leskovec et al. consider the Amazon network as a person to person recommendation network. For example, customer i recommends product x to customer j at time t. From this concept, the nodes correspond to the customers and the edges to the recommendations with all information about the product. However, we take the views that Amazon network is not a recommendation network as it is not specific to one customer. In our analysis of the network, we deal with it as an undirected network which has customers' reviews. If product x appears in the similar products of product y and vice versa, there will be a double edges between nodes and we simply remove one of them from the graph.

- The data was collected in Summer 2006. So, the network will have changed and increased with time, not just increasing the connections, but also adding more products with their properties, for example: downloaded times, and *avg rating*.

- Some of the *avg rating* are based on few reviews, for example, we can find a product with *avg rating* equal 4 but only with one rating equal 4 from one customer review. In contrast, another product has the same *avg rating* with more than 10 ratings from different customers. This may lead us to do further analysis taking account of uncertainty in the *avg rating* values.

- The *avg rating* of a product is calculated as the average of all ratings from the customers with rounding to the nearest half.

2.5 Discussion

In this chapter, we covered the basic and most relevant concepts and measurements in graph theory that we use in this study. We also discuss four different graph models, the classic random model (ER), the small-world model (WS), scale-free networks with Barabási and Albert model (BA), and forest fire model (FF).

Then, we introduce two real networks, and give a general description of the their structure

and the nodes features based on Leskovec and Krevl (2014). The first network is Facebook, which has dense connected components and an interesting topology structure with lots of ties. However, the nodes' features are not straightforward to deal with, as they are represented by lots of zero values and few ones. In contrast, the Amazon network is rich in information, which gives a chance to apply more analysis on the nodes' features. However, the topology structure is a tree model.

In our study, we apply some machine learning techniques on both networks, for example, clustering in Chapter 4, classification in Chapter 5, and prediction in Chapter 6. We compare the results from these real networks with results from simulated graphs generated from the four models previously discussed.

There are other social networks which have interesting structures, for example, Twitter, and Google+. The user profiles contain a wealth of information and details which may lead to a variety of relationships between users. This allows researchers to think of and suggest new ways to analyse these networks which could help in different ways, for example, psychologists could understand the behaviour of individuals, how they interact within groups, and how they make decisions. Further, studying the interactions between individuals and analysing the influential nodes in circles or big components may help in predicting the pattern of network growth over time. This may or may not be related to the idea of link prediction which is a common topic in graph theory and more common in directed graphs. Also, such new analysis could help in improving these network websites.

Chapter 3

Distances in Graphs

3.1 Introduction

In this chapter, we provide an introduction to distance algorithms and metrics in graphs, with an overview of the history of shortest path algorithms. We also describe the most common shortest path measures, limitations of these measures, and how they motivated us to think of new and different ways to overcome these limitations. We introduce a new distance metric for graphs, that resolves the limitations of the known measures. This new distance metric is used with different graph models, and is shown to produce effective results in graph analysis in the following chapters.

3.2 Background

The problem of computing distances and shortest paths between vertices in graphs is one of the most fundamental and well-studied problems in graph theory. The shortest path corresponds to the minimum path length between any pair of vertices in a graph, and in the case of directed graphs there are source and destination vertices which determine the direction of the path. The shortest path is useful in many different applications, resulting in different measures having been created to match the applications' purposes and the different graph types. For example, some applications require exact shortest paths with no constraints on time taken to find the shortest

paths, and others allow for approximate shortest path but have constraints on computation time.

The earliest application of distance computation was in transportation network analysis, where the shortest path between two cities is required. This is the principal idea of a navigation system, which guides the driver to their destination through the shortest path (Udaya Kumar Reddy, 2016).

In computer science, routing algorithms in network routers depend on some shortest path algorithms to find the minimum cost link (the fastest route) between routers in a network (Kurose and Ross, 2017).

Another application in social network analysis is the "six degree of Kevin Bacon" network. There is a website called "The Oracle of Bacon", which is a connected graph of all American actors and actresses in which two actors are connected if they have appeared in the same film. This website downloads and updates the database from the Internet Movie Database, which includes millions of actors, movies, TV shows and nicknames. The object of the website is to find the shortest path between any actor or actress and Kevin Bacon. The longest shortest path of this graph is six; the shortest path length between Kevin and any actor is called that actor's Bacon number. Bacon numbers rarely exceed 4 (Newman, 2010).

Due to this variety of applications, there are many studies on graph types. For example, the graph can be either static or dynamic, where the former has a fixed number of vertices and edges, and the latter updates the graph's structure by adding or deleting vertices and edges or changing edges' positions. The graph's edges can be directed or undirected, and can have either zero, positive or negative weight.

Madkour et al. (2017) classified shortest-path algorithms into two groups: (1) single source shortest-path (SSSP) algorithms, which calculate the shortest path from a source vertex to other vertices in the graph based on the adjacency list representation. (2) All-pair shortest path (APSP) algorithms, which calculate the shortest path between all pairs of vertices in the graph based on the adjacency matrix representation. Madkour et al. (2017) presented a taxonomy of multiple levels of shortest path algorithms as a useful tool to understand the shortest path categories, and to choose suitable techniques for the application object. They also mentioned some challenges with solutions in each group of algorithms.

Brodnik and Grgurovič. (2017) produced an algorithm that improved the time complexity of calculating APSP algorithms in a directed graph with non-negative edge weights via independent SSSP computations.

Udaya Kumar Reddy (2016) reviewed the static APSP shortest-path algorithms that produce exact results for both weighted and unweighted graphs and for both dense and sparse graphs. He also presented some studies on APSP algorithms for some families of graphs. For example, interval graphs, that determine an interval for each vertex containing its neighbours. This interval is determined by the user which then determines the number of neighbours or edges in the interval. Chordal graphs, which are graphs with a cycle length of at least four vertices and an edge connecting two non-consecutive nodes of this cycle. He also discussed some measures in graphs such as the Wiener index, which is the sum of all the shortest paths between all vertices in a graph; and average distances. He also describes some examples from chemistry and discusses the molecular graph structure of chemical components, where the vertices correspond to the atoms, and the edges represent the covalent bonds between atoms.

Nannicini and Liberti (2008) discussed some shortest path techniques in dynamic graphs. They focused on industrial applications related to transportation networks. In road networks, the arcs (edges) are weighted depending on the traffic and travelling time.

Zwick (2001) surveyed the theory of APSP and SSSP algorithms for weighted and unweighted graphs, as well as directed and undirected graphs in the cases of exact or approximate results. He also discussed some related topics, for example, spanners which are subgraphs consisting of all graph nodes with minimum number of edges (also called minimum spanning tree); distance oracles, which give approximate shortest path lengths (instead of exact lengths) between nodes in a graph within exact bound in less time.

As finding an efficient shortest-path algorithm on graphs is still useful in many disciplines, we introduce a new distance measure for undirected and unweighted graphs. This new algorithm has a unique feature, it has the ability to distinguish between shortest paths of equal length. More precisely, whereas the shortest path length is a positive integer, the new measure breaks up the equal integers into rational number. Our idea and its framework are presented in Section 3.4. This measure is used in graph clustering and classification applications, which are discussed in

Chapter 4, 5, and 6.

Before introducing the new algorithm, we shall review the most common shortest-paths algorithms in graph theory, which help in the understanding and developing our new measure.

3.3 Distance Algorithms in Graphs

In this section, we present two of the most common shortest path algorithms for unweighted and undirected graphs. However, the algorithms for weighted and directed graphs are in the appendix. We divide them into two categories: SSSP and APSP algorithms.

3.3.1 Single Source Shortest Path Algorithms

Undirected Unweighted Graphs

The basic types of graphs are unweighted and undirected graphs, where all the edges have weight equal to one and do not have a direction. There are two standard approaches to find the shortest paths in this type of graphs: the breadth-first search algorithm (BFS) and the depth-first search algorithm (DFS).

The BFS algorithm is one of the first and simplest searching algorithms which was created by Moore (1959) whilst trying to find paths in mazes. The pseudo code in Algorithm 1 and a graph example in Figure 3.1 present the BFS algorithm's steps.

BFS constructs a breadth-first tree by setting a source node s as a root, and contains all nodes $v \subset V \backslash \{s\}$ which are reachable from s. Then, it computes the shortest path lengths from s to $v \subset V \backslash \{s\}$ by counting the number of edges of the SP. BFS works on both directed and undirected graphs.

Algorithm 1 deals with the graph G as an adjacency list, it searches the graph by starting with the source vertex s and finding all s's neighbours with shortest paths (immediate neighbours) before moving to the next vertices of s.

BFS requires some vertex attributes, such as adjacent nodes (the nearest neighbours from the adjacency list), predecessors (nodes' parents) and vertex colours. Algorithm 1 starts by initializing all nodes with their colour set to white, their distance from s to infinity, and their predecessor π to be empty. Also, the initial colour setting of s is gray, and the distance d equal to 0 with no parent for s. Then, it defines the first -in, first-out queue Q and sets s as the first node in Q. Each node adds to the bottom of Q and has its colour set to gray by ENQUEUE operation, then leaves the top of Q and has its colour set to black by DEQUEUE operation. The "while" loop over lines 10-18 iterates until all nodes have black colour (in case of connected graph) and updates the distances in each iteration. The node colour is changed successively from white to gray when it is reached and visited, and then changed to black when it is examined. At the end of the algorithm, the shortest path $d(s,t)$ from source s to any vertex t is defined if and only if t is accessible from s, otherwise the shortest path is ∞.

Algorithm 1. BFS (G, s) (Cormen et al., 2009)

```
1:  for each vertex u ∈ V\{s}
2:      u.colour =WHITE
3:      u.d = ∞
4:      u.π =NIL
5:  s.colour =GRAY
6:  s.d = 0
7:  s.π =NIL
8:  Q = ∅
9:  ENQUEUE(Q, s)
10: while Q = ∅
11:     u = DEQUEUE(Q)
12:     for each v ∈ G.Adj[u]
13:         if v.colour == WHITE
14:             v.colour = GRAY
15:             v.d = u.d + 1
16:             v.π = u
17:             ENQUEUE(Q, v)
18:     u.colour =Black
```

Theorem 3.3.1 (Jungnickel, 2013) *At the end of the BFS algorithm, every vertex t of G satisfies:*

$$d(s,t) = \begin{cases} d(t) & \text{if } d(t) \text{ is defined,} \\ \infty & \text{otherwise.} \end{cases}$$

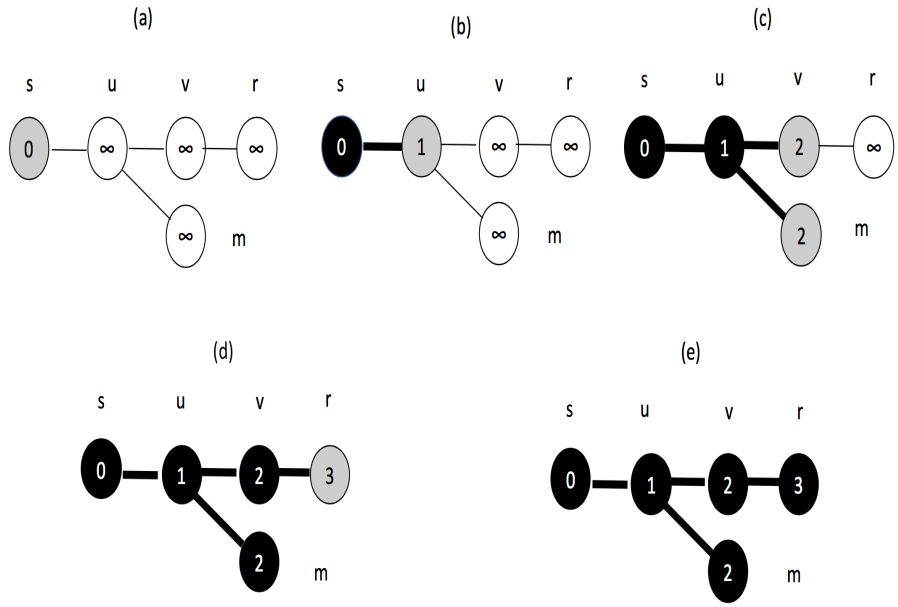


Figure 3.1: An example of an unweighted undirected graph to show the procedure of the BFS algorithm. (a) start with the source vertex s, which is coloured gray (the rest is white); $s.d = 0, u.d = \infty$ for $u \in V \backslash \{s\}$. (b) move to the first level of neighbours u and change its colour to gray; $u.d = 1, u.\pi = s$. (c) move to the second level of neighbours v, m and changes their colour to gray, $v.d = 2, m.d = 2, v.\pi = u, m.\pi = u$. (d) continue to the node third r with shortest path $= 3$ and repeat the action. (e) All vertices are visited and discovered. The queue is $Q = s, u, v, m, r$, which starts with s, until reach the furthest vertex r.

The DFS algorithm is an old algorithm in graph searching; the earliest unpublished version was in 1882, and was devised by the French mathematician Charles Pierre Trémaux (Lucas, 1959).

Similar to the BFS algorithm, the DFS algorithm starts with a source vertex s, and marks it as a visited point; but, it searches by discovering one node v of s's neighbours with $s+1$ shortest path length, and marks v as a visited point. Then, it discovers all of v's neighbours and marks one of them as a visited point before reaching and visiting s's other neighbours. It continues the same process until all V nodes have been visited. In the case of directed graphs, if there are

some vertices that have not yet been visited after visiting all v's neighbours, the algorithm will select a new source vertex and apply the same criteria. DFS uses the same criteria of colouring the nodes, white as initial colour, gray when the node is reached, then black after it is examined. Figure 3.2 shows DFS algorithm's steps.

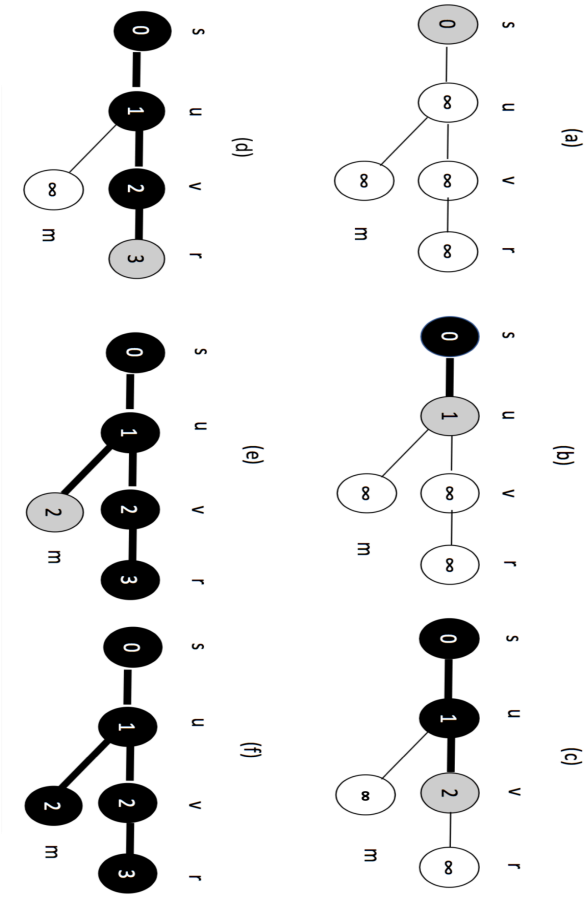


Figure 3.2: An example of an undirected and unweighted graph to show the procedure of the DFS algorithm. (a) start with the source vertex s, which is coloured gray (the rest is white). (b) move to the first level of neighbours u and change its colour to gray. (c) move to the second level of neighbours v and choose one of them to change its colour to gray. (d) continue to third level of neighbours r (before visiting all u neighbours) and repeat the action. (e) visited and discover all vertices in the branch s, u, v, r, then go back to discover the rest of the v neighbours m. (f) All vertices in the graph are visited and discovered. The queue is $Q = s, u, v, r, m$, which starts with s, and then the next vertex is visited, deleting each vertex until the queue is empty.

Because both the BFS and DFS algorithms scan the adjacency list rather than the adjacency matrix, they take $O(|E|)$ time to complete scanning. Over a graph of V vertices, the total running time of each algorithm is $O(|V| + |E|)$ (Douglas, 2001; Cormen et al., 2009; Jungnickel, 2013). Chan (2012) presented two different strategies for calculating APSP for unweighted

undirected graphs, which improved the running time of the breadth-first search algorithm. The first strategy is based on the node degree level as it treats high-degree vertices separately from low-degree vertices. The idea behind this separation is that working with high-degree vertices makes the process faster as the paths pass fewer vertices. In contrast, low-degree vertices have fewer edges, which makes the algorithm faster as we do not have to pass many edges to find the shortest paths. In the second strategy, Chan was inspired by the algorithm of Aingworth et al. (1996); the main difference between them is that the former produces exact distances while the latter produces approximate results. Aingworth et al.'s algorithm is very similar to Chan's algorithm, as it deals with short shortest paths and long shortest paths separately. However, Chan considered a modification of his second algorithm to be suitable for unweighted and directed graphs.

In the case of undirected unweighted graphs, Seidel (1995) introduced a new algorithm called APD, which computes the APSP in time $O(M(n)log\,n)$ where $M(n)$ is the time to multiply two $n \times n$ matrices, and n is the graph size $|V|$. The APD algorithm uses the adjacency matrix and produces a distance matrix with integer values without any information of the paths themselves. However, it cannot be generalized to directed or weighted graphs.

As these algorithms cannot deal with weighted graphs, we introduce in Appendix A the Dijkstra's algorithm as a fundamental method to solve shortest path problems in nonnegative weighted graphs, and the Bellman-Ford algorithm for graphs with negative weighted edges.

3.3.2 All-Pairs Shortest Path Algorithms

Solving APSP problems can be done easily by running one of the SSSP algorithms $|V|$ times ($|V|$ is the total number of vertices), i.e. repeating the algorithm for each vertex v as a source. In the case of undirected unweighted graphs, the BFS algorithm will spend $|V|O(|V| + |E|) = O(|V|^2 + |V||E|)$ time to solve the APSP problem in the graph. For the directed weighted graphs, we introduce the most common algorithms in the Appendix B.

3.4 A New Distance Measure in Graphs

Finding the shortest path in graphs is a significant topic in graph theory as it is required in various real-life applications. Some examples were described in Section 3.2. Therefore, it has attracted much of our interest during this research. There is a diverse range of graph models: weighted or unweighted, directed or undirected, dense or sparse. However, there is no one universal approach to find shortest-distances that could be suitable for all different graph models.

Most of the straightforward approaches that solve the shortest-path problem use the tree search idea. This starts the search from the source vertex (root) and passes along the branches to reach the destination; this is done by following the BFS algorithm or Dijkstra's algorithm from each vertex to the rest of the vertices. In recent years, there have been many improvements to the algorithms, which usually focus on decreasing the time complexity. In this study, we focus on undirected unweighted graph models, and we devise a new measure for distances in graphs by introducing a different methodology.

The majority of dense graphs have ties between the shortest paths, which means that there are different paths of equal length connecting some pairs of nodes. In this research, we found that these ties are real obstacles in several applications. For example, in clustering of vertices, the aim is to allocate a vertex to the nearest centroid's cluster, which is hampered by a tie in the distances between the vertex and several of centroids. Therefore, due to this issue, we thought of a new way of measuring distances in graphs, which allows breaking of ties. Thus, the main aim of this new approach is to break the ties between equal distances, and distinguish between these equal distances. Most path lengths are integer, while our output distance metric produces real values of distances. For example, in the case of the shortest paths of a graph with a diameter equal to 5, the BFS algorithm produces five integer values $\{1, 2, 3, 4, 5\}$ with equal distances between several pairs of nodes.

Our new algorithm, however, produces five non-overlapping sets of real-value distances between all pairs in the graph. This means that the values in these sets are ordered from smallest to biggest and the maximum value of any set is less than the minimum value of the next set.

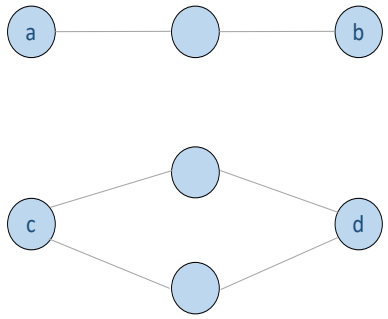


Figure 3.3: Small graph example show the ties between pair of nodes with equal length shortest paths. With our breaking ties distance, $d(c,d) < d(a,b)$, while with BFS, $d(c,d) = d(a,b)$.

In Figure 3.3, a small example shows how our measure breaks the ties in equal shortest paths.

The proposed distance is not a transient thought; rather, it is a result of cumulative trials of various updated distance metrics until the breaking ties aim was achieved. During several experiments, we could determine the important and effective parameters that cause these ties: the vertex degree, the graph diameter and the plurality of shortest paths between a pair of nodes.

Our concept is inspired by the general principle: 'more relations, more strength'. We believe that a pair of nodes joined together by multiple equal paths should be thought of as closer than a pair of nodes joined by a single path of the same length. In social networks, for example, any two vertices (two persons) that have three relations (such as gender, neighbourhood and school) appear to be closer than two vertices have just one relation.

Based on this assumption, we have carried out a lot of experiments to find a distance metric formula that combines all available paths between any pair of vertices. However, the resulting distances do not refer to existing (virtual) routes in the graph. This new distance measure produces distances between vertices which often distinguish between equal shortest paths.

My metric starts with a simple version of the distance metric, and then undergoes several iterations until reaching our final version. Here we outline the first iteration and some of the essential changes, with explanations of each update.

First version

The first trial involved bounded shortest paths in graphs. For example, consider an undirected connected graph G with 15 nodes $V(G) = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o\}$ and 18 edges, where the edges connect these nodes as in Figure 3.4.

Let SP be the shortest path, and d_{SP} is the length of the shortest path. So, here we have $d_{\text{SP}}(a, o) = 3$, and $d_{\text{SP}}(a, h) = 3$. But, it is obvious that there are two SP between nodes a, o and one SP between nodes a, h. In our approach, we are looking for a new distance between a, o that is less than 3 but larger than 2. Therefore, we consider the length of the shortest path d_{SP} and the frequency of d_{SP} (the number of equal shortest path d_{SP} connecting a pair of nodes) to be key values in the trial. We connect them in the formula:

$$d(v_i, v_j) = \frac{d_{\text{SP}}(v_i, v_j)}{f^{1/f}} \tag{3.1}$$

where f is the frequency of the SP of the length d_{sp} and $v_i, v_j \in V$. For the denominator power, we start by trying a simple constants $1/2, 1/3, 1/4, \ldots$ until we satisfied with the power $1/f$ which depends on the frequency of the SP. It minimizes the denominator as much as f increased. In the case of just one short path between (v_i, v_j), the shortest distance is equal to $d_{sp}(v_i, v_j)$. In Figure 3.4, there are two shortest paths of length 3 between a and o, so $d(a, o) = \frac{3}{\sqrt{2}} = 2.12$.

However, this formula has some obstacles, especially in dense graphs when $d(v_i, v_j)$ produces unrealistic results. For example, there is the case of $f \in \{2, 3\}$, the distance in Expression 3.1 is $d_{(f=2)}(v_i, v_j) > d_{(f=3)}(v_i, v_j)$. While by increasing f to $f \in \{4, 5, ...\}$, the result for the distance reverses, $d_{(f=5)}(v_i, v_j) > d_{(f=4)}(v_i, v_j)$.

Second version

The first version formula was updated to overcome these drawbacks by using all possible paths between vertices which have different lengths. However, we still had some situations, for example, where two nodes in a graph have a variety of paths with different lengths, and we aim to merge these paths to obtain a distance less than the shortest path d_{SP} between these nodes. Then, if all the paths' lengths have similar distances (e.g., length (1^{st} path)$= 3$, and length (2^{nd}

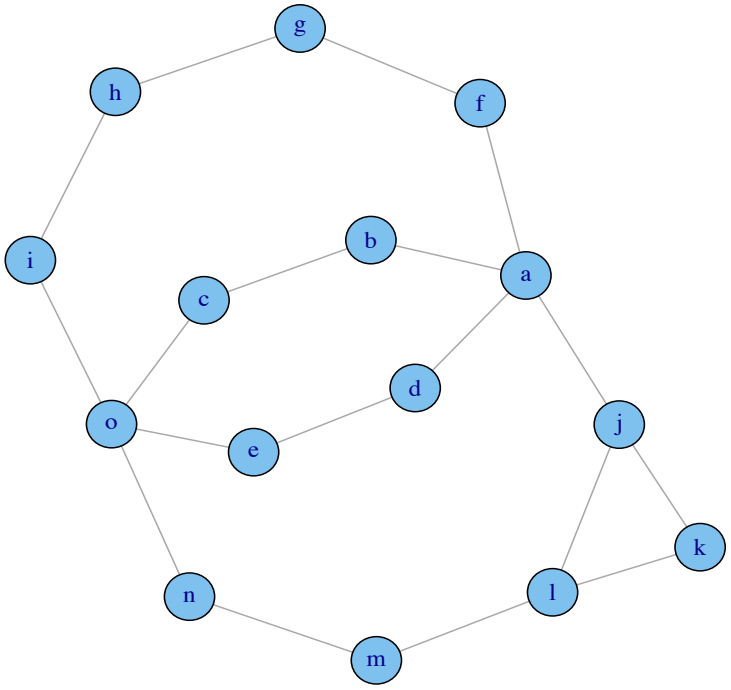


Figure 3.4: Undirected graph G with 15 nodes and 18 edges.

path) $= 4$ between v_i and v_j), it gives a distance smaller than length (1^{st} path). In contrast, if the paths' lengths were quiet different (e.g., length (1^{st} path)$= 2$, and length (2^{nd} path) $= 10$ between v_i and v_j), we should have a distance $d(v_i, v_j) < 2$ after merging all paths together, which is impossible. So, the merging process is not viable.

We then rejected the idea that the proposed distance $d(v_i, v_j)$ should necessarily be less than or equal to the d_{sp} and introduced a new version of the formula $d(v_i, v_j)$. We did this by merging all possible paths between any pair of nodes simply with a condition that the distance matrix $d(v_i, v_j)$ produces distances which preserves the order of shortest path lengths without overlap. This means if $d_{\text{SP}}(v_i, v_j) < d_{\text{SP}}(v_k, v_s)$ then the proposed distance should also satisfy $d(v_i, v_j) < d(v_k, v_s)$.

This distance metric was mainly based on measuring similarities between vertices in a graph. The vertices that are close together or neighbours have more similarities. The further apart the vertices are, the less similar they are. In addition, we added the parameter x (the largest node degree in the graph), which plays a key role in a similarity matrix as it is a sign of the plurality of paths in the graph. If x is large, the graph has variety of paths between

vertices. However, if x is small, the graph has few possible paths between vertices. Based on this concept, we formulate the similarity matrix S_{ij} in Eq. (3.2). The dissimilarity matrix D_{ij} in formula (3.3) transforms the matrix S_{ij} to have small distances between closer nodes and large distances between the further nodes.

The similarity matrix S_{ij} of graph $G(V, E)$ is given by:

$$S_{ij} = 1/2 \sum_{r=1}^{diam} \frac{(A^r)_{ij}}{2^r x^r}, \tag{3.2}$$

where

- $A_{(n\times n)}$ is the adjacency matrix, which has entries of $0, 1$ with 0 in the diagonal, and n is the number of nodes.
- $diam$ is the diameter of G, the longest SP in the graph, where $diam \leqslant n - 1$.
- x is the largest vertex degree in G.
- A^r is the usual matrix multiplication.
- $(A^r)_{ij}$ is the number of possible paths between nodes v_i and v_j of length r.
- The factor $\frac{1}{2}$ is to make $S_{ij} < 1$.

S_{ij} is based on matrix multiplication of the adjacency matrix and the result of Theorem 2.2.2, which is repeated here for convenience.

For the dissimilarity matrix we propose

$$D_{ij} = -\log(S_{ij}), \quad i \neq j \quad \text{with} \quad D_{ii} = 0. \tag{3.3}$$

D_{ij} is a matrix of all the distances between vertices in graph G.

We experimented with different forms of monotonic transformation which preserve the order of the distances to be dissimilarity function of S_{ij} until we found that $D_{ij} = -\log(S_{ij})$ was more suitable with my S_{ij}; this is because $0 < S_{ij} \leq 1$ and the logarithm converts the values of S_{ij} to the range $[-\infty, 0]$ with less ratios between the large and

small distance than other monotonic transformation. Then, taking negative logarithm maps values onto $[0, \infty]$, where closer pairs have smaller values.

Theorem 3.4.1 (Chartrand et al., 2011) *If A is the adjacency matrix of a graph G with $V(G) = \{v_1, v_2, ..., v_n\}$, then the (i,j) entry of $A^k, k \geq 1$, is the number of $v_i - v_j$ walks of length k in G.*

However, the walk of length k does not necessarily pass through unique vertices $v_0, v_1, \ldots, v_k$ even if there is an edge between v_{i-1} and v_i for each $i = 1, 2, \ldots, k$. Any walk linking a pair of nodes in a graph could revisit the intermediate nodes more than once. This phenomenon is called backtracking walks, and it differs from the path between a pair of nodes which is a sequence of different unique nodes. In real life, there are situations of how information can flow along a non-shortest path, for example, spreading a gossip in a social network, when the news could backtrack repeatedly before arriving at the final destination (Estrada and Hatano, 2008).

Our distance metric is based on both the shortest path length and the different walks between nodes. The main aim of this metric is to break the ties within sets of shortest path lengths and distinguish between them.

For example, if $G(V, E)$ is a graph with $diam = 5$, the distance d_{sp} has five different sets of integer values: $1, 2, 3, 4$ and 5, and my distance matrix considers five different non-overlapping sets of real values as shown in Figures 3.5 and 3.6. This formula works very well with different graph models, but there are some cases where the distance does not preserve the ordering of SP, especially in dense graphs. In other words, we could have a chance of not satisfying the condition $d(v_i, v_j) < d(v_k, v_s)$ even though $d_{sp}(v_i, v_j) < d_{sp}(v_k, v_s)$. For example, let $G(V, E)$ be a graph with $diam = 6$, and there be one shortest path between v_i and v_j that has a length of four, zero paths of length five and two paths of length six. Also, there are four shortest paths between v_k and v_s with a length of five and ten paths of length six. In this situation, the distance between v_k and v_s is less than the distance between v_i and v_j due to the imbalance in the number of possible paths between vertices.

However, some applications in networks (which are based on the data flowing) are greatly affected by the plurality of paths between nodes and this overlap could be appropriate. For

example, in computer science, a message from the sender (v_i) to the recipient (v_j) passes the vertices in the shortest path route but, if the path is blocked, the message chooses a longer path to travel. Thus, the distance between (v_k, v_s) is likely to be shorter than the distance between (v_i, v_j). In addition, transportation applications, which require the shortest path, can suggest that the distance between (v_k, v_s) is shorter than (v_i, v_j). This is due to the number of routes of length 5 between k and s are larger than the number of paths of length 4 between i and j, which gives the driver flexibility to change the route if any path is blocked.

Third version

In our research, we improved the distance metric to resolve this issue, and updated the formula to be more sensitive by giving the shortest paths more weight and the longer walks less weight in our distance measure. This improvement makes our metric appropriate for any graph model (even for very dense graphs). For the similarity matrix S_{ij}, we update the formula to be (Almulhim et al., 2019):

$$S_{ij} = \sum_{r=1}^{\text{diam}} \frac{(A^r)_{ij}}{(2\max_{i,j}(A^r))^r}, \tag{3.4}$$

where $\max(A^r)$ is the largest element in A^r. Note that

- $S_{ij} = 1/2$ only if diam $= 1$.
- In the case of $r > 1$, $\max(A^r)^r_{ij} > (A^r)_{ij}$.

The dissimilarity matrix $D = (D_{ij})$ is given by:

$$D_{ij} = -\log(S_{ij}), \quad i \neq j \quad \text{with} \quad D_{ii} = 0. \tag{3.5}$$

It is necessary to assume that we have two sequences $a_r,\ b_r \geqslant 0$

$$a_r = \frac{(A^r)_{ij}}{(2\underset{i,j}{max}(A^r))^r}, \quad b_r = \frac{1}{2^r}$$

from the comparison test theorem in Dienes (1931) in theorem I in Chapter3, if:

$$a_r \leqslant b_r \quad \text{then} \quad \sum a_r \leqslant \sum b_r$$

Thus:

$$\sum \frac{(A^r)_{ij}}{2^r (max(A^r)_{ij})^r} \leqslant \sum \frac{1}{2^r}$$

Because b_r is a geometric series and has the property:

$$\sum b_r = \sum_{r=1}^{\infty} \frac{1}{2^r} = \frac{1/2}{1 - 1/2} = 1$$

Therefore:

$$\sum \frac{(A^r)_{ij}}{2^r (max(A^r)_{ij})^r} \leqslant \sum \frac{1}{2^r} = 1 \Rightarrow \sum \frac{A^r}{2^r (max(A^r)_{ij})^r} \leqslant 1$$

So, $0 < S_{ij} \leqslant 1$.

The breaking-ties distance (in short BTD) satisfies some conditions which makes it suitable to measure distances in all graph models in this study:

Lemma 3.4.1 *Let $G(V, E)$ be an undirected, unweighted graph with vertices V (nodes) and edges E , then:*

1. *BTD is symmetric, i.e. $d_{BTD}(v_i, v_j) = d_{BTD}(v_j, v_i)$ for $v_i, v_j \in V$.*
2. *BTD preserves the shortest-paths order, i.e. if $d_{SP}(v_i, v_j) < d_{SP}(v_k, v_s)$ then BTD satisfies $d_{BTD}(v_i, v_j) < d_{BTD}(v_k, v_s)$. This is evident from the examples in Figures 3.5 and 3.6.*
3. *BTD satisfies the triangle inequality $d_{BTD}(v_i, v_j) \leq d_{BTD}(v_i, v_s) + d_{BTD}(v_s, v_j)$.*

Proof:

1. Since the similarity matrix depends on the adjacency matrix, and the similarity matrix is symmetric, so the BTD is symmetric.

2. We make a simulation study to generate 10^5 graphs from each graph model, and calculate the BTD and SP to check that there are no overlaps in BTD distances between different sets of SP distances.

3. Consider $\{v_i, v_j, v_s\} \in V(G)$, and we have SP between each pair of these nodes: $d_{\text{SP}}(v_i, v_j)$, $d_{\text{SP}}(v_i, v_s)$ and $d_{\text{SP}}(v_s, v_j)$, so either $d_{\text{SP}}(v_i, v_j) = d_{\text{SP}}(v_i, v_s) + d_{\text{SP}}(v_s, v_j)$ in case of node v_s is an intermediate node of the shortest path between v_i and v_j. Or $d_{\text{SP}}(v_i, v_j) < d_{\text{SP}}(v_i, v_s) + d_{\text{SP}}(v_s, v_j)$ in case of v_s is not an intermediate node of the same shortest path. In both cases, $d_{\text{SP}}(v_i, v_j) \leq d_{\text{SP}}(v_i, v_s) + d_{\text{SP}}(v_s, v_j)$. From Lemma 3.4.1(2), we can generalize the triangle inequality to BTD.

The main difference in the similarity function between Eq. (3.2) and Eq. (3.4) is the change in the denominator from x^r to $\max(A^r)^r$. Because x is the maximum node degree in the graph, it is a fixed value and does not depend on r. While $\max(A^r)$ is variable and increases in each iteration, it is equal to x only in the second iteration when $r = 2$. This is because $\max_i\{(A^2)_{ii}\}$ refers to the maximum number of paths between the vertex to itself of length two, and this is equal to x as it is the number of connections between the maximum super vertex with its neighbour. We believe that this change precludes any overlap between BTD, and simulation experiments confirm this. We carry out many experiments which compare BTD and SP on different random graphs according to four different graph models; the ER, FF, WS, and BA graph models Csárdi and Nepusz (2006). We also test some real network from the Facebook and the Amazon networks.

The experiments show that for these graphs BTD produces distances which preserve the SP order without any overlap between distances, i.e. if $d_{\text{SP}}(v_i, v_j) < d_{\text{SP}}(v_k, v_s)$ then BTD satisfies $d_{\text{BTD}}(v_i, v_j) < d_{\text{BTD}}(v_k, v_s)$.

In the first rows of Figures 3.5 and 3.6, all graph models show both a spreading and different values for the BTD in each SP group. Also, in Figure 3.7, the plots show the distances which correspond to the first ego network from Facebook (see Figure 2.9), and connected component from the Amazon network (see Figure 2.15).

In tree like graph models, such as the BA model, there is a unique shortest path between all pairs of nodes. In this type of graph, ties appear between paths that have the same source node

but different destination nodes. For example, in the BA model in Figure 3.6, the tie-break effect on distances is limited compared with other models due to the tree graph structure, which has less pairs tied for the same length for each length between nodes (the range of the BTD which correspond to SP $= 2$ is equal to 10^{-3}).

In non-tree like graph models, there are multiple pathways connecting a pair of nodes, and the ties issue is clearly visible. Therefore, by computing the BTD and comparing each distance in this algorithm with the associated SP, the scatter plots show a visible variation in the BTD compared with sets of identical distances in d_{sp}.

From this experiment, we concluded that BTD succeeds in breaking ties between vertex pairs with equal SP, especially in dense graphs.

It is clear that there are gaps in the scatter plots of Figures 3.5 and 3.6 (this gaps are between the maximum of any set of BTD and the minimum of the following set), which are the results of the power of the denominator. We tried to minimize the gaps by reducing the power of the denominator, but the overlap appeared in some cases.

Before ending this chapter with a conclusion, it is worth mentioning that there is some literature on node similarity measures in networks but, it differs from our study. These measures depend on finding similarities between nodes profiles (features) which has different applications such as link prediction, graph clustering, friends suggestion, web page filtering and movie suggestions (Rawashdeh and Ralescu, 2015).

Also, some other studies are interested in similarities between graphs and similarities between nodes from different graphs. More details for some measures and applications can be found in Nikolić (2012); Blondel et al. (2004).

Through our search in the node graph similarity topic (in a single graph), we have noticed the lack of studies in this area. Most of these studies are based on the relations between the nodes in directed graphs. For example, consider a small web graph shown in Figure 3.8, there are two web pages of two professors: “ProfA” and “ProfB”, and two of their students: “StudentA” and “StudentB”, with the home page of their university “Univ”. The hyperlinks between web pages are represented by the edges. One of the oldest similarity measures is called co-citation

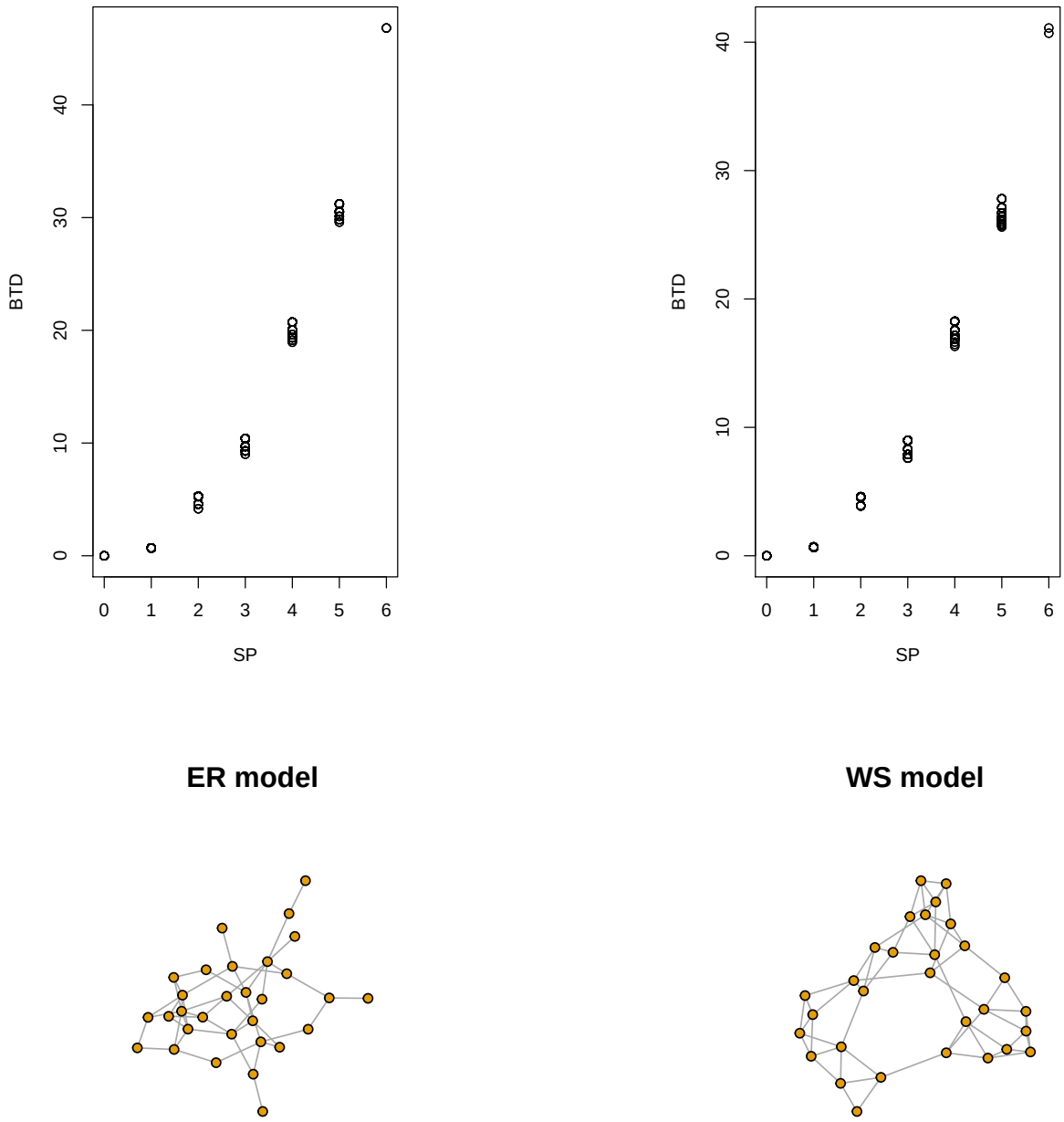


Figure 3.5: Examples of graph models with 30 vertices, and the corresponding plots illustrate the breaking of ties in SP distance by BTD.

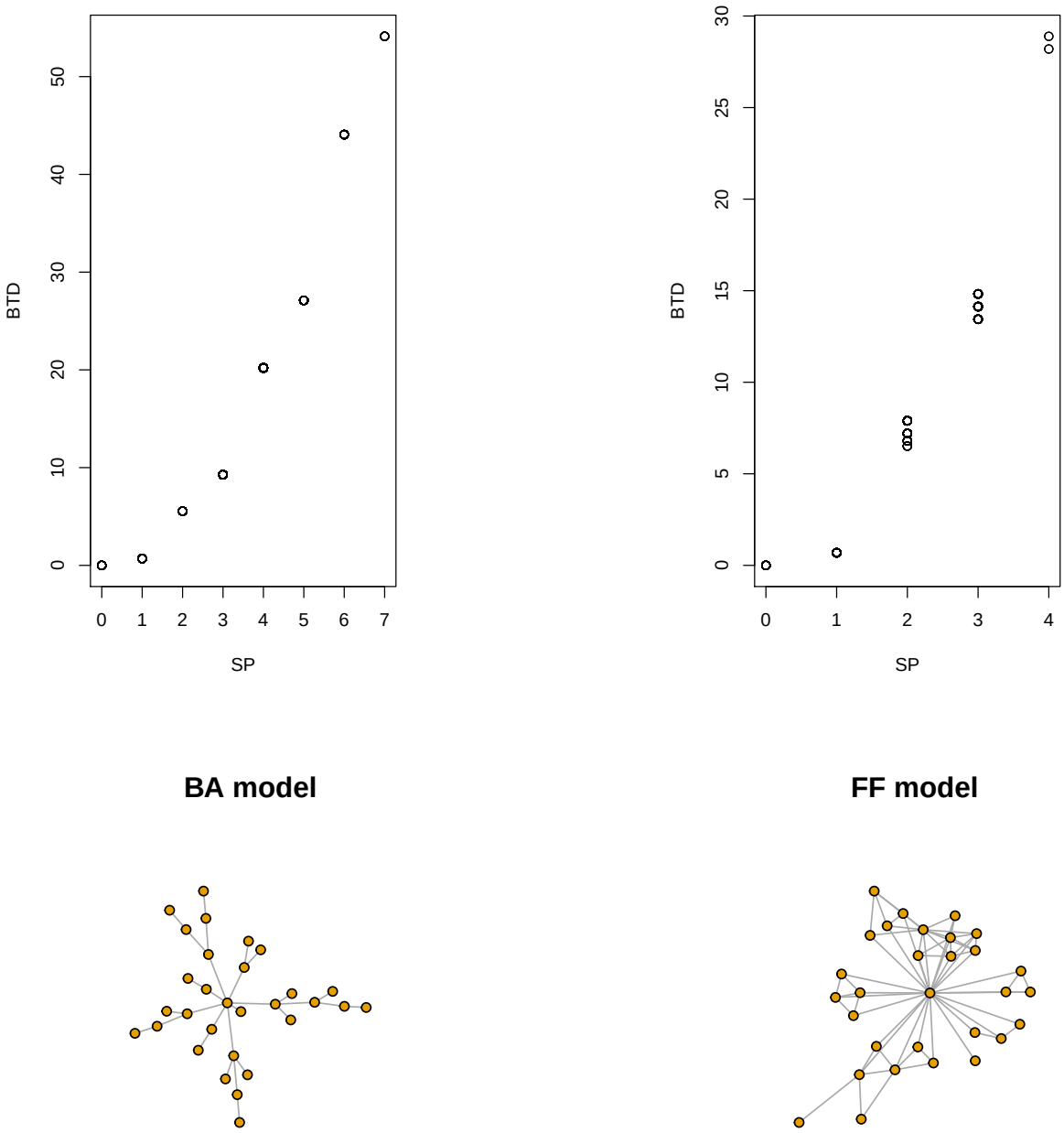


Figure 3.6: Examples of graph models with 30 vertices, and the corresponding plots illustrate the breaking of ties in SP distance by BTD.

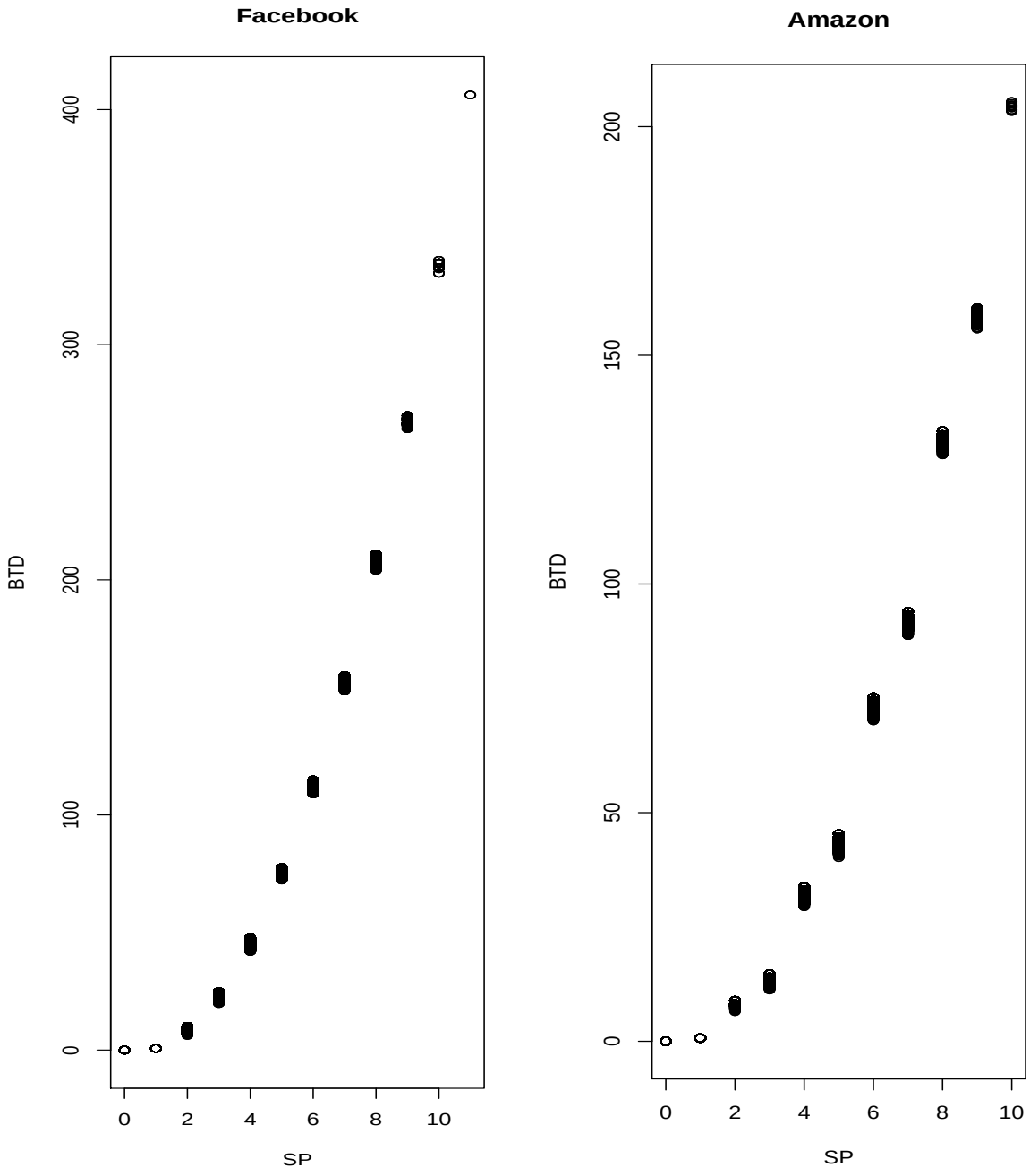


Figure 3.7: Two plots show the breaking ties in SP distances by BTD in Facebook first ego and connected component from the Amazon network.

(Small, 1973). It considers that "ProfA" and "ProfB" are similar because they are both linked to "Univ". This idea was generalized by Jeh and Widom (2002) so that if two objects are considered to be similar like "ProfA" and "ProfB", and they are references to another two objects like "StudentA" and "StudentB", we can conclude that "StudentA" and "StudentB" are also similar. This measure is called SimRank and relies on the idea that "two objects are similar if they are related to similar objects". It has different applications in web pages, graphs and scientific papers with their citations graphs.

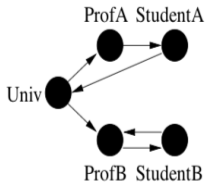


Figure 3.8: Example of small web graph.

Other node similarity measure by Yang et al. (2017) is based on neighbourhood features in graphs. They introduce four different measures of similarities based on popular neighbourhood features: the maximum common neighbourhood pattern (MCNP), the neighbourhood pattern (NP), labelled walks (LW), and k-hops neighbours (KN).

The closest study to our research is the node similarity metric proposed by Huang and Lai (2006). They introduce a similarity (and dissimilarity) metric for the purpose of clustering, but there are multiple differences between our metric and their metric:

- We rely on the adjacency matrix as the basis of our similarity metric, while Huang and Lai depend on the incidence matrix. The incidence matrix is an edge-by-node matrix R defined as $R = (r_{ij})_{|E|\times|V|}$, with entries $\{0, 1\}$:

$$r_{ij} = \begin{cases} 1 & \text{if node } v_j \text{ is incident with edge } e_i, \\ 0 & \text{otherwise.} \end{cases}$$

- We find the similarities between nodes through the similarity metric in Eq. (3.4), while Huang and Lai produce the similarities in two steps (Eq. (3.6) and (3.7)) which cost more time than ours. They calculate the Jaccard coefficient to measure the degree of overlap

between nodes by:

$$\text{sim}(v_i, v_i^*) = \frac{\#(v_i = v_i^* = 1)}{\#(v_i = 1) + \#(v_i^* = 1) - \#(v_i = v_i^* = 1)} \tag{3.6}$$

where v_i and v_i^* are binary vectors of the nodes v_i and v_i^* from matrix R, $v_i, v_i^* \in V$. The numerator is the number of shared attributes i (i.e., edge) linking between both v_i and v_i^* nodes. $\#(v_i = 1)$ is the degree of node v_i. Eq. (3.6) produces similarity score of all pairs of neighbour nodes, but with non-neighbour nodes, the similarity score is zero. So, Huang and Lai suggest to use transitive similarity to solve this issue and find all pairs of shortest paths in the graph. For each shortest path, they calculate the product of similarity value of all node pairs. More precisely, if we have a shortest path between nodes v_i and v_j with intermediate points $v_{k1}, v_{k2}, \ldots, v_{kl}$, and their corresponding node vectors include the set of $P : (r_i, r_{k1}), (r_{k1}, r_{k2}), \ldots, (r_{kl}, r_j)$, the similarity value $s(r_i, r_j)$, which minimizes all the values of $\text{sim}(v_i, v_j)$ over all possible shortest path sets P' between nodes v_i and v_j is given by:

$$s(r_i, r_j) = \begin{cases} \min_{P \in P'} \{\prod_{(r_m, r_n) \in P} \text{sim}(r_m, r_n)\} & \text{if } P' \neq \phi, \\ 0 & \text{if } P' = \phi, \end{cases} \tag{3.7}$$

where $1 \leqslant i, j \leqslant |V|$, and $i \neq j$. Thus, the node similarity matrix is defined by:

$$S = [s(r_i, r_j)]_{|V| \times |V|}.$$

- We produce non-overlapping distances with dissimilarity matrix D in Eq. 3.5, while Huang and Lai produce overlapping distances with their dissimilarity:

$$D = [1]_{|V| \times |V|} - S.$$

Figure 3.9 shows a comparison between our BTD and D compared with SP. It is clear that D has overlaps between distances compared with SP which we succeeded in avoiding.

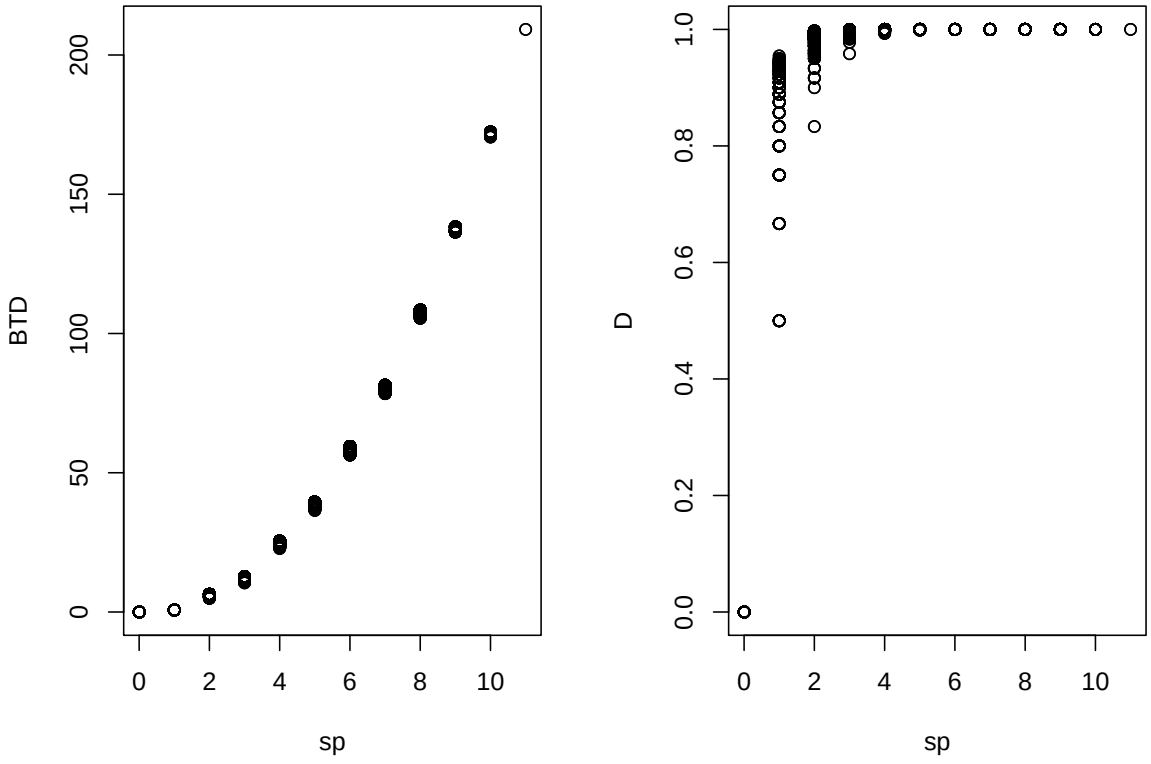


Figure 3.9: Example figures show a comparison between our distances in a graph and D distance compared with SP distance. The distances are produced from an FF graph model with $n = 100$.

3.5 Conclusion

In this chapter, we introduced an overview of common shortest path algorithms for different graph types, weighted or unweighted graphs, and directed or undirected graphs. We focus in this study on unweighted and undirected graph models and aim to find a different distance measure between vertices. The main feature of BTD is to break the ties between most of the identical shortest paths length. Moreover, we carried out experiments to compare the performance of BTD and SP on different random graph models and examples of real networks. We conclude that BTD is a good measure to break the ties and distinguish between equal SP distances, which helps in different applications, as we will discuss in Chapter 4, 5, and 6.

Chapter 4

Graph Clustering

4.1 Introduction

This chapter focuses on clustering in graphs, and how we can improve clustering results by using our distance metric BTD which breaks the ties between nodes (as we discussed in the previous chapter) rather than the classical shortest path distances. In this chapter, we introduce the clustering concept and provide a short review of some clustering algorithms, concentrating on the k-means algorithm. We survey the methods of choosing the initial centroids of the k-means algorithm, and describe our methods in detail. We introduce some common clustering validity measures followed by experimental results. Our experiments are divided into three parts: a performance comparison between the BTD and the SP in hierarchical clustering, a performance comparison between the BTD and the SP in k-means clustering, and a performance comparison between the classic k-means algorithm and the adaptive k-means algorithm by using the BTD. The final section shows the clustering results on a real network example from the Facebook network.

4.2 Clustering background

Clustering, in general, refers to the process of partitioning and grouping similar items, objects or records into groups. Objects in each cluster are similar to each other, and dissimilar to objects in

other clusters. Therefore, optimal clustering maximizes the similarity within each cluster object (or homogeneity), and minimizes the similarity to the records outside the cluster (objects from different clusters are maximally heterogeneous).

The goal of clustering is usually descriptive, either for visualizing or exploring the latent data structure. It captures the natural structure of the data by grouping it into different categories, and discovers natural groups in real data. Therefore, cluster analysis is a common task in data mining and machine learning, as it can be used for investigating different practical problems.

One of the common applications is data summarizing, especially in high dimensional data. Clustering data in different categories helps to represent and interpret the data types. Clustering can be also a preliminary step of other data mining techniques such as classification, which assigns a new item to a cluster.

Moreover, the wealth of clustering applications in different disciplines has inspired researchers to develop various algorithms, which can meet the specific data and application purposes.

Most of the earliest studies divide clustering methods into two categories: hierarchical and partitioning methods. However, new studies suggest expanding the list to include additional categories: density-based methods, grid-based clustering, and the fuzzy clustering method.

Hierarchical clustering (HC) methods represent nested clusters in either a binary tree or a dendrogram, so each leaf corresponds to one observation, and the root is connected to all data objects. It can be done by an agglomerative technique, which assigns each object in a separate cluster, then merges clusters to end with all objects in one cluster. So cutting off the branches is done by the user to choose the desirable cut level. Alternatively, clustering can be achieved by a divisive technique, which initially puts all data objects into one cluster, then recursively divides it into sub-clusters. It continues dividing until it reaches a desirable structure.

Partitioning clustering methods divide the data initially into a predefined number of clusters, then move objects from one cluster to another until it achieves “optimal” global clusters. These methods are usually based on minimizing a criterion function. For example, sum of

squared error (SSE), which is used to compare distances between and within cluster objects via some representative points such as cluster means. The most common and well known method based on this criterion is the K-means algorithm. We give more details about it in the next section.

Mixture density-based clustering methods are designed to deal with data from mixture distributions. Therefore, the objects in each cluster are thought to belong to the same probability distribution and those in different clusters to a different distribution. This is a first step in analysing the data, as it enables us to study the distribution's parameters and behaviour.

The Grid-based clustering method was proposed by Warnekar and Krishna (1979). The general idea of this method is to partition the data space into a finite number of cells to produce a grid structure, then design the clusters based on the regions with high-density cells. It has an advantage in that it reduces the time complexity of clustering in large high dimensional data sets.

The Fuzzy clustering technique considers a soft clustering where each object can belong to one or more clusters with different degrees of membership. This technique is useful in real-life applications where the data is not disjoint. For example, in image analysis, fuzzy clustering produces more accurate results due to the images structure.

There are other methods which are designed for different data types, for example, categorical data, text data, time series data and more. More details on graph clustering or other methods can be found in Aggarwal and Reddy (2014), Larose and Larose (2014), Maimon and Rokach (2010), Tan et al. (2006), (Schaeffer, 2007), (Fortunato, 2010) and Xu (2005).

4.3 General graph clustering

In graphs which consist of a set of nodes V and edges E, the clustering idea is to divide the nodes into different groups, where each group's nodes share more relations. These groups are called clusters, and in some cases, there are restrictions on the number of clusters, or their sizes.

In real networks, these groups are often called communities and appear naturally due to

the real network's structure. A general feature of real graph structure is inhomogeneities in distributions of edges between vertices, and the degree distribution often follows the power law distribution; the majority of nodes have low degrees and have a high tendency to connect with large degree nodes. Therefore, in real networks, the edges distribution between nodes within a cluster is more dense than between nodes from different clusters. In small real networks, distinct communities are usually observable.

In graph theory, there are different algorithms to find clusters in graphs. We distinguish between two major approaches: global (ordinary) clustering and local clustering. In global clustering, the process deals with the graph as a whole which means each node belongs to a single cluster and the union of all clusters is the set of all nodes. Having a graph $G(V, E)$, and clusters $C_1, ..., C_k$, global clustering satisfies the conditions

$$C_i \cap C_j = \phi \quad \text{if} \quad i \neq j,$$

and union of all clusters is

$$\bigcup_{i=1}^{k} C_i = V.$$

Alternatively, there is the local clustering approach, which applies a clustering algorithm on a subset of the graph's nodes rather than the whole graph. In short, it starts with an initial node called the seed and examines all nodes in the neighbourhood (ignoring the rest of the graph's nodes), then repeats the process with different seeds to produce a good cluster. This method is helpful in the case of massive networks, as it saves computation time compared with global clustering. Moreover, most clustering methods use the adjacency matrix or the edges list, and storing such data is hard for massive graphs, for example, the world wide web network which includes billions of nodes and edges. In this case, global clustering is much harder as the graph could grow and expand to a large size. However, local clustering is on the way an approach to find global clustering. This is achieved by rerunning local clustering several times, each time with a different seed node. Then, using quality measures to merge the outputs of the local clustering to give the global output. There is another useful way to find global clustering by local methods; by choosing a set of initial nodes (seeds) based on special features, for example, node degree. A global-via-local approach makes a huge difference in the time complexity

compared to global clustering. In this research, we focus on the global clustering approach to cluster undirected unweighted graphs.

In graph theory, there are different methods which are designed to cluster nodes in graphs, called *graph partitioning methods*. The main idea of these methods is to divide the graph's nodes into k clusters of predefined equal size n/k,and keep the number of edges between clusters minimal. One of the earliest methods is the Kernighan-Lin algorithm (Kernighan and Lin, 1970), which has important applications in physical situations, such as partitioning electronic circuits into circuit boards. Its aim is to minimize the connections between boards. The Kernighan-Lin algorithm divides the graph's nodes into two equal size groups, and optimizes the partitions with respect to a benefit function Q. It is the difference between the sum of the edges within a cluster and the sum of edges between clusters. Suaris and Kedem (1988) produced a generalization of the Kernighan-Lin algorithm to produce any number of partitions in a graph. Computation costs increase with the number of partitions.

Another common method introduced by Ford and Fulkerson (1956) is the max-flow min-cut method. The partition idea is based on keeping the maximum flow within each group and minimum flow between groups. The minimum cut edges between any two vertices v_i and v_j in graph G have a maximum flow between v_i and v_j in G.

Although graph partitioning methods are simple and quite fast, they are not a preferred tool to detect communities in graphs. This is because these methods assume known groups sizes and the number of groups.

Therefore, most researchers prefer the traditional clustering methods which have fewer requirements than *partitioning methods*, such as, *hierarchical clustering*, *partitional clustering* and *spectral clustering*.

Partitioning clustering is one of the most popular clustering strategies in graph theory. It is based on two essential inputs: the predefined number of clusters k and the similarity/dissimilarity distance between the graph's nodes. Briefly, it partitions the nodes into k groups, the groups' nodes close to each other more than other group's nodes, and optimizes this partition based on a defined quality measure. For example, minimum k-clustering, keeps the diameter of each cluster (the maximum distance between pair of nodes in same cluster) as small as possible.

The K-clustering sum, keeps the average distance between all pairs of nodes in each cluster as small as possible. The K-centre and k-median (or k-medoids) define a representative node for each cluster (centre or median). K-centre keeps the maximum distance within a cluster between any node and the cluster's representative as small as possible. While in k-median, it keeps the average distance within a cluster between any node and the cluster's representative as small as possible. More details can be found in Everitt et al. (2011); Fortunato (2010); Schaeffer (2007).

There is also an algorithm for detecting *communities* in graphs introduced by Girvan and Newman (2002). It relies on the edge betweenness centrality concept, where the edge betweenness of an edge e is the number of shortest paths between any pair of nodes that pass through the edge e. Therefore, the edges between communities in a graph share a large number of shortest paths in the graph and subsequently have higher betweenness than the edges within communities. Girvan and Newman exploited this idea to find the communities in graphs by removing the highest betweenness edge and recalculating the edges betweenness for the sub edges affected by the removal. The algorithm continues removing until all edges disappear or a predefined number of clusters is achieved.

Rattigan et al. (2007) tested two clustering algorithms in graphs: the k-medoids algorithm, and the Grivan-Newman algorithm. They showed that both algorithms have a high complexity in large graphs, thus, they introduce a scalable technique designed to capture the graph structure called network structure index (NSI). NSI can discover the geodesic distance (shortest paths SP) between graph nodes faster when combined with a search method. By adapting the k-medoids algorithm and Grivan-Newman algorithm to incorporate the NSI technique, their experimental results showed improvements in the time complexity for both clustering algorithms.

4.3.1 K-means clustering

K-means clustering algorithm is one of the oldest methods in cluster analysis. Although its first appearance was in the 1950's, it is still one of the most commonly used methods, and has a rich history in the literature as it has been applied in various scientific areas. Three ingredients are required in the k-means algorithm: number of clusters k, initial centroids $\{c_1, c_2, ..., c_k\}$ and distance metric d (Jain, 2010).

Given a graph $G(V, E)$, the algorithm below summarizes the algorithm steps on graphs with d:

Algorithm 2. K-means clustering $G(V, E)$

1: Compute the $(n \times n)$ distance matrix.
2: Select k nodes randomly as initial centroids $v_1^*, \ldots, v_k^*$ for clusters $C_1, \ldots, C_k$.
3: Allocate each node v_i to cluster C_k, where $k = arg\min_j d(v_i, v_j^*)$, $i = \{1, \ldots, n\}, j = \{1, \ldots, k\}$.
4: **repeat**
5: Allocate each node v_i to C_k if $k = arg\min_l \frac{\sum_{v_j \in C_l} d(v_i, v_j)}{|C_l|}$
6: **until** convergence criterion is met no change in clusters members.

We start the algorithm by computing the distance matrix d_{BTD}, then choose k initial centroids randomly. To form the clusters, we assign each node v_i to the cluster C_k which satisfies the condition in 3. After the first allocation, we repeat allocating all nodes to their fitted clusters until no nodes change their clusters. This differs from the original k-means algorithm in Euclidean space where the points have coordinates and it is possible to calculate centroids in each iteration. In graphs, the nodes do not have coordinates, and in this case, we can not recompute the centroids in each iteration.

The most crucial step in the k-means algorithm that can affect its performance is the selection of the initial centroids. The random selection of initial starts often leads to very different clustering solutions. This is because, k-means can only converge to local minima, and this problem increases if the dataset structure does not have natural clusters. At best, we can repeat the algorithm for different sets of initial centroids. Then, either evaluate the results by subjective choice (in small data sets it may be possible to choose the solution which has obvious clusters by eye), or choose the clusters solution which has a minimum squared error between the nodes. Given a graph of size n, $V = \{v_1, v_2, \ldots, v_n\}$, and k-means clustering result $C = \{C_1, C_2, \ldots, C_k\}$, the sum of squared errors (SSE) is given by:

$$SSE(C) = \frac{1}{2} \sum_{l=1}^{k} \sum_{v_i, v_j \in C_l} d(v_i, v_j)^2, \tag{4.1}$$

where k is the number of clusters. More details are given in Aggarwal and Reddy (2014) and Jain (2010).

In our study, we did a simulation study of clustering four different graph models using the k-means clustering algorithm, the goal of this simulation is to compare the performance of the distances d_{BTD} and d_{SP}. The results are presented in Section 4.5.3.

4.3.2 Choosing initial centroids for the k-means clustering algorithm in Euclidean space

Applying k-means clustering, and repeating the algorithm for different sets of initial centroids could produce different solutions. This issue becomes more difficult in large graphs, as running the algorithm consumes a lot of time. This is because we are re-run the algorithm for each set of initial centroids, and some of these sets take more time (than other sets) to converge.

Various methods have been proposed to select initial centroids in the k-means algorithm in order to improve the performance of the method, either in the accuracy or the speed.

The first approach was introduced by MacQueen (1967), and his idea is based on random selection of starting points, updating the centroids after assigning each point. This means after assigning the first point to the nearest centroid, the algorithm updates the centroids, before assigning the next point to the nearest centroid. This idea is extremely expensive due to the computational complexity.

A simple cluster seeking method was suggested by Tou and Gonzalez (1977). This method depends on the objects order, where the first point in the dataset is selected as the first initial centroid. Then, it calculates the distance between the chosen point and the second point in the dataset. If the distance is greater than a threshold (selected by the user), the point is chosen to be the second centroid. If the distance does not meet the condition, it switches to the next object in the data set, and iterates, until the algorithm selects the second centroid. After that, it calculates the distances from the first and second initial centroids to the next object in the dataset. It chooses this point as the third initial centroid if the distances are greater than the same threshold. It repeats the steps until k centroids are successfully selected. Two drawbacks are clearly apparent: first, it depends on the dataset order, and second, it depends on the threshold as a subjective choice by the user.

Katsavounidis et al. (1994) suggested a method called the KKZ algorithm. It selects a sequence of initial centroids such that the first centroid is chosen from the edge of the data, and each new centroid is furthest from the previous centroids, until k seeds are finally selected. The difficulty of this method is with outlying points, which are usually selected as a centroid, but might be far away from other nodes which makes the SSE higher.

Cao et al. (2009) propose a method to select initial clusters' centroids in the k-means algorithm based on the neighbourhood model. The method computes the cohesion degrees of the data points according to the set of points within distance ε (where ε is a neighbourhood size parameter) from each point. The idea of the method is to select the point with the maximum cohesion degree as a first initial centroid, and select the rest of the initial centroids based on the order of cohesion degrees. The method achieves good results in some cases compared with random initial selection, but as the selected parameter ε is subjective, it has a large effect on the k-means performance.

Erisoglu et al. (2011) introduced an algorithm to compute initial cluster centres for the k-means algorithm in high dimensional data. Their idea is based on shrinking the data dimension to two variables which best describe the data structure. The first variable has the largest variation coefficient among all variables, and the second variable has the minimum correlation coefficient with the first chosen variable. After distributing the dataset in two axis space, we calculate the mean position of the data points, and select the furthest point from the mean to be the first initial centre. The algorithm selects the rest of the centres by computing the cumulative distance from each remaining point to the previously selected centres, then choosing the point with the maximum cumulative distance.

Baswade and Nalwade (2013) proposed a method of choosing initial centroids for the k-means clustering algorithm in some datasets. It is based on choosing the first initial centroid as the average attribute values of n objects, and for the rest of the centroids, the algorithm chooses the node with maximum Euclidean distance to the previous centroids. This idea does not work well with large k, and it is not suitable for some data types: for example, categorical data.

Bansode and Bhusare (2014) proposed an approach to optimize the initial centroids for the k-means clustering algorithm in 2-dimensional feature space. Their idea is based on selecting

the initial centroids as far apart as possible. They rely on the idea of designing a building and separating the pillars separated from each other to have high protection against the next floor pressure. They start by computing the grand mean of the data points, which is calculated as the gravity centre of the data distribution, then calculate the distance between each point and the grand mean. The point with maximum distance is chosen to be the first initial centroid c_1, and the point with maximum distance to the first centroid is chosen to be the second centroid c_2. To select the rest of centroids, Bansode and Bhusare calculate the accumulated distance $d(c_1, c_2) + d(c_2, x)$ and the point x with highest cumulative distance is selected as the next centroid.

Bansode and Bhusare set a mechanism to check for outliers in the chosen initial points, by counting the neighbour points within the neighbourhood boundary. A threshold parameter β is determined as the maximum distance between a selected initial centroid x and the rest of the points. If the number of x's neighbours inside the area y (which is determined by β) is lower than a suggested number, x is considered as an outlier. Bansode and Bhusare method has a drawback as the parameter β is subjective choice which affects the choice of initial starts.

4.3.3 Adaptive k-means clustering in graphs

In this section, we consider ways of choosing initial centroids in graphs, with the aim of producing solutions as good as the best solution from repeated random starts, and at the same time avoiding the time consuming process of repeating k-means for different random initial centroids.

We did experiments to adapt k-means algorithm to be faster than the original version by applying two methods of choosing initial centroids at the first step of the algorithm, then choosing the better solution over these two methods by using the modularity measure of Section 4.4. Illustrative examples are also given which shown the differences in results between clustering with totally random initial centroids and totally deterministic initial centroids by our methods.

Method 1

The idea behind this method is to choose well-separated initial centroids from a graph, which means maximizing the sum of distances between the initial centroids. This idea may seem similar to Katsavounidis et al. (1994) and Bansode and Bhusare (2014), but it has significant differences. Firstly, it is applied on graph data using our BTD, while the previous methods are designed for data in Euclidean space. Secondly, it does not depend on the cumulative distance as in previous methods, instead, it uses just the BTD matrix to find each initial centroid. This saves time and space especially in the case of determining a large number of initial centroids. Thirdly, It chooses the first and second centroids together. While the previous methods choose first centroid from the edge of the data then choose the second centroid in the next iteration, which could affect by the outliers. However, in graphs, there is no clear meaning of the outliers as some leaf nodes (which have degree 1) may be in the edge of a branch or could have links to the high-degree nodes. To describe method 1, we show a simple example with the algorithm steps.

For example, if $k = 3$, we have to choose 3 initial centroids c_1, c_2, c_3, such that the distance between c_1 and c_2 is the maximum distance between any pair of nodes in the graph. Then, we choose c_3 which has the maximum of minimum distances between c_1, c_2 and the rest of nodes. This method allows us to permute between nodes if several nodes are tied for maximum distance.

Algorithm steps in $G(V, E)$

- Calculate the BTD distance matrix (from Chapter 3).
- Select the element of the BTD matrix with the maximum value. Then, choose the nodes associated with this element to be the centroids c_1 and c_2. If there is more than one element that satisfies this condition, pick one pair randomly.
- Produce a $(n \times 2)$ matrix M which consists of c_1 and c_2 columns from the BTD matrix.
- Calculate new vector which is the minimum value over the first and second columns of matrix M.

- Choose the third initial centroid c_3 from the $n-2$ remaining nodes which corresponds to the maximum value from the vector in the previous step.
- Add c_3 column from the BTD matrix to matrix M.
- Continue choosing initial centroids with same strategy of choosing the next centroid until there are k initial centroids.

Method 2

Method 2 is based on choosing the super nodes (which are the maximum degree nodes) in the graph as the initial centroids in the k-means algorithm.

This idea began with an investigation of the power of the super nodes in graphs. They play a role similar to the gravity as they have connections with a lot of nodes, and often, they have short distances to the nearby nodes. So, we thought that the super nodes seems to construct communities with their neighbours. However, in some graph models, the super nodes occur near the centre of the graph. So, choosing the super nodes as the initial centroids helps the algorithm in detecting communities easily.

If more than the required number of centroids have the same degree, this method allows us to choose randomly between the super nodes.

In our study, we use both Method 1 and Method 2, and choose the best clustering result based on the quality of the results.

4.3.4 Hierarchical clustering

Hierarchical clustering (HC) is one of the most popular methods in graph theory, and as mentioned in Section 4.2, it is divided into two strategies: agglomerative algorithms and divisive algorithms. It represents the clustering results by a dendrogram where the leaves correspond to the graph nodes. It has a special feature distinguishing it from the rest of clustering methods that it produces multi-level clustering. Each level produces different clusters, and each higher level

cut produces of a subset of clusters from the lower level structure. This feature makes HC common in graph clustering of real networks as they have a hierarchical structure or communities. Examples exist in social, biology, marketing networks, etc.

Before choosing one of the strategies, we have to define the similarity measure (or dissimilarity measure) between nodes, usually, an $n \times n$ similarity matrix. In our case we use the BTD matrix. The closer nodes tend to be in the same cluster. This idea achieves the aim of HC algorithms which detect the gathering or communities in graphs. Next, we choose one of the clustering techniques. We choose the agglomerative strategy in our study, as most of the work in the literature is based on it. However, as the clusters merge from the lower level, there are different criterion to merge the clusters; each one estimates the similarity between clusters in different way.

- First, single linkage which defines the distance between any two clusters as the minimum distance between their elements.
- Second, complete linkage which defines the distance between a pair of clusters as the maximum distance between their elements.
- Third, average linkage considers the average distance between two group's nodes as the distance between clusters.

However, the final step of the algorithm should determine the best level to cut the branches, and this is under the control of either predefined number of clusters or optimization by a quality function for the entire clusters (Fortunato, 2010; Schaeffer, 2007).

In this study, we apply agglomerative hierarchical clustering with complete linkage, as it more likely to produce compact clusters than other methods, on 4 different graph models and show a comparison between both distances: SP and BTD.

4.4 Clustering results validity

The evaluation of the quality of a cluster is one of the most critical tasks in cluster analysis. As one of the clustering goals is exploring the latent structure of a graph, high-quality clustering results can describe the communities in the underlying graph. However, Bonner (1964) argued that there is no single unique measurement to check the clusters quality. In the case of graphs which can be easily visualized by the researcher, the evaluation is often subjective.

Aggarwal and Reddy (2014) summarized the common measurements of clustering quality using graph clustering algorithms. The *Mincut* approach is one of the simplest measures of the clusters quality. Given a graph $G(V, E)$, and clusters $C_1, ..., C_k$, and define the function $W(A, B) = \sum_{i \in A, j \in B} w_{ij}$, where W is the adjacency matrix of the graph G. The *Mincut* approach nominates the clusters which minimize the cut

$$\text{cut}(C_1, C_2, ..., C_k) = \frac{1}{2} \sum_{i=1}^{k} W(C_i, \bar{C}_i), \tag{4.2}$$

where $\bar{C}$ is the complement of C.

The problem with the cut measure is that it can produce a minimum value even if the clusters are undesirable. For example, in case of $k = 2$, cut reaches the minimum if one node is in C_1 and the rest of nodes are in C_2. In this situation, it does not provide any useful information about the network structure. To deal with such results, the cut approach has been given two improvements: first, *Ratiocut* which was introduced by Hagen and Kahng (1992) and uses the cluster sizes to weight the distances in Expression 4.2. Second, the normalized cut *Ncut*, which was proposed by Shi and Malik (2000) and weights the clusters by the clusters degree. They are given by

$$\text{Ratiocut}(C_1, C_2, ..., C_k) = \frac{1}{2} \sum_{i=1}^{k} \frac{W(C_i, \bar{C}_i)}{|C_i|} = \sum_{i=1}^{k} \frac{\text{cut}(C_i, \bar{C}_i)}{|C_i|}, \tag{4.3}$$

where $|C_i|$ is the size of cluster C_i, and

$$\text{Ncut}(C_1, C_2, ..., C_k) = \frac{1}{2} \sum_{i=1}^{k} \frac{W(C_i, \bar{C}_i)}{\text{vol}(C_i)} = \sum_{i=1}^{k} \frac{\text{cut}(C_i, \bar{C}_i)}{\text{vol}(C_i)}. \tag{4.4}$$

where vol(C_i) is the total nodes degree in C_i .

The objective of these improvements is to ensure that clusters are balanced in their sizes, either by the number of nodes or the number of edges in each cluster in Expressions 4.3 and 4.4 respectively.

Kannan et al. (2004) differentiated between high and low degree nodes by introducing a generalization of the normalized cut Ncut called *conductance*. This gives more weight to clusters which consist of high-degree nodes. The conductance of a cluster C in G is defined by:

$$\text{Conductance}(C) = \frac{\sum_{i \in C, j \in \bar{C}} W(i,j)}{min(\sum_{i \in C} \text{vol}(i), \sum_{i \in \bar{C}} \text{vol}(i))}, \tag{4.5}$$

and the Conductance of a graph G is the minimum conductance over all clusters in G:

$$\text{Conductance}(G) = \min_{C \subseteq G} \text{Conductance}(C).$$

Another popular quality function for measuring the goodness of network partitions is the Modularity function introduced by Newman (2006). The idea of this approach is that most random networks do not have clear communities in their graph structure. It assesses partition quality based on the differences between the number of edges within the clusters, compared to the expected number if there is no structure. It can be either positive or negative and we look for clusters with high modularity as an indicator of good partitions. It can be written as

$$Q = \frac{1}{4m} \sum_{i,j} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j)$$

where

- k_i is the degree of the vertex i.
- $m = |E|$ is the total number of edges in the graph.
- $\frac{k_i k_j}{2m}$ is the expected number of edges between vertices i and j if edges are placed at random (null model).
- The function $\delta(c_i, c_j)$ is equal to 1 if i and j are in the same cluster and 0 otherwise.

In our study, we use the modularity measure as an optimization function of the quality of our clustering results. The reasons behind our choice are that, the modularity function is widely used by most academic researchers in cluster analysis as a best measure of the goodness of partitions (Fortunato, 2010; Clauset et al., 2004; Blondel et al., 2008; Song and Zhao, 2014). Moreover, the Modularity function is quite a simple tool and faster than most of the available quality measures even for large and sparse networks (Clauset et al., 2004).

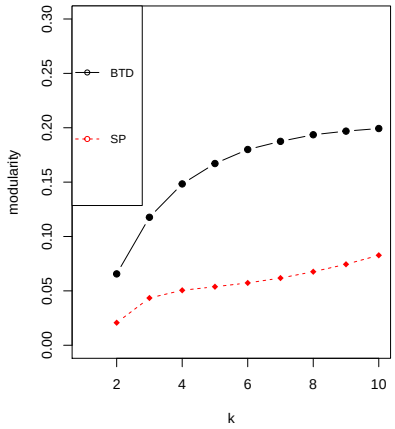
4.5 Experimental results

4.5.1 Model selection

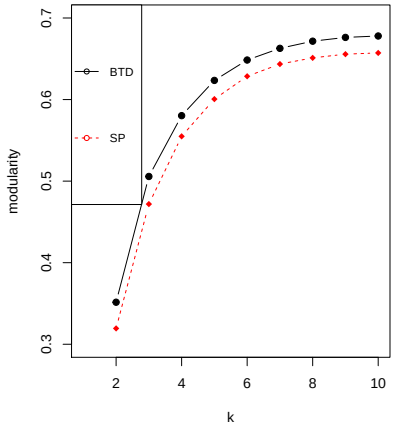
To test our ideas we simulate graphs from four different graph models which are disparate in shapes and properties. We chose 4 models to create our graphs from Csárdi and Nepusz (2006). They are all simple undirected graphs without loops and multiple edges. The BA model is a tree shape graph model, and the WS, ER, and FF graph models have multiple paths between nodes. A full description was given in Chapter 2.

4.5.2 Hierarchical clustering results: BTD vs SP

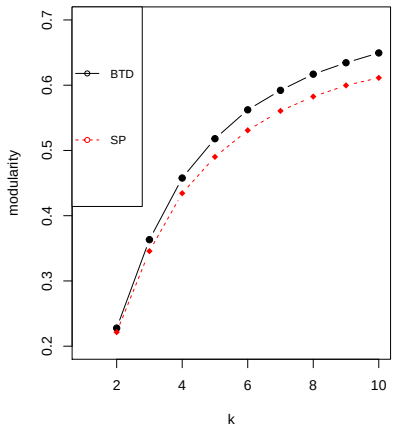
In this section, we compare results of hierarchical clustering for different graph models using BTD and SP. We achieved these results by doing practical experiments and simulating 1000 graphs of size 100 from each proposed model. Then, we applied hierarchical clustering (complete link) to each graph with each distance measure, BTD and SP. The clustering results cover different numbers of clusters $k = \{2, 3, \ldots, 10\}$. Plots of the results are in Figure 4.1. The results show obvious improvements in BTD results compared with SP results in all graph models, which is evidence of the efficiency of BTD in hierarchical graph clustering. In particular, the plot of the ER model has a larger difference. This is due to the structure of the ER graph model which is considered as random in edge distribution and has a dense graph structure, which means a lot of ties in distances. Therefore, we conclude that BTD is much better than SP in dense graphs, and most of the real networks are dense and large graphs.



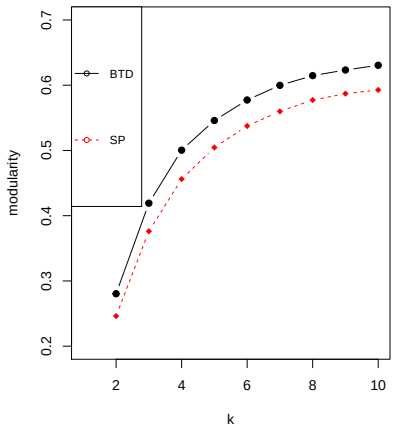
(a) ER model



(b) WS model



(c) BA model



(d) FF model

Figure 4.1: Plots of the average modularity Q (vertical axis) of HC (complete linkage) using BTD (black line) and SP (red line) over 1000 simulations of each graph model, with different number of clusters $K = 2, 3, \ldots, 10$ (horizontal axis).

4.5.3 K-means clustering results: BTD vs SP

In this section, we compare simulation results of K-means clustering with different graph models between BTD and SP. We achieved these results by doing practical experiments and simulating 50 graphs from each proposed model; each graph has $n = 200$ nodes. Then, we apply k-means clustering for 100 different sets of centroids (randomly selected) for different numbers of clusters, $k = \{5, 7\}$ in each simulated graph. These k numbers are chosen subjectively based on similar previous studies and the ratios between n and k.

Note that in the `R` code, we use a technique to detect any duplication in k-means iterations as it sometimes does not converge to one solution. The detecting technique controls k-means algorithm and allows to converge after detecting the duplication especially in the case of SP which does not distinguish between ties.

Rattigan et al. (2007) mentioned the same issue as some nodes in graph clustering will have equal distance to more than one centroid and the algorithm will never converge. They try to deal with this problem by random assignment for these nodes. However, to avoid this problem, they consider a threshold value, usually $1 - 3\%$, that stops the algorithm if the changes between iterations are below this threshold.

Table 4.1 shows the simulation results for the four models. In each simulation, we calculate the modularity measure for each clustering result for all 100 random initial centroids and choose the maximum modularity over these 100 modularities. Over 50 simulations, we therefore obtain 50 modularity measures. In Table 4.1, *max*, *min*, and *avg* correspond to maximum, minimum and average values over the 50 modularity measures. *Avg Itr* is the average iteration number of the k-means algorithm over 50 simulations. The *avg* time is the average time taken over all 50 simulations. All these measurements are calculated for both distances: BTD and SP. *Length 1* is the average number of BTD clustering results which have modularity measures larger than the maximum modularity measure of SP clustering results for all 50 simulations. *Length 2* is the average number of SP clustering results which have modularity measures less than the minimum modularity measure of BTD results over all 50 simulations. *Length 1* and *length 2* show the efficiency of BTD in producing better quality clustering results than SP.

Table 4.1 shows that for the ER model, WS model, and FF model, the BTD produces slightly higher quality results compared with SP using the modularity function. This is because that BTD is more clever than SP in choosing the nearest centroid for each node in k-means iteration. This is clear from the first three measurements: *max*, *min*, and *avg*. For the next two measurements: *avg itr* and *avg time*, the results are slightly different. As BTD breaks the ties between equal shortest paths, it records more iterations on average than SP for ER, WS, and FF models, while it records similar iteration numbers for the BA model due to its tree structure. These results will affect the *avg time* measurement for the ER, WS, and FF models using BTD, which tends to take more time than SP. However, in the BA model, the time taken is approximately equal between BTD and SP.

It is clear that *length 1* and *length 2* are affected by the graph structure. In the ER graph model, which has dense graph structure and large number of ties between equal SP, *length 1* and *length 2* are higher in ER model than the other models with approximately 13 and 19 for $k = 5$, and 21 and 35 for $k = 7$, respectively. These averages decrease from the WS model to the FF model, and reach the minimum in BA model due to fewer edge ties in the graph models. In the BA model, the results are similar between BTD and SP because the graph model has a tree structure with a lack of ties between nodes. This means that BTD is particularly efficient and highly recommended in dense graphs.

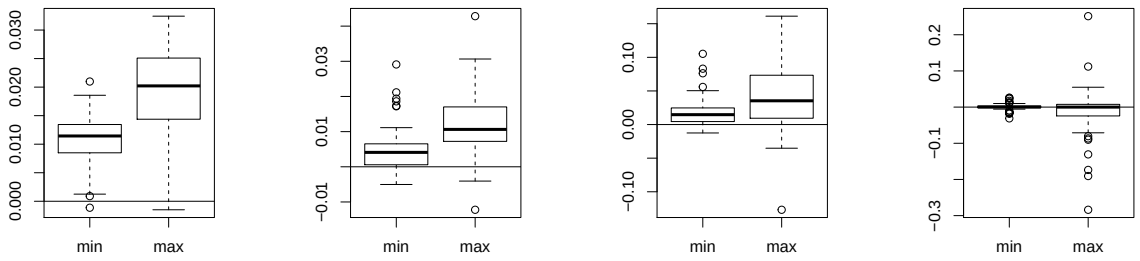
We also applied a paired t-test on the simulation results to check if the mean difference between BTD and SP results is zero:

$$H_0 : \max(Q_{\text{BTD}}) = \max(Q_{\text{SP}})$$
$$H_1 : \max(Q_{\text{BTD}}) \neq \max(Q_{\text{SP}}),$$

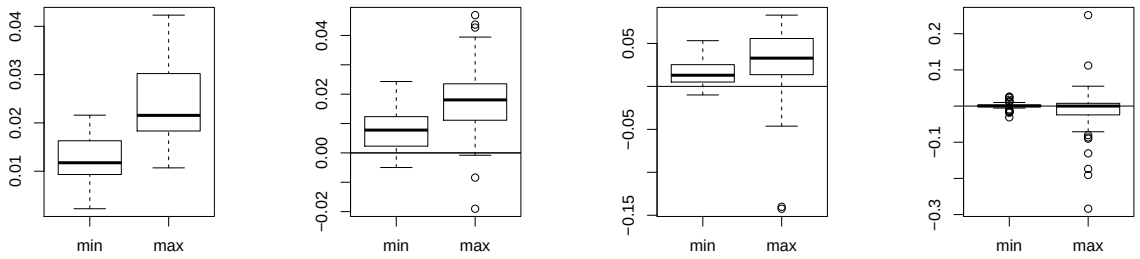
where the maximum is taken over the 100 random starts. The p-values of the paired test are less than 0.05 for the ER, WS, and FF model models, which indicates a significant difference between the distance results. For the BA model, the p-value reaches 0.05 which means both BTD and SP produce similar results, and this result is expected as the BA graph model has tree structure with no real ties between nodes.

Figure 4.2 shows box plots of the differences between maximum/minimum modularity values between BTD and SP over 50 simulations; each figure corresponds to one model and one

value of k. We can see that in ER model, all differences are above the zero horizontal line for $k = 7$, and approximately more than 90% of the differences are above the zero horizontal line for $k = 5$. In WS and FF model, more than 75% of the differences are above the zero horizontal line for $k = 5, 7$. In the BA model, 50% of the differences are above the zero horizontal line for $k = 5, 7$ all graph models except the BA model, most of the differences are larger than zero in most simulations. This is evidence that BTD generally produces higher cluster quality than SP.



(a) ER model, k=5 (b) WS model, k=5 (c) FF model, k=5 (d) BA model, k=5



(e) ER model, k=7 (f) WS model, k=7 (g) FF model, k=7 (h) BA model, k=7

Figure 4.2: Box plots of the differences of maximum/minimum modularity values between BTD and SP over 50 simulations of all proposed graph models of size $n = 200$, $k = 5, 7$ and 100 initial starts sets in each simulation. The horizontal line crosses at zero to show the positive differences.

From Table 4.1 and Figure 4.2, we conclude that for dense graphs, BTD produces higher quality clusters than SP when assessed by the modularity function Q. This makes BTD a more preferable distance measure in dense graphs than SP. Note that BTD is less good when the graphs are tree-like.

	ER model				WS model			
	$k=5$		$k=7$		$k=5$		$k=7$	
Distance	BTD	SP	BTD	SP	BTD	SP	BTD	SP
Max	0.35	0.33	0.35	0.33	0.70	0.70	0.75	0.73
Min	0.31	0.29	0.31	0.30	0.64	0.63	0.68	0.66
Avg	0.33	0.32	0.33	0.31	0.67	0.66	0.70	0.69
Avg Itr	13.6	12	13.1	13	12.2	13.2	13.2	12.8
Avg time	13.3	12	13.1	11.9	13.8	13.2	14.4	13.3
length 1	13.4		21.46		3.24		6.24	
length 2	19.16		35.02		4.46		6.76	
	BA model				FF model			
	$k=5$		$k=7$		$k=5$		$k=7$	
Distance	BTD	SP	BTD	SP	BTD	SP	BTD	SP
Max	0.78	0.78	0.82	0.82	0.57	0.55	0.58	0.56
Min	0.67	0.68	0.77	0.77	0.32	0.32	0.26	0.27
Avg	0.75	0.75	0.80	0.80	0.46	0.44	0.45	0.43
Avg Itr	6.5	6.4	6.7	6.6	10.4	8.7	12.04	8.7
Avg time	7.6	7.3	8.7	8.3	1.35	1.39	1.57	1.38
length 1	0.42		0.38		3.34		3.38	
length 2	0.54		0.48		2.96		3.38	

Table 4.1: Table of statistics to compare the efficiency of BTD and SP over four different graph models with the k-means algorithm for $k = 5$ and 7. *Max*, *min*, and *avg* correspond to maximum, minimum and average values over the 50 modularity measures. *Avg Itr* is the average iteration number of the k-means algorithm over 50 simulations (in each simulation we choose the number of iteration which corresponds to the maximum modularity clustering results among 100 random sets of initial centroids). The *avg time* is the average time taken over all 50 simulations. *Length 1* is the average number of BTD clustering results which have modularity measures larger than the maximum modularity measure of SP clustering results for all 50 simulations. *Length 2* is the average number of SP clustering results which have modularity measures less than the minimum modularity measure of BTD results over all 50 simulations.

In this simulation study, we have different parameters to deal with; number of simulations, graph size, the number of clusters and the models parameters. We carefully select some values from each parameter which could cover a range of cases.

4.5.4 Clustering results by BTD: classic k-means vs adaptive k-means

In this section, we present the simulation results of the proposed adaptive k-means clustering which we discussed in Section 4.3.2 compared with classic k-means using BTD for five different models of graphs, which are disparate in shapes and properties (described in Section 4.5.1).

These proposed algorithms for choosing initial centroids in k-means clustering are efficient in the experimental results for two reasons. First, they converge to acceptable clustering results compared with the classic k-means algorithm (using the modularity function Q). Second, they converge faster than the random choice of centroids. Moreover, the most important feature for our deterministic choice of initial centroids is that it is simpler and easier to implement, unlike the previously proposed algorithms in the literature.

The results in Table 4.2 show a comparison between the efficiency of Method 1 in Section 4.3.3, Method 2 in Section 4.3.3 and the maximum of these two methods, with the classic k-means algorithm, which uses random assignment of initial centroids. In each comparison we simulate 100 different graphs from each model. Our parameters in this study are: the graph size n, and the number of clusters k, and our settings in each simulation are ($k = 3, n = 50$), ($k = 3, n = 200$), ($k = 5, n = 50$) and ($k = 5, n = 200$). These settings are chosen subjectively to cover two different size of graphs with two different number of clusters, where the ratios between n and k are balanced. Moreover, the chosen values of k are similar to the suitable number of classes in the Facebook network that we analyse it later in the following section.

We use the following notation in Table 4.2: $\bar{Q}_{10}, \bar{Q}_5$ are the averages of the modularity function over $10, 5$ random initial centroids sets, $\bar{m}_1, \bar{m}_2, \bar{m}_{\max}$ are the averages of the modularity function of Method 1, Method 2 and the maximum of methods 1 and 2, N is the average number of random choice solutions below the result with maximum modularity using both m_1 and m_2. This measurement allows us to imagine how many trials we could make until we

ER model	$k=3, n=50$	$k=3, n=200$	$k=5, n=50$	$k=5, n=200$
$\bar{Q}_{10}$	0.33	0.17	0.35	0.18
$\bar{Q}_5$	0.32	0.17	0.35	0.18
$\bar{m}_1$	0.30	0.16	0.33	0.18
$\bar{m}_2$	0.31	0.16	0.33	0.17
$\bar{m}_{max}$	0.32	0.17	0.34	0.18
N	2.75	4.56	3.59	3.99
M	2.08	2.66	2.63	3.45
BA model	$k=3, n=50$	$k=3, n=200$	$k=5, n=50$	$k=5, n=200$
$\bar{Q}_{10}$	0.55	0.58	0.64	0.70
$\bar{Q}_5$	0.54	0.57	0.63	0.69
$\bar{m}_1$	0.48	0.51	0.59	0.66
$\bar{m}_2$	0.55	0.57	0.62	0.67
$\bar{m}_{max}$	0.54	0.58	0.63	0.69
N	2.49	3.81	5.11	5.97
M	0.89	0.72	1.78	1.3
WS model	$k=3, n=50$	$k=3, n=200$	$k=5, n=50$	$k=5, n=200$
$\bar{Q}_{10}$	0.49	0.52	0.55	0.61
$\bar{Q}_5$	0.48	0.51	0.55	0.60
$\bar{m}_1$	0.47	0.50	0.53	0.59
$\bar{m}_2$	0.47	0.51	0.54	0.59
$\bar{m}_{max}$	0.48	0.51	0.54	0.60
N	4.34	6.97	5.26	4.74
M	1.92	1.66	2.28	2.8
FF model	$k=3, n=50$	$k=3, n=200$	$k=5, n=50$	$k=5, n=200$
$\bar{Q}_{10}$	0.40	0.30	0.44	0.42
$\bar{Q}_5$	0.39	0.29	0.43	0.41
$\bar{m}_1$	0.35	0.25	0.41	0.36
$\bar{m}_2$	0.37	0.26	0.42	0.37
$\bar{m}_{max}$	0.38	0.29	0.43	0.39
N	3.03	4.06	4.45	4.09
M	1.784	1.29	2.5	2.95
Mixture of FF	$k=3, n=50$	$k=3, n=200$	$k=5, n=50$	$k=5, n=200$
$\bar{Q}_{10}$	0.63	0.65	0.75	0.78
$\bar{Q}_5$	0.63	0.64	0.75	0.77
$\bar{m}_1$	0.62	0.64	0.74	0.76
$\bar{m}_2$	0.60	0.59	0.68	0.70
$\bar{m}_{max}$	0.63	0.65	0.75	0.77
N	1.15	1.58	2.35	2.74
M	0.36	0.06	0.94	0.55

Table 4.2: Table of performance between maximum modularity over the random starts and Method 1 (m_1), Method 2 (m_2) for different graph models, $\bar{Q}_{10}, \bar{Q}_5$ are the average modularity over 10 and 5 random initial centroid groups, $\bar{m}_1, \bar{m}_2, \bar{m}_{max}$ are the average modularity of Method 1, Method 2 and the maximum of Methods 1 and 2, N is the average number of random solutions needed to reach the $max(m_1, m_2)$, and M is the average number of solutions (from k-means)which have modularity above the $max(m_1, m_2)$.

achieve a result as good as the m_1 and m_2 result. M is the average number of k-means solutions which have modularity above the result with maximum modularity by m_1 and m_2. M measurement is a significant indicator of how much the random choices of centroids produce better clustering results than m_1 and m_2.

In general, our adaptive method gives high efficiency clustering results compared with the classic k-means algorithm as measured by the modularity measure, and it always runs in less time.

In the case of comparing between Method 1 and Method 2, Method 2 usually has better performance than Method 1 in the modularity measure. By contrast, Method 1 always takes less time to run than Method 2. This is because it converges more quickly due to the separation between initial centroids. Therefore, we recommend to use Method 1 as an adaptation of the k-means algorithm if the running time is highly important to the user. If the sensitivity in the result is more important, we recommend to use both Method 1 and Method 2 as an adaptation of the first step in the k-means algorithm.

In the ER graph model, the graph structure is produced by randomly distributing edges with a parameter to control the density of the edges in the graph. In this model, (as we mention in Chapter 2) the average shortest path is small compared with other models, so we expect that most of the nodes have edges connected with other nodes which makes the clustering and partitioning more difficult. Therefore, the clustering results have a small modularity measure, less than all other graph models. The performance of both m_1 and m_2 is approximately equal to $\bar{Q}_5$, and with larger n, it is equal to $\bar{Q}_{10}$. This means by increasing the graph size, we expect an improvement in m_1 and m_2 performance, and they might exceed the performance of $\bar{Q}_5$ and $\bar{Q}_{10}$. This is a significant indicator of the efficacy of Method 1 and Method 2 in large ER graphs. N values increase with n and k increasing, which means in large graphs and large k, we have to try more different random centroids sets in order to find better cluster solutions based on modularity measure. This leads to spend more time in the study, while following deterministic centroids can produce good solutions and will save time.

In the BA graph model, the graph has a tree structure and the nodes degree follow an exponential distribution. This means that there are a few nodes which have large degrees and

most of the nodes have small degrees. So, by applying Method 2, there is a small chance to permute between the chosen centroids, and we suggest to choose one set of centroids based on Method 2 in addition to one set of centroids from Method 1. From Table 4.2, we see that the $\bar{m}_{max}$, in most cases, reaches $\bar{Q}_5$ and in some cases $\bar{Q}_{10}$ which means that performance of our adaptive methods produce result as quality as the default k-means algorithm but is less time. There is a gradual increase of N with increasing in n and k, so we conclude that by increasing k and n we would need to try more than 5 groups of random centroids to get the result with maximum modularity by Method 1 and Method 2. From the M row which is the average number of solutions (from classic k-means) that have modularity above the $\bar{m}_{max}$ (it has a range of 0.72 to 1.78). It shows an evidence that if we want to reach the clustering result by $\bar{m}_{max}$, we have to try more than eight different random groups of centroids on average.

In this situation, time plays an important part in the decision. From Table 4.3, we see that the time taken for both Methods 1 and 2 ($\bar{T}_{m_1+m_2}$) is very small compared with the time of the k-means with 10 random sets of initial centroids $\bar{T}_{r10}$. The adaptive k-means uses on average about 10% to 17% of the average time of the original method over 10 different groups of random centroids and this decreases with increasing n and K.

In the WS graph model, the graph structure is nested (but less than the ER model) so there are no clear clusters or communities in the graph. However, we achieved good results with our adaptive method compared with the default k-means which are given in the third section of Tables 4.2 and 4.3. From Table 4.2, it is clear that $\bar{m}_{max}$ results reach $\bar{Q}_5$ most of the time specially in large size graphs which shows the efficiency of the adaptive methods. Also, the N values support the adaptive methods as it reaches, for example in the second column approximately 7. This means we have to try 7 different sets of random centroids until we got a result as good as the $\bar{m}_{max}$ result. From the M row, we can see that approximately 2 to 3 results from the default k-means which have modularity larger than $\bar{m}_{max}$ result. From Table 4.3, the time consumed in both methods m_1 and m_2 is less than half time of the time consumed by 10 random sets of centroids, and we show that this ratio decreases as the graph size increases.

By similar comparison in Table 4.2 for the FF graph model, there is a slight decrease in the modularity measure due to the density of the edges between nodes in the graph structure, and still the $\bar{m}_{max}$ can reach the $\bar{Q}_{10}$ in most situations. Also we allow to permute in choosing

ER model	$k=3, n=50$	$k=3, n=200$	$k=5, n=50$	$k=5, n=200$
$\bar{T}_{q10}$	0.066	1.385	0.158	1.773
$\bar{T}_{m1}$	0.005	0.132	0.013	0.160
$\bar{T}_{m2}$	0.006	0.122	0.016	0.164
$\bar{T}_{m1+m2}$	0.012	0.254	0.029	0.325
$\bar{T}_{m1+m2}/\bar{T}_{q10}$	0.185	0.184	0.186	0.183
BA model	$k=3, n=50$	$k=3, n=200$	$k=5, n=50$	$k=5, n=200$
$\bar{T}_{q10}$	0.07	0.98	0.24	1.72
$\bar{T}_{m1}$	0.005	0.045	0.014	0.075
$\bar{T}_{m2}$	0.007	0.06	0.019	0.11
$\bar{T}_{m1+m2}$	0.012	0.1	0.033	0.19
$\bar{T}_{m1+m2}/\bar{T}_{q10}$	0.178	0.106	0.14	0.11
WS model	$k=3, n=50$	$k=3, n=200$	$k=5, n=50$	$k=5, n=200$
$\bar{T}_{q10}$	0.09	1.89	0.27	3.03
$\bar{T}_{m1}$	0.007	0.13	0.021	0.25
$\bar{T}_{m2}$	0.034	0.62	0.116	1.13
$\bar{T}_{m1+m2}$	0.04	0.754	0.13	1.38
$\bar{T}_{m1+m2}/\bar{T}_{110}$	0.46	0.398	0.5	0.458
FF model	$k=3, n=50$	$k=3, n=200$	$k=5, n=50$	$k=5, n=200$
$\bar{T}_{q10}$	0.098	2.19	0.324	3.77
$\bar{T}_{m1}$	0.006	0.14	0.019	0.23
$\bar{T}_{m2}$	0.021	0.224	0.065	0.44
$\bar{T}_{m1+m2}$	0.028	0.365	0.085	0.67
$\bar{T}_{m1+m2}/\bar{T}_{q10}$	0.288	0.167	0.26	0.178
Mixture of FF	$k=3, n=50$	$k=3, n=200$	$k=5, n=50$	$k=5, n=200$
$\bar{T}_{q10}$	0.062	0.817	0.24	1.72
$\bar{T}_{m1}$	0.0045	0.036	0.012	0.073
$\bar{T}_{m2}$	0.009	0.074	0.031	0.2
$\bar{T}_{m1+m2}$	0.0148	0.11	0.043	0.27
$\bar{T}_{m1+m2}/\bar{T}_{q10}$	0.22	0.136	0.18	0.159

Table 4.3: Table of performance of k-means algorithm in time (in seconds) between the default k-means (repeated random initial centroids) and the adaptive k-means by Method 1, and Method 2 for five different graph models.

centroids by method 2 four times in with one set of centroids by method 1, so we test 5 different groups of centroids from the adaptive K-means with 10 and 5 different groups of centroids from the original method. From Table 4.3, the time consumed by our adaptive methods is much lower than $\bar{T}_{q10}$ which means that it saves huge time by using the adaptive choices even if we use more than one set of deterministic centroids. Also the time consuming when we use 5 different sets of initial centroids from the adaptive methods, is still less than the $\bar{T}_{q5}$. This makes our adaptive methods highly recommended in FF model specially in large graph sizes.

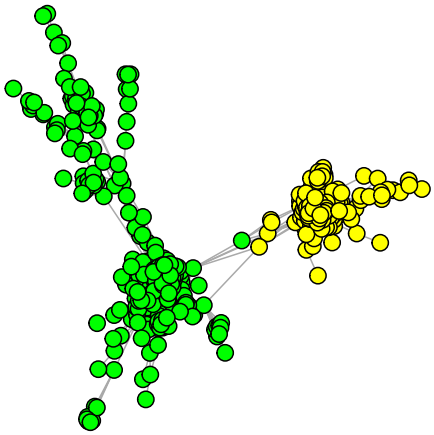
In the mixture model, the graphs are built from the FF model with clearly three and five (communities) clusters. It can be seen from Tables 4.2 and 4.3, that Method 1 always dominates Method 2 due to the graph shape and this clear in the modularity results. The reason for this is that constructing well separated clusters and choosing well separated nodes (m_1) as initial centroids could offer high chance to detect the correct clusters. While choosing centroids as the maximum nodes degrees (m_2) leads to a chance of choosing a couple of centroids from the same cluster which might lead to wrong separation. So, choosing well separated centroids in this model means that each centroid falls in a unique cluster, which leads to correct clusters and quicker convergence. In this model, there is less than one solution M from random selection centroids (on average) which has modularity measure higher than $max(m_1, m_2)$

Finally, we conclude that choosing the initial centroids by using our adaptive k-means algorithm is efficient based on the modularity measure and the time taken over all considered graph models.

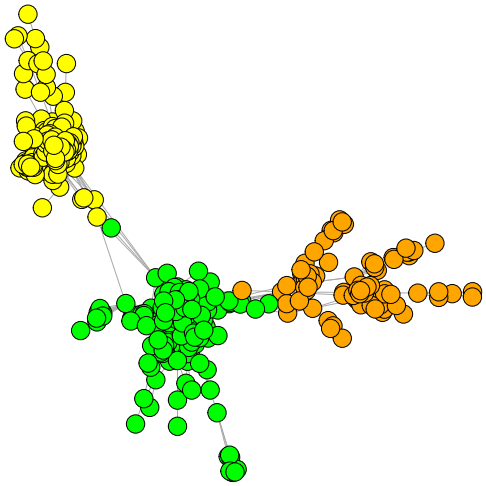
4.5.5 Facebook example

In this example, we compare the performance of BTD with SP in hierarchical and k-means clustering algorithms on a connected component consisting of two ego networks excluding the ego nodes (Leskovec and Krevl, 2014). It has 547 nodes and 5706 edges.

In Figure 4.3, we show the proposed Facebook network with two clusters as it consists from two egos, and with three clusters produced by the k-means algorithm and BTD. As well as the modularity results for HC using BTD and SP for $k = \{2, \ldots, 10\}$. The results show that



(a) Facebook with two clusters

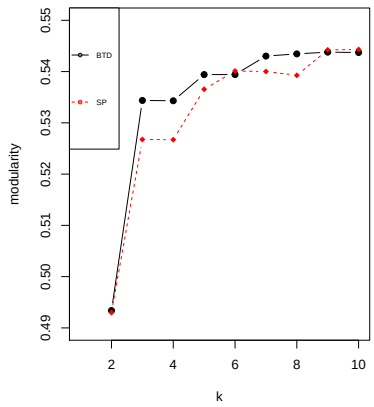


(b) Facebook with three clusters

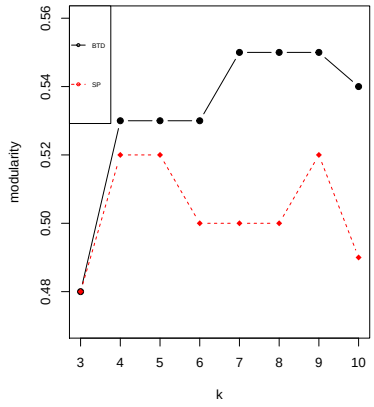
Figure 4.3: A Facebook graph example described by with two and three clusters.

BTD produces higher modularity than SP for the more important k values: in this network it appears that there are three natural communities, and for $k = \{3, 4, 5\}$, HC produces a higher modularity score for BTD compared with SP. For $k > 8$, both distance measures produce equal results, but the structural features have disappeared by this point. Even though Q reaches larger scores for large k, this is not a sign of a better number of clusters or communities, as the Q function is constructed to assess the cluster quality but not to choose the number of clusters (Fortunato, 2010).

We apply the k-means algorithm with different number of clusters $k = \{3, \ldots, 10\}$, and for each k, we choose 10 different random initial centroids sets and check the maximum and minimum over these 10 sets for each k. We compare the performance of BTD with SP in this experiment by running the same 10 sets with each distance function. Figure 4.4 illustrates the differences between BTD and SP using the minimum modularity scores over k. The results show a significant difference in favour of BTD; this means the arbitrary choice of initial centroids always has a smaller modularity score limit by SP than BTD. In the maximum comparison, both clustering methods behave similarly and produce a similar modularity score. From HC and k-means algorithm experiments, we conclude that BTD produces higher quality results compared with SP in the Facebook network.



(a) HC results



(b) Minimum K-means results

Figure 4.4: Comparison between BTD and SP in HC, and k-means clustering results in Facebook network.

4.6 Conclusion

This chapter focused on clustering in undirected and unweighted graphs. A review of the common clustering methods was discussed with a focus on k-means algorithm. A survey about choosing initial centroids methods was provided with new strategies of choosing initial centroids in graphs. We conducted experiments to compare the performance of BTD with SP over simulated graph models. Another study was also performed to check the performance of adaptive k-means with selected initial centroids, compared with classic k-means. The experiments show improvements in the clustering results by using BTD compared with SP, and also improvement in the clustering results and time taken by selecting initial centroids in k-means compared with the classic k-means algorithm. The clustering results for a real network from Facebook show improvements in both comparisons.

Further investigations could be done with different clustering algorithms to confirm the preference of BTD compared with SP on simulated graphs and different real networks.

In the next chapters, we continue our analysis by applying some machine learning algorithms such as classification and prediction.

Chapter 5

Graph nodes classification

5.1 Introduction

In this chapter, we focus on classification in graphs, and discuss some algorithms which are suitable for graphs in Section 5.3. In Section 5.4, we discuss the evaluation criterion of the classification result. We describe in detail four different classifiers: the nearest neighbour classifier, the fixed kernel, the adaptive kernel, and the support vector machine (SVM) with kernel trick classifier in Section 5.5. We notice that the kernel classifiers have received little attention in the literature of graph node classification. We also introduce a new strategy of approximating the piecewise constant error function to choose the smoothing parameter in Section 5.5.3. Another contribution in this chapter is that we introduce a new formula for choosing the smoothing parameter in an adaptive classifier based on the node degrees in Section 5.5.4. In Section 5.5.5, we modify our distance matrix by multidimensional scaling to represent the nodes as co-ordinates in low-dimensions, to be able to apply SVM methods on the graphs' nodes.

We also check in our experiments the efficiency of BTD compared with SP (BTD and SP are defined in Chapter 3) over these four classifiers in simulated graphs, and a real data network in Sections 5.6 and 5.9.

5.2 Classification for data mining

Data mining refers to the process of discovering knowledge from data. It is not limited to collection and clustering of data, it also goes further by analysing the pattern of the data, which helps in classification and prediction tasks. Most of the significant data mining applications involve classification and prediction.

Any classification task is a combination of three essential elements: first, the prior probability distribution, which is the relative frequency (probability) of each class. Second, the classification criterion or method, either implicit or explicit. Third, the misclassification cost or the error rate, which is used to evaluate the classification criterion and we are looking to minimize the cost of misclassification (Michie et al., 1994).

If the classification output has two different classes, the problem is called binary classification, while if the output has more than two classes, the problem is referred to as multi-class or multinomial classification.

Recently, classification has become a fundamental task in different fields: for example, in computer science, web-page classification and spam filtering are common tasks. In biology, classification is essential with animals and plant species. Also, gene classification in bioinformatics, and image classification and disease diagnosis in medicine (Murphy, 2012).

Moreover, the wealth of classification applications in different disciplines has inspired researchers to develop various algorithms and techniques, which can apply to a variety of data types and application purposes. Real life applications have different data types. For example, text data, uncertain data (the data which contains noise or inaccurate values), time series, multimedia (which has a combination of different data types, for example, text, videos, images, and audio), and network data. Each each data type could require different classification techniques.

The most commonly used classification techniques are probabilistic methods, decision trees, rule-based methods, instance-based methods, support vector machine (SVM) methods, and neural networks (details are below). In this section, we introduce the general idea of each technique, with more details of some methods in the next sections.

Probabilistic methods predict the class label of an object by producing a probability for each possible class, then selecting the predicted class based on the highest probability. One of the simplest probabilistic techniques is the naïve Bayes classifier which uses Bayes theorem to find the posterior class probability.

Decision trees are a popular data classification method. The general idea is to partition the training data (which has known class labels) hierarchically as a tree structure using split criteria based on the data attributes. Each branch of the tree denotes an outcome (it could be probabilities or choices or any other outputs) until we reach the leaf nodes which have the decision output as a categorical or numerical class label. The constructed tree is used to predict classes for the test data (which has unknown class labels). One of the well known decision tree methods is ID3. It constructs a decision tree from a training set with a local optima solution, which leads to less classification accuracy than other methods. The improved version of ID3 is C4.5 which, for example, can deal with missing values and can also handle continuous attribute values. The CART (classification and regression tree) algorithm can be used with both discrete and continues attributes (Wu et al., 2008; Hssina et al., 2014).

Rule-based methods are similar to decision tree methods with the major difference that the induced rule sets can allow overlap, while in decision trees, overlapping is not permitted. Different methods are based on different ways of setting these rules. For example, classification based on associations (CBA) which determines rules that satisfy a specific support, and RIPPER which is an efficient way to handle large and noisy data (Aggarwal, 2015).

Instance-based methods, which predict the test instance based on some training instances, which are determined based on the distances between the test instance and training instances. Also it is called lazy learning, because it depends only on the knowledge of training data to predict the closest test instance. The simplest instance-based method is the nearest neighbour classifier, which we discuss in detail in Section 5.5.2.

Neural networks were originally conceived as an approximation of the human brain to find patterns in the input data. They are used to group the unlabelled data based on similarities in clustering, and to predict the data classes based on input data in classification. One of the most common applications of neural networks is image recognition, where the program can learn and

discover a pattern from different input images to classify new images.

A large number of machine learning methods which are used in classification are described in the literature and survey papers. For example Aggarwal (2015); Wu et al. (2008).

5.3 Classification in graphs

In graph data, classification is considered as a fundamental task in graph mining and machine learning, due to the increasing number of applications in real networks. Graph classification involves two different tasks. First, label propagation, which constructs a model from the training set (labelled nodes) to classify the test set (unlabelled nodes). Our study focuses on this approach. Second, graph classification, which deals with graphs rather than nodes, so the training set is a set of labelled graphs and the test set is a set of unlabelled graphs. We do not cover this approach in our study (Aggarwal and Wang, 2010).

Moreover, classification in graphs can be node label classification or link prediction. The latter focuses on predicting the relation between existing nodes based on node features and graph structure. We do not explicitly cover this type of classification in this study. See Aggarwal (2011) for more detail about this.

In a node classification task, we first identify the features of the labelled nodes in order to construct a classifier model. These features could represent personal labels, such as gender and age group, political affiliation, or other interests. Especially in social networks, the relations between nodes are not always random, and are based on similar features, so if classification accuracy is based on the node neighbours, then it can be much higher (Aggarwal, 2011).

Before introducing node classification methods, we define some notation. For any graph (V, E), we denote the labelled nodes as V_l and the unlabelled nodes as $V_u = V \setminus V_l$. Also, we denote the label vector by Y, and the labels of V_l nodes by Y_l.

The most common node classification method is an iterative method (considered as local classifier), which depend on the node's neighbours' features to define local classifiers. Iterative methods construct feature vectors for the nodes V in cooperation with the known class values

Y_l (training set) to learn a local classifier, such as k-nearest neighbour, Naive Bayes, or decision trees.

For example in k-nearest neighbour, the method predicts the class label for the unlabelled nodes which have labelled neighbours. However, if the neighbours of a node are all unlabelled, it will remain unlabelled in the first iteration, and it remains unlabelled until at least one neighbour is labelled. The chosen number of iterations should be large enough to make sure that all nodes get labelled, and no labels are changed after the the chosen number of iterations (result stable). However, in some cases, with some local classifiers, stability is not always guaranteed. This often occurs if an unlabelled node has two neighbours from two different classes, so the predicted class can be alternate between these different classes. Therefore, we can either choose a fixed number of iterations, or continue to iterate until the change in the labels is below a fixed threshold. There are also suggested methods to reduce the number of iterations which appear to be faster than the simple iterations, which are called second-order methods (Aggarwal, 2011).

Moreover, with this strategy, the labelling procedure (after the first iteration) depends on the labelled nodes from the training set and the predicted nodes from the previous iterations. The iterative method with k-nearest neighbour classifier is an efficient tool in social network data classification (Aggarwal, 2011).

Another common approach for node classification in weighted graphs is the random walk based method. In this technique, the labels propagate over the unlabelled nodes by considering random walks (with specific length) starting from an unlabelled node and ending at a labelled node. More precisely, every unlabelled node $v_i \in V_u$ has a probability to have a label c from label set Y, which is the total probability of all random walks that start from v_i and end at a node labelled c. The predicted class is the class which has the highest probability. The length of the random walk is considered as a smoothing parameter which controls the degree of the smoothness in this method, which means by choosing a random walk of length one, just the adjacent nodes are involved in the classification (Aggarwal, 2011).

There are different techniques in the literature which are based on the random walk. For example, label propagation uses a transition matrix to find the class probabilities. The $(i, j)^{\text{th}}$ element of the transition matrix $P^{(t)}$ is the probability of the walk leaving v_i and reaching v_j in

t steps (Aggarwal, 2011, 2015).

Most of these methods could be considered as node-centric approaches of node classification, which predict one node at a time. Part of the node centric methods are local classifiers (non relational) which rely only on the local information. Another part of the methods are relational classifiers, which depend on the graph structure with the relations between nodes and the local information.

In addition, there are studies in node classification based on methods for selecting sets of labelled nodes (rather than using the whole training set) which have special features (determined by the researcher). Guillory and Bilmes (2009) thought that this idea may help in increasing the accuracy rate in predicting the unlabelled nodes in a graph. Some of these methods use the graph structure, and some use clustering techniques, to select these sets and achieve their aim. This idea is useful in the case where extracting labels from the labelled nodes is time-consuming, especially in large graphs.

Other suggested techniques select and predict a small set of the unlabelled nodes based on the graph structure, which may achieve a higher-accuracy classification rate. Then, they use these predicted nodes with the labelled nodes (from the training set) to predict the rest of the unlabelled nodes. This idea is based on active learning, which is considered as an important topic in machine learning, and complex networks. Ping et al. (2018) use batch mode active learning for node classification, which selects more than one unlabelled node to label at each query. It also provides a selection criteria of which unlabelled nodes we have to predict before other unlabelled nodes. This model performs better than single mode active learning in experimental results and in different network types; Assortative structure (similar nodes tend to link between each other, e.g. similar degrees) and Disassortative structure (dissimilar nodes tends to link between each other, e.g. high degree nodes and low degree nodes).

Another suggested view is that in some situations, the overall rate of accuracy is bad and it is difficult to improve it. However, focusing on predicting some selected unlabelled nodes (under some conditions) may succeed in achieving high probability for the correct prediction compared with overall accuracy rate. One study which improved this idea was proposed by Chaudhari et al. (2014) with applications in social networks.

Moreover, there have been studies which combine two or more different classifiers. For example, a toolkit called NetKit, which is introduced by Macskassy and Provost (2007), compares between some relational classifiers, some local classifiers, and some collective inference methods (combined between the local classifiers and relational classifiers).

There are some other network classification methods which are widely used in image segmentation, where the goal is to predict the label of each pixel: for example, Markov random fields (MRFs), where the Markov network is a random variable with random properties presented by undirected graphs (Macskassy and Provost, 2007).

In our study, we examine different node classification methods and describe them in detail in the next sections. Before that, we talk about assessment of the classifiers' performance and results.

5.4 Evaluation of classification methods and results

As we know from the previous section, any classifier requires a training set to build the classifier model and a test set to evaluate this model. One of the simplest validation methods is hold-out evaluation, which divides the data into a training set and a test set, where the training set usually has double the size of the testing set. This method has some drawbacks; by splitting the data, there is no chance to discover all the patterns as the classifier is evaluated in a subset of the data (test set). Also, the classifier performance is affected by the choice of the training and testing sets, as in some cases, the test set has easy or difficult nodes to classify. However, there is a chance that some influential nodes do not participate in the model structure as they are in the test set rather than in the training set. Although we can repeat the method to avoid these problems, there is still a chance that there might be some essential nodes not explored in the training set.

A useful development of hold-out evaluation was proposed by Stone (1974) which is called K-fold Cross Validation (K-CV). Its idea is based on splitting the data into K approximately equal parts, where each part is considered in turn as a test set and the remaining $K - 1$ parts as a training set. This assures that all nodes are tested once in the classification task.

The crucial point is choosing an appropriate K, as it affects the trade-off between bias and variance. Choosing a small K, for example $K = 2$, means that the classification results are more biased as the training sets have different nodes, so the classifier performance could change based on the differences in the training sets. Some researches suggest appropriate values of K to be 5 or 10.

The most widely used technique is known as Leave-One-Out CV (LOOCV). With this method ($K = n$), each node is classified by using all $n - 1$ remaining nodes, so, we have n different classifiers, each trained by $n - 1$ data and tested by one node. The result is the average of the n tested nodes which is considered to be unbiased, as all nodes are used for testing. We use this technique to evaluate our classifiers in the simulated graphs and real networks. Other evaluation methods are mentioned in the literature, but most of them have a high computational cost. More details can found in Aggarwal (2015).

After choosing a validation technique, and applying a classification method, we have to evaluate the quality of the results to check the classification performance. This is usually done by comparing the predicted classes to the real classes of the test set. The most common approach is to use a confusion matrix M which is a square matrix with a dimension equal to the number of classes. Each entry M_{ij} corresponds to the number of nodes whose real class is i and predicted class is j. The classification performance is assessed by the classification accuracy rate, which is measured by the trace of M divided by n (Aggarwal, 2015).

$$\text{Accuracy rate} = \frac{Tr(M)}{n}.$$

5.5 Node classification approaches

In this section, we focus on the node classification methods that we use in our study in more detail.

5.5.1 Naive classifier strategy

This classifies a test observation v_u to the largest frequency class of labelled nodes in graph G. Let $V = \{v_1, ..., v_n\}$ be the nodes in graph G and $c_i \in \{1, ..., J\}, i = \{1, \ldots, n\}$ is the nodes' classes in the graph, so, the naive classifier for each unlabelled node $v_u \in V$ is

$$\hat{c}_u = \text{Predicted class } (v_u) = \arg \max_j |A_j|, \tag{5.1}$$

where $A_j = \{i : c_i = j\}$, so $|A_j|$ is the number of nodes which have class j. In a naive classifier, all unlabelled graph nodes have the same predicted class, which is the majority class over all the labelled nodes.

5.5.2 The K-Nearest Neighbour classification method (KNN)

Let v_u and c_i be as in the previous section, and K be a positive integer. The KNN classifier identifies the K nodes in the graph that are closest (with respect to distance) to v_u which are represented by $N_u(k) \subset V \setminus v_u$. It then estimates the conditional probability for class j as the fraction of nodes in $N_u(k)$ whose response value (their class) is equal to j:

$$\hat{P}(c_u = j|v_u) = \frac{1}{K} \sum_{i \in N_u(k)} I(c_i = j), \qquad j \in \{1, ..., J\}. \tag{5.2}$$

Finally, KNN classifies the test node v_u to the class with the largest probability. This can be done for all graph nodes. The KNN classifier is therefore a vote amongst K-nearest neighbours:

$$\text{Predicted class } (v_u) = \hat{c}_u = \arg \max_j \hat{P}(c_u = j|v_u) \tag{5.3}$$

The accuracy rate of KNN classifier for all graph nodes V is calculated by the confusion matrix.

It is clear from the KNN concept that two factors are involved in the results. First, the distance measure between nodes, which is BTD and SP in our study. Second, the choice of K, which plays an important role in the classification result. By choosing K too small e.g. ($K = 1$), the classification may be affected by some outliers (in Euclidean space) which could have unusual location and may have different behaviour, so the prediction maybe unrealistic. However, choosing a large K may lead to over-smoothing in the classification, and the local structure may be lost. Therefore, we can either choose K as a subjective parameter which depends on the application, or find the optimal K by trying different values and choosing the

one which maximizes the accuracy rate. In the case of $K > 1$, it is possible to weight the K nodes differently based on the distance between the test node and each neighbour, and this could be based on the subjective opinion of the user or any other factor related to the application. We could suggest that each node in a graph acts differently than other nodes, so we may weight them based on different standards. For example, super nodes have higher weights than most remote nodes due to their centrality.

Various extensions of KNN are introduced in the literature. For example, weighted vote relational neighbour classifier (wvRN) which considers a weight for each neighbour that participates in the procedure. This weight could depend on different features, for example; the distance between the unlabelled node and its neighbours, or the neighbour's node degree. Or it could depend on the distribution of the classes around the unlabelled node as in class distribution relational neighbour classifier (cdRN). More details about KNN can be found in Larose and Larose (2014); Aggarwal (2015).

KNN is one of the most popular methods in nonparametric classification, because it is easy to understand and applicable for different applications. Also, it usually achieves high accuracy rates in several studies in the literature of node classification. For example, Li et al. (2012) compare the accuracy of KNN compared with traditional diagnostic methods in an experiment distinguishing lymph node metastasis from node metastasis in the gemstone spectral imaging (GSI) in gastric cancer. The study includes 38 lymph node samples in gastric cancer and the results show an overall accuracy of 96.33% with KNN compared with traditional diagnostic methods which get an accuracy of 75.2%.

However choosing $k = 1$ may be affected by outliers, this issue does not exist in connected components. Therefore, we apply the simplest version of this method with $k = 1$, so the prediction strategy depends on the class of the nearest (closest) node. This is easier in computations, and also will avoid the problem of getting equal frequently classes among the nearest neighbours in case of $k > 1$.

As BTD breaks the ties in graph distance we expect to have one nearest neighbour for each node (in maximum cases two nearest neighbours for some nodes), while in SP there are multiple nearest neighbours for each node, which makes the random choice between them variable

comparison. To address this issue, we decide to repeat the random selection from each node's neighbours multiple time to end with the average of all trials. Simulation are described in Table 5.1 in Section 5.8.1.

We also thought about different methods to predict the classes based on nearest neighbour in the case of SP. According to our simulation in WS graph model, each node in SP has, on average, approximately four neighbours. The idea is to predict the node class based on the most frequent class among the node's neighbours. However, in the case of equal frequency between classes, the predicted class takes half the weight of the correct rate. This idea makes inequitable comparison between BTD and SP due to unequal number of neighbours in each distance measure. In BTD, the average number of nearest neighbours is approximately 1 for all nodes, while in SP the average number of the nearest neighbours is at least four. Therefore, comparing $K = 1$ in BTD and $K \geqslant 4$ in SP is inappropriate. Therefore, we consider the idea of repeating the random selection from the neighbours to end with the average of all trials.

5.5.3 Naive Bayes classification by kernel function

KNN classification is based on the classes of the K neighbours of a node, and the Kernel classification rule is similar to KNN classification, as the predicted class of a node is chosen based on the majority of classes in the vicinity (kernel) of this node.

In the general case, the method is applied in *Euclidean space*. Noting expression 5.2, we can use Bayes' theorem to calculate the conditional probability of class j given the data x in *Euclidean space*, $p(j|x)$, which is called the posterior probability of the class. These posterior probabilities are calculated from both the prior π_j and the conditional probabilities $p(x|j)$ for each class j, where $j \in \{1, \ldots, J\}$ and $x \in \{x_1, \ldots, x_n\}$. The formula is written as

$$\begin{aligned} p(j|x) =& \frac{p(j)p(x|j)}{p(x)} \\ =& \frac{\pi_j p(x|j)}{\sum_{k=1}^{J} \pi_k p(x|k)} \\ =& \frac{\pi_j f_j(x)}{\sum_{k=1}^{J} \pi_k f_k(x)}, \end{aligned} \tag{5.4}$$

which can be estimated using a kernel density estimates:

$$\frac{\pi_j \hat{f}_j(x)}{\sum_{k=1}^{J} \pi_k \hat{f}_k(x)},$$

where

$$\hat{f}_j(x) = \frac{1}{|C_j|h} \sum_{x_i \in A_j \setminus x} K\Big(\frac{x - x_i}{h}\Big), \tag{5.5}$$

and h is a smoothing parameter which we assume to be equal for all classes.

The classification rule is equivalent to allocating x to the class j which has the maximum posterior probability

$$\begin{aligned} arg\max_j p(j|x) =& arg\max_j \pi_j \frac{1}{|C_j|h} \sum_{x_i \in A_j \setminus x} K\Big(\frac{x - x_i}{h}\Big) \\ =& arg\max_j \frac{|C_j|}{n} \frac{1}{|C_j|h} \sum_{x_i \in A_j \setminus x} K\Big(\frac{x - x_i}{h}\Big) \\ =& arg\max_j \frac{1}{nh} \sum_{x_i \in A_j \setminus x} K\Big(\frac{x - x_i}{h}\Big), \end{aligned} \tag{5.6}$$

where π_j is estimated by $\frac{|C_j|}{n}$, and we can ignore the denominator in Eq. (5.4) as it does not depend on j.

This calculates the probabilities of classifying data point x_u to class j. We use the LOOCV method to predict the class of x_u based on other data classes:

$$\hat{c}_u = \arg\max_j \hat{P}_h(c_u = j|x_u) = \arg\max_j \sum_{x_i \in A_j \setminus x_u} K\left(\frac{x_u - x_i}{h}\right), \tag{5.7}$$

which is applied to all data points x_i.

Different types of kernels can be used in kernel density estimation, the most common kernel are (normalization constants which do not affect the result are omitted)

- Gaussian kernel $K(x_i, x_j) = e^{-(d)^2/2\sigma^2}$, where $d = |x_i - x_j|$.
- Epanechnikov kernel $K(x_i, x_j) = (1 - d^2)$, , where $I_{[d \leqslant 1]}(1 - d^2)$.
- Cosine kernel $K(x_i, x_j) = cos\big(\frac{\pi}{2}(d)\big)$, $I_{[d \leqslant 1]} cos(\frac{\pi}{2} d)$.

The probability that x_i has predicted class $\hat{c}_i$ equal to the original class c_i satisfies

$$P_h(\hat{c}_i = c_i|x_i) = \sum_{j=0}^{J} P_h(\hat{c}_i = j)I[c_i = j], \tag{5.8}$$

and the prediction accuracy rate is calculated by

$$\text{Accuracy rate} = \frac{1}{n}\sum_{i=1}^{n} I[P_h(\hat{c}_i = c_i|x_i) > P_h(\hat{c}_i \neq c_i|x_i)], \tag{5.9}$$

where the chosen h which maximizes the accuracy rate (Michie et al., 1994; Hand, 1982; Kulkarni et al., 1998).

In graphs, the data $\{v\}$ are nodes (finite countable set), and do not belong to Euclidean space. The kernel classifier for a node is a weighted vote amongst other nodes in the graph. In this method, choosing a kernel function is the first step in the method, then choosing an appropriate smoothing parameter h is highly important to minimize the classification error rate. In this study, we choose the Gaussian kernel function as the most common and preferable kernel among researchers.

Consider an undirected and unweighted graph $G(V, E)$, $|V| = n$, where $d(v_u, v_i)$ is the distance between v_u and v_i. The kernel function is

$$K_h(v_u, v_i) \propto \phi_h(v_u, v_i) = \begin{cases} exp\left(\frac{-d^2(v_u,v_i)}{2h^2}\right) & v_u, v_i \in \{v_1, ...v_n\} \\ 0 & v_u = v_i, \end{cases} \tag{5.10}$$

for a given value of smoothing parameter h.

The classification rule in the case of two classes (i.e. $j \in \{0, 1\}$) is

$$\hat{c}_u = \begin{cases} 0 & \text{if } \sum\limits_{v_i \in A_0 \setminus v_u} K\left(\frac{d(v_u, v_i)}{h}\right) > \sum\limits_{v_i \in A_1 \setminus v_u} K\left(\frac{d(v_u, v_i)}{h}\right) \\ 1 & \text{otherwise.} \end{cases} \tag{5.11}$$

Note that the equality case is rare (in practice) in kernel classifier. The probability that x_i has predicted class $\hat{c}_i$ equal to the original class c_i satisfies

$$\begin{aligned} P_h(\hat{c}_i = c_i|v_i) =& \sum_{j=0}^{1} P_h(\hat{c}_i = j))I[c_i = j] \\ =& P_h(\hat{c}_i = 1)I[c_i = 1] + P_h(\hat{c}_i = 0)I[c_i = 0]. \end{aligned} \tag{5.12}$$

and the prediction accuracy rate is calculated by

$$\begin{aligned}\text{Accuracy rate} =& \frac{1}{n}\sum_{i=1}^{n} I[P_h(\hat{c}_i = c_i|v_i) > P(\hat{c}_i \neq c_i|v_i)] \\ =& \frac{1}{n}\sum_{i=1}^{n} I[P_h(\hat{c}_i = c_i|v_i) > 0.5]. \end{aligned} \tag{5.13}$$

However, the accuracy rate is a piecewise constant function in h, so it is non-differentiable and we can not optimize the best value of h by a gradient method. So, instead we have to apply a grid search. Which is also hampered by local minima, as shown for example in Figure 5.2.

Therefore, we initially consider a continuous function (which respects the probability of true prediction of the classes) to choose the optimal h by LOOCV method (Figure 5.1):

$$\begin{aligned} h_{optim} =& \arg\ \max_h \prod_i (P_h(\hat{c}_i = c_i|v_i)) \\ =& \arg\ \max_h \log \prod_i (P_h(\hat{c}_i = c_i|v_i)) \\ =& \arg\ \max_h \sum_i \log(P_h(\hat{c}_i = c_i|v_i)). \end{aligned} \tag{5.14}$$

After finding the optimal smoothing parameter h from Eq. (5.14), we could use it to calculate the accuracy rate with the same kernel by using Eq. (5.13).

However, we have an issue in that the h value which maximizes Eq. (5.14) is not equal to the h which maximizes Eq. (5.13). So, we introduce a transformation of Eq. (5.14) which can preserve smoothness and approximates Eq. (5.13), and yields the same accuracy as given by optimization of Eq. (5.13). The transformation is as follows:

$$\sum_i \log(P_h(\hat{c}_i = c_i|v_i)) \rightarrow \sum_i F(P_h(c_i = \hat{c}_i|v_i)), \tag{5.15}$$

where

$$F(p) = \begin{cases} 1 - \frac{1}{2}[2(1-p)]^{\alpha} & 0.5 \leqslant p \leqslant 1 \\ \frac{1}{2}(2p)^{\alpha} & 0 \leqslant p < 0.5, \end{cases} \tag{5.16}$$

and α controls the degree of the transformation as shown in Figure 5.3. Note that $\alpha = 1$ gives $F(p) = p$. There are different transformation formulas can solve the problem. However, this formula has some desirable properties with the parameter α which controls the smoothness.

Firstly, it makes $p = 0.5$ a fixed point and allows the probabilities to change around it in two different directions (increasing the probabilities above 0.5, and decreasing the probabilities below 0.5). Secondly, it also makes $p = 0$ and $p = 1$ as fixed points. For any value of α, $F(0) = 0$, $F(0.5) = 0.5$, and $F(1) = 1$. In our simulation study, we choose $\alpha = 10^6$.

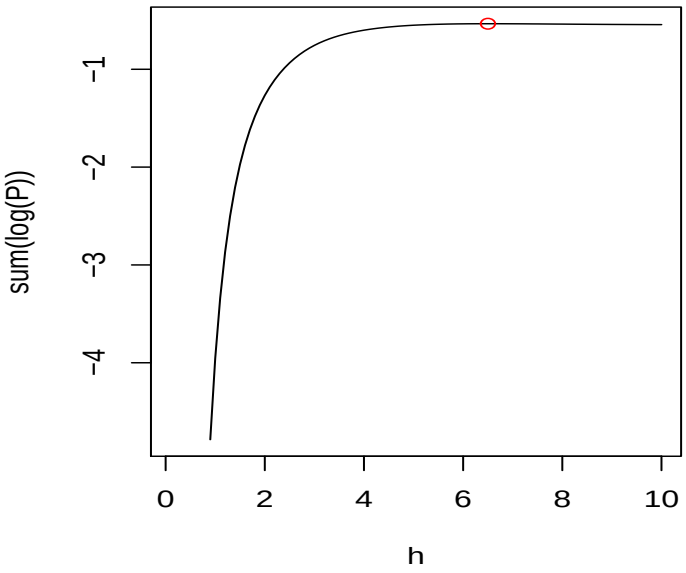


Figure 5.1: Plot of the relation between values of bandwidth h and probabilities in Eq. (5.14) for a graph example from FF model. The maximum of the is at curve at $h = 6.5$.

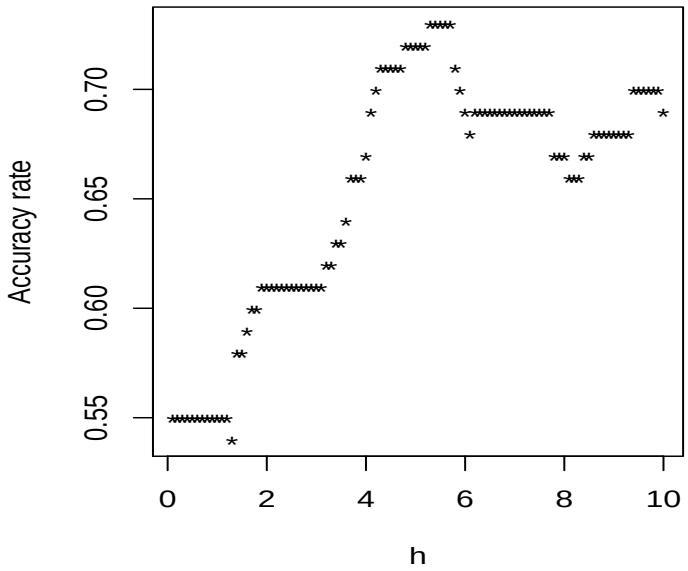


Figure 5.2: Plot of the accuracy rates corresponding to values of the bandwidth h for an example graph. The maximum of the curve is at $h = 5.3$ and the corresponding accuracy rate is 0.73, while the accuracy corresponding to $h = 6.5$ is 0.68.

In Figure 5.4, we show the correct classification probabilities from E.q. (5.15) for different values of parameter α, compared with the accuracy rate function E.q. (5.13), corresponding to a range of h values. The figure shows that for a large enough value of α, the maximum point of the curve is identical to the maximum point of the accuracy rate, and corresponds to the same optimal h. Note that, as α increases, the function F in Eq. (5.15) (black curve) converges to the accuracy rates (expression (5.13), blue curve). Although the black curve is differentiable, it still suffers from local maxima.

5.5.4 Adaptive kernel classifier

We know from the fixed kernel method in Section 5.5.3 that the smoothing parameter h controls the degree of smoothness. Choosing small h allows for small details to appear in the estimation and (in the *Euclidean case*) is approximately equivalent to 1NN classification, while choosing large h leads to disappearance of important details in the structure. This means that the kernel function with the chosen h is used to estimate the density at each data point equally. However,

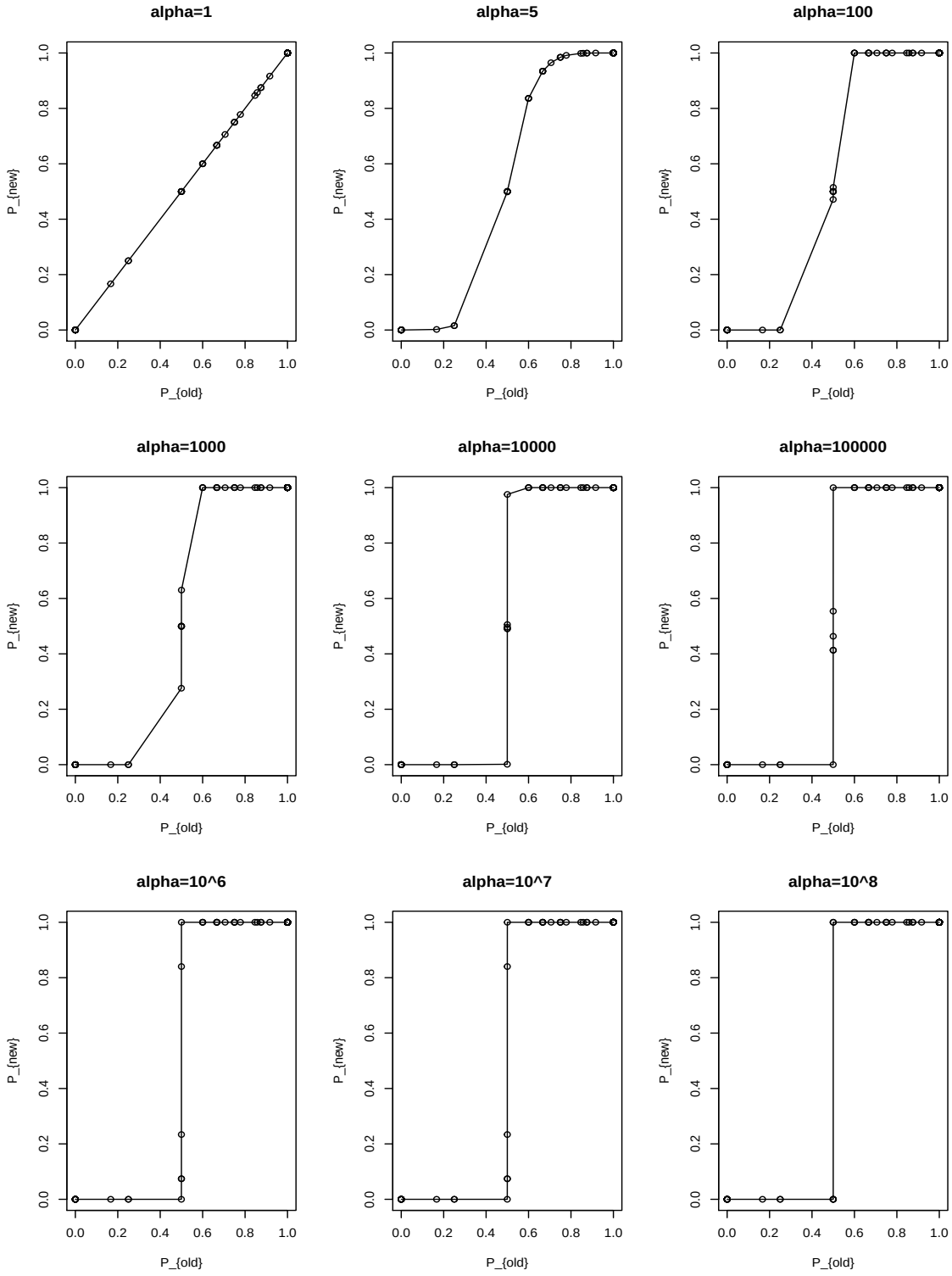


Figure 5.3: Nine plots showing the transformation of the probabilities with different parameter α.

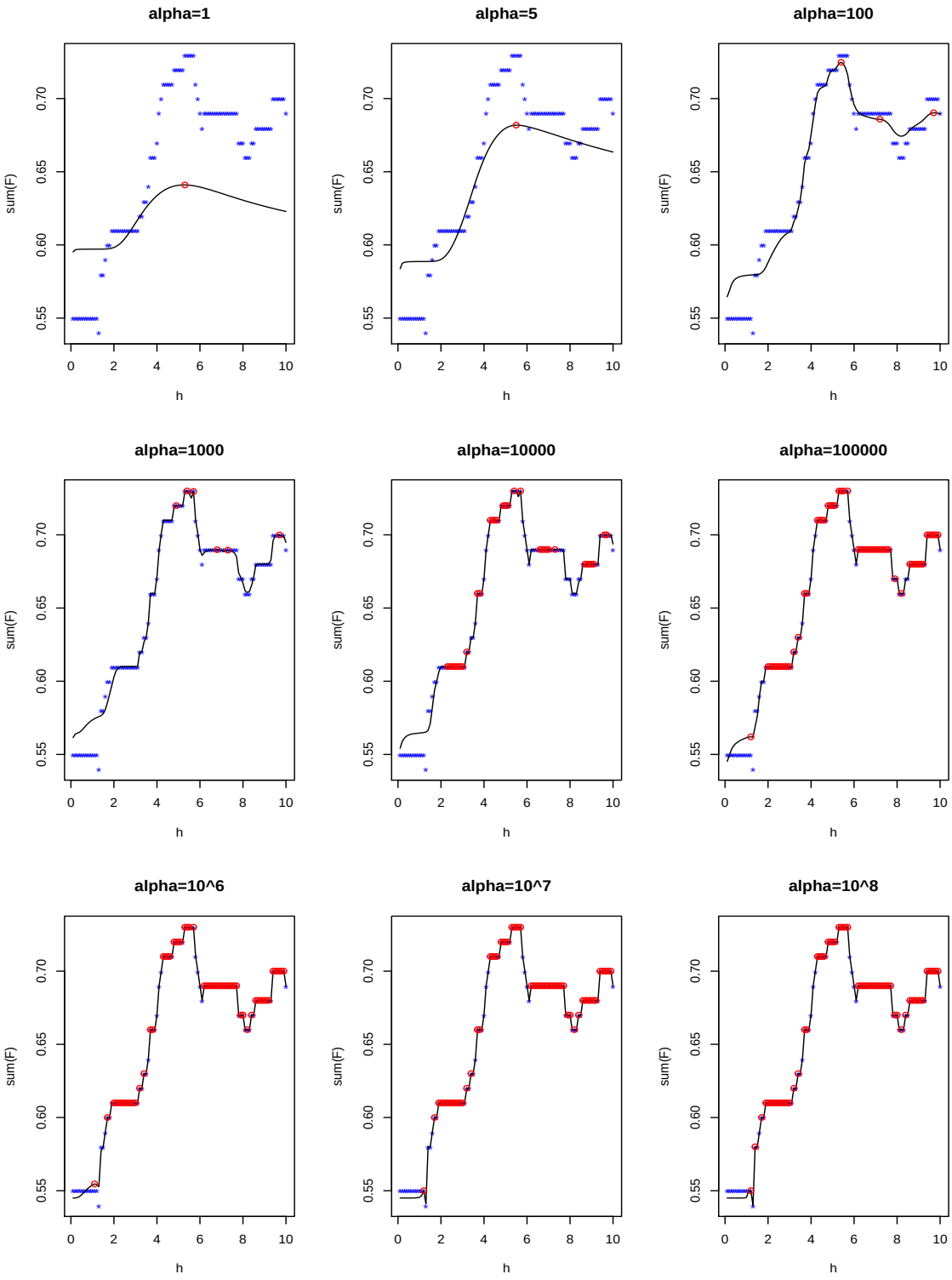


Figure 5.4: Plots showing the effect of the probability transformation on choosing the optimal h based on expression (5.15) (black curve). Accuracy rate is the blue curve, and the global and local maximum points of the black curve are marked as red.

choosing optimal h is important as it could produce high quality estimation or classification in various applications. In some cases, the fixed bandwidth kernel may produce poor estimation due to the nature of the data, for example, data with local noise, heavy tails, or multi-modality, which makes it important to find a proper h value that is suitable for all data points. Silverman (1986) examined two different sets of real data: suicide data, which has a long right tail, and the eruption lengths (in minutes) of the Old Faithful geyser, which has a bimodal structure. In both examples, a fixed kernel loses its efficiency in estimating the data due to the data structure. Hence, it requires an adapted smoothing parameter to match the data density structure. This means that any point in a dense area needs smaller h than a point in a sparse area, which is the aim of the adaptive kernel method.

Two different categories of methods are based on the concept of the adaptive kernel: balloon estimators and sample point estimators. In a balloon estimator, each estimated point x_j, $j = 1, \ldots, n$, has its own smoothing parameter, so each estimator $\hat{f}(x_j)$ is an average over fixed scaled kernels with fixed h_j, and the estimator takes the form

$$\hat{f}(x_j) = \frac{1}{nh_j} \sum_{i=1}^{n} K\Big(\frac{x_i - x_j}{h_j}\Big). \tag{5.17}$$

In a sample point estimator, the smoothing parameter depends on the sample points x_i, so each estimator $\hat{f}(x_j)$ is an average over different scaled kernels with different h_i. This estimator is written as

$$\hat{f}(x_j) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h_i} K\Big(\frac{x_i - x_j}{h_i}\Big). \tag{5.18}$$

Many studies in the literature compare these two estimators by analysing the advantages and disadvantages. In our study, as our data points are the graph nodes, we choose the balloon estimator to make sure that each node has its own h in the estimation, and this h depends on the node itself without being affected by other nodes. This is unlike the sample point estimator, where the estimation at any node could be affected by other nodes' features even if they are far from the point under classification. This idea is influenced by the idea of Terrell and Scott (1992) that the sample point estimator may suffer from the "non-locality" phenomenon as the estimated point could be affected not only by the neighbours data points but also by the far data points.

In our study, as we have applications in some social networks which have relationships

between the nodes and their neighbours, we suggest that the smoothing parameter can be a function of node degree. This means that the degree of each node controls the smoothness of its estimator, so by decreasing the smoothness on the high-degree nodes, the nodes' kernels will cover (approximately) the nearest neighbour nodes, while by increasing the smoothness on the small-degree nodes, the nodes' kernel will cover beyond the nearest neighbour nodes. This takes the form

$$\hat{f}(v_j) = \frac{1}{n}\frac{1}{h_j}\sum_{i\neq j}^{n} K\left(\frac{d(v_j, v_i)}{h_j}\right), \tag{5.19}$$

where we choose the parametrization $h_j = a \times (d_j)^b, a > 0, b \leqslant 0$, and $d_j = \text{degree}\ (v_j)$.

The expression of h_j is constructed to have a realistic dependency on node degrees. The parameter a allows h to increase as it has positive range of values, while parameter b allows h to decrease based on the node degrees as it has negative range of values. We start searching with initial value $a = 1$ which starts searching with small values of h without any dependency on the node degree with $b = 0$. Then, decreasing b to make a balance between different node degrees, so the higher degree nodes will have smaller h than lower degree nodes. However, the optimized values of a and b are chosen by the gradient method by using `optim` function in `R` to assure that we find the optimal pair of a and b for each node in the graph depending on the $n - 1$ node (by LOOCV). We also use the transformation in the Expression 5.16 with $\alpha = 10^6$.

5.5.5 Support vector machines (SVMs)

Support vector machines (SVMs) are a common and well known machine learning method for different learning tasks, for example, classification, regression, and object recognition. The goal of SVM is to learn good classification of the data in high dimensional feature space from a training data set which leads to correct predictions on the test (or new) data. In this section, we consider binary classification as a simple example of the classification task, which appears in several real world problems.

Suppose initially, we have linearly separable training data $T = \{(x_i, y_i);\ i = 1, \ldots, n;\ x_i \in \mathbb{R}^n\}$ where y_i denotes the labels $\{-1, 1\}$. SVM provides a hyperplane which could have a max-

imum total separating distance from points x_i, and has the equation

$$w \cdot x + b = 0, \tag{5.20}$$

where $\cdot$ is the scalar or inner product, so $w \cdot x \equiv w^T x$, b is the perpendicular distance from the hyperplane to the origin, and the weights w determine its orientation. This hyperplane separates the data based on the two classes. Let $w \cdot x + b = 1$ for the nearest point on the upper side to the hyperplane (in Figure 5.5 as x_1), and $w \cdot x + b = -1$ for the nearest point on the lower side to the hyperplane (in Figure 5.5 as x_2), which are called the canonical hyperplanes, and the distance between the hyperplane and the canonical hyperplane is called the margin band. Therefore, the points with positive label ($y = 1$) on one side ($w \cdot x + b \geq 1$), and the points with negative label ($y = -1$) on the other side ($w \cdot x + b < -1$).

We let $f(x)$ be the decision function

$$f(x) = \text{sign}(w \cdot x + b), \tag{5.21}$$

and classify the data points as negative if $f(x) = -1$ and lies below the decision surface, and classify the data points as positive if $f(x) = 1$ and lies on or above the surface. Both of these can be combined as one inequality,

$$y_i(w \cdot x_i + b) \geq 1 \quad \forall i. \tag{5.22}$$

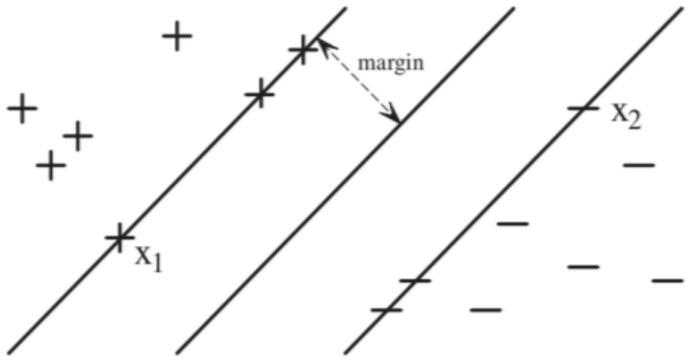


Figure 5.5: Plot showing the optimal hyperplane with maximum margins.

To find the best hyperplane which has the maximum margin band. We have $w \cdot x_1 + b = 1$ and $w \cdot x_2 + b = -1$ which can be combined to $w \cdot (x_1 - x_2) = 2$, and the normal vector for the separating hyperplane $w \cdot x + b = 0$ is $\frac{w}{||w||_2}$, where $||w||_2$ is the square root of $w^T w$.

Therefore, the distance between the two canonical hyperplanes can be interpreted as the projection of $x_1 - x_2$ onto $\frac{w}{||w||_2}$, which leads to $(x_1 - x_2).\frac{w}{||w||_2} = \frac{2}{||w||_2}$. As our aim is to

maximize this margin with respect to Eq. (5.22), we convert it into minimization problem and minimize an object function (cost function)

$$\frac{1}{2}w.w = \frac{1}{2}||w||_2^2, \tag{5.23}$$

and because this object function is quadratic and convex and the constraints Eq. (5.22) are linear, we introduce Lagrange multipliers $\alpha_i \geq 0;\ i = 1, \ldots, n$ in a function consisting of the sum of the constraints over n multiplied by corresponding α_i and subtract it from the object function. The Lagrangian formula is

$$L(w, b) = \frac{1}{2}w \cdot w - \sum_{i=1}^{n} \alpha_i [y_i(w \cdot x_i + b) - 1]. \tag{5.24}$$

The optimal solution can be found by minimizing Eq. (5.24) with respect to either minimizing the primal variables w and b, or maximizing the dual variable α, and both lead to equal optimal values by the Duality Theorem (Haykin, 1999). For the former, we can find the derivatives of Eq. (5.24) with respect to b and w and set them to zero

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \Rightarrow \sum_{i=1}^{n} \alpha_i y_i = 0, \tag{5.25}$$

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \Rightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i. \tag{5.26}$$

By expanding Eq. (5.24), we get

$$L(w, b) = \frac{1}{2}w.w - \sum_{i=1}^{n} \alpha_i y_i w.x_i - b\sum_{i=1}^{n} \alpha_i y_i + \sum_{i=1}^{n} \alpha_i, \tag{5.27}$$

and as the third term on the right side is equal to zero from Eq. (5.25), we get

$$L(w, b) = \frac{1}{2}w \cdot w - \sum_{i=1}^{n} \alpha_i y_i w \cdot x_i + \sum_{i=1}^{n} \alpha_i. \tag{5.28}$$

The first term on the right hand side can be written as

$$\frac{1}{2}w.w = \frac{1}{2}w\sum_{i=1}^{n} \alpha_i y_i x_i = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j), \tag{5.29}$$

by substituting Eq. (5.29) into Eq. (5.28), we get the dual formulation, which is known as the Wolfe dual

$$W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j). \tag{5.30}$$

Now, we have to find the optimal Lagrange multipliers $\alpha_{i,0}$ which maximize Eq. (5.30) with respect to the constraints $\sum_{i=1}^{n} \alpha_i y_i = 0$ and $\alpha_i \geq 0,\ \forall i$, using the training data of size n, so we have $\alpha_{i,0}$ for each training point x_i. The training points which have $\alpha_{i,0} > 0$ lie on the

hyperplane H_1 or H_2, and are called support vectors. The points which have $\alpha_{i,0} = 0$ lie on the side of H_1 or H_2. From these optimal $\alpha_{i,0}$, we can write the optimal weight w_0 from Eq. (5.26) as

$$w_0 = \sum_{i=1}^{n} \alpha_{i,0} y_i x_i, \tag{5.31}$$

and the decision function Eq. (5.21) can take the form

$$f(x_j) = \text{sign}(\sum_{i=1}^{n} \alpha_{i,0} y_i (x_i . x_j) + b_0). \tag{5.32}$$

To find the optimal value of b, we pick a support vector point x_s which has a positive label and substitute it in Eq. (5.22):

$$w_0 \cdot x_s + b_0 = 1 \Rightarrow b_0 = 1 - w_0 \cdot x_s, \quad \text{for } \; y_s = 1. \tag{5.33}$$

This method is useful in the case of linearly separable data in an input space. However, in most cases, the data are not linearly separated in the input space (see Figure 5.6 (left)), but can be separated in higher dimensional space by mapping the data points into a different dimension space (see Figure 5.6 (right)). This is called the feature space, and the mapping function is $\Phi(x_i)$. The dot product in Eq. (5.30) is replaced by

$$W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (\Phi(x_i) \cdot \Phi(x_j)). \tag{5.34}$$

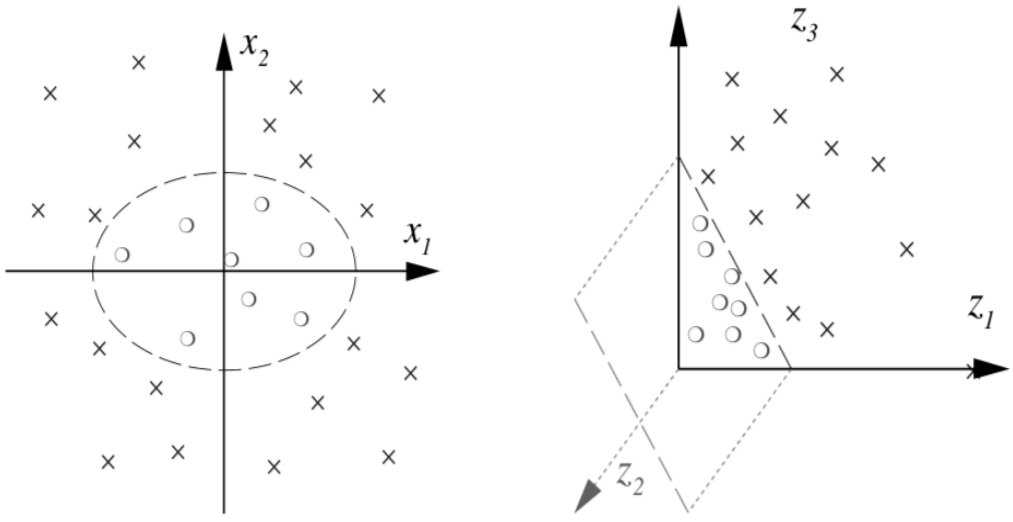


Figure 5.6: Non linearly separable data (left), and after mapping from R^2 to R^3 (right) (Maimon and Rokach, 2010).

Fortunately, we do not need to calculate the value of $\Phi(x_i)$ as it is implicitly defined by a

kernel function $K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$. Different kernel functions can be used, and the most common kernel is Gaussian kernel:

$$K(x_i, x_j) = e^{-(x_i - x_j)^2/2\sigma^2},$$

where the parameter σ is specified by the user. This strategy of the non-linear classifier is called the kernel trick, and was developed by Boser et al. (1992). We can maximize the dual formulation by using the kernel trick:

$$W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j). \tag{5.35}$$

Finally, we can update the optimal decision function as

$$f(x) = sign(\sum_{i=1}^{n} \alpha_{i,0} y_i K(x_i, x_j) + b_0), \tag{5.36}$$

where b_0 is calculated from Eq. (5.33) in the feature space.

For more details, see Campbell and Ying (2011); Deng et al. (2013); Maimon and Rokach (2010).

In graphs, we have the nodes $\{v\}$ as a finite countable set, that do not belong to Euclidean space. The SVM classifier is designed for the data in space, while we just have distances between nodes. It is a challenge to transform the nodes in distance space to points in data space with high dimension distances. So, we approximate the distances to represent the nodes positions in a low-dimensional coordinate system by applying multidimensional scaling (MDS) which allows us to apply the SVM classifier.

Multidimensional scaling (MDS)

Multidimensional scaling is a non-linear dimension reduction technique used to find a low-dimensional representation of the data, which tries to preserve internal distances.

This procedure is effective when the new distances between objects in space δ_{ij} are close to the actual distances d_{ij}

$$\delta_{ij} \cong d_{ij}, \quad \forall i, j \in \{1, \ldots, n\}.$$

There are two different methods of MDS: the first is called metric or classical multidimensional scaling (CMDS) which is also known as principal coordinate analysis. It is used in the case where we have the actual distances between objects, as in our study, where the actual distance δ_{ij} is the single dissimilarity between nodes, using either BTD or SP. The second is called non-metric multidimensional scaling (NMMDS), which is used in the case where just the ranks of the distances between objects are known.

In the CMDS method, we start with the original distance matrix $D_{n\times n} = (d_{ij})$, and we look for new distances δ_{ij} which represent the n points in k dimensions. Commonly, $k = 2$ is preferred for visualization purposes, $k = 3$ is efficient, but higher dimensions make the data more complex. The coordinates are constructed by these steps:

- Calculate matrix A as $A_{n\times n} = (a_{ij}) = (-\frac{1}{2}d_{ij}^2)$.
- Calculate matrix $B = (b_{ij})$ from matrix A, where $b_{ij} = a_{ij} - \bar{a}_{i.} - \bar{a}_{.j} + \bar{a}_{..}$, where $\bar{a}_{i.} = \sum_{j=1}^{n} a_{ij}/n$, $\bar{a}_{.j} = \sum_{i=1}^{n} a_{ij}/n$, and $\bar{a}_{..} = \sum_{ij} a_{ij}/n^2$.
- Determine the k largest eigenvalues $\lambda_1 > \lambda_2 > \cdots > \lambda_k$ of B with corresponding eigenvectors $Y_k = (y^{(1)}, y^{(2)}, \ldots, y^{(k)})$, normalized by $||y^{(j)}||^2 = \lambda_j$ where $j \in \{1, 2, \ldots, k\}$.
- The new coordinates of n objects are the rows of Y_k, which are called the principal coordinates in k dimensions.

In our study, we need to apply MDS to our distance matrix to represent the nodes in a low-dimensional coordinate system in order to use it in SVM classifier with the kernel trick. MDS can be applied in `R` using a function called `cmdscale` for CMDS, and `isoMDS` function for NMMDS (Team, 2017).

We can check the efficiency of the MDS by calculating the stress factor between the actual distance and the scaled distance by the formula

$$\text{stress} = \sqrt{\frac{(d_{ij} - \delta_{ij})^2}{\sum \delta_{ij}^2}},$$

where $\delta_{ij} = \sqrt{\sum_{p=1}^{k}(x_{ip} - x_{jp})^2}$ are the scaled distances, k is the number of dimensions of the

reduced space, d_{ij} is the actual distance (in our study it can be BTD or SP). The stress factor values fall between 0 and 1, and a good stress factor value is close to 0. They should have an inverse relationship between the number of dimensions and stress values, so by increasing the number of dimensions, the stress values should decrease.

Borg et al. (2013) mention some features of MDS, and we notice some of them appear in the case of graphs. The first feature is the relationship between the graph size and the stress factor value, large n leads to a high stress value and vice versa. More details about MDS and the concept are in Rencher and Christensen (2012); Seber (1984).

In our study, after several experiments, we found that `isoMDS` function (NMMDS method) in `R` is the best to use for our purpose. It produces the nodes' coordinates in space with a selected number of dimensions k. Its behaviour follows the expected relationship between the number of dimensions and the stress values (negative correlation), and also the expected relationship between the graph size and stress values (positive correlation) as is clear from Figure 5.7.

After that, we use `ksvm` function from package `kernlab` in `R` with Gaussian kernel over $(n-1)$ training set as we use LOOCV method. The σ in the Gaussian kernel is automatically calculated by the `ksvm` function. We also predict the test node (the i^{th} removed node) by using the `predict` function. The classification accuracy rate is calculated by using the confusion matrix.

5.6 Graphs simulation with distribution of class labels

In our simulation study, we focus on the same graph models: the FF model, the WS model, the ER model, and the BA model.

For each simulated graph, we distribute binary classes according to a weight parameter w, which controls the spatial distribution. The distribution process starts by selecting two initial nodes v_1 and v_2, for example, two super nodes or two well separated nodes, and assigning two different classes to these nodes. In this study, we always select two well separated nodes. Then, we distribute binary classes $\{0, 1\}$ equally over the graph nodes. This allocation is a function of

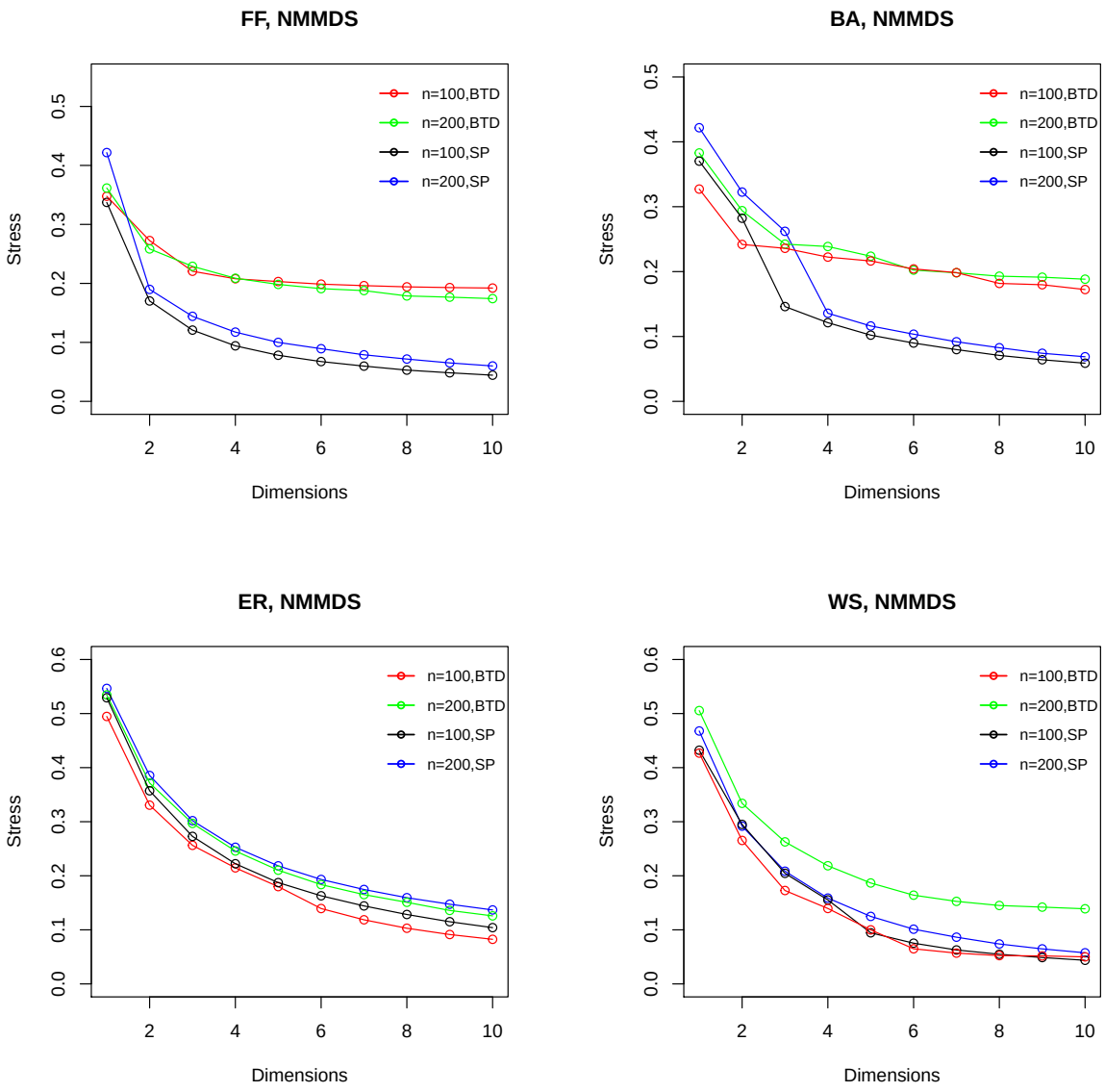


Figure 5.7: Four plots from different graph models, FF, BA, ER, and WS showing the relationship between stress factor values and dimensions. Each plot shows a behaviour comparison between BTD and SP in two different graph sizes $n = 100, 200$.

parameter w, which controls the probability p which also depends on how close the nodes are to the initially chosen nodes in each graph.

The probability p is proportional to the ratio between the distances to the initial nodes,

$$p(c_i = 0) \propto e^{-w(d_{i1}-d_{i2})/s},$$

where w is a weight parameter which controls the degree of clustering in the graph classes, d_{i1} is the distance between v_1 and node i, d_{i2} is the distance between v_2 and node j, and $s = \max|d_{i1} - d_{i2}|$.

This differences $(d_{i1} - d_{i2})$ are positive for nodes which are closer to v_2, and negative for nodes which are close to v_1, so for these nodes who are close to v_1, the power of the exponential is positive which leads to high probability p to have a class similar to v_1 class. But, for these nodes who are close to v_2, the power of the exponential is negative which leads to low probability p to have a class similar to v_1 class. The parameter w controls the clusters in the classes distribution, so high w leads to a clear clusters in the graph, while low w leads to an overlap between classes.

In all simulations, we use four values of the weight parameter, $w \in \{1, 5, 10, 20\}$. We choose these values based on our experiments of a range of w values and the structure of the classes distribution over the simulated graphs. We also check the behaviour of the range of w values with Moran's *I* coefficient which we describe in the following section.

5.6.1 Moran *I* statistic

Moran *I* is a spatial autocorrelation coefficient, which measures the degree of correlation between adjacent observations from different classes. It was proposed by Moran (1950) and has the form

$$I = \left(\frac{n}{A}\right)\frac{\sum_{i,j} a_{ij} z_i z_j}{\sum_i z_i^2},$$

where x_i is the node class $\{0, 1\}$, $z_i = x_i - \bar{x}$, and $\bar{x}$ is the average class which is here always $\frac{1}{2}$ as we distribute the classes equally over the nodes. a_{ij} are the adjacency matrix values between nodes i and j (if v_i and v_j are adjacent, $a_{ij} = 1$, and $a_{ij} = 0$ otherwise), A is the sum of the adjacency matrix a_{ij} and $\sum_{i,j}$ is the double summation over i and j. Moran *I* is a global

measure of spatial autocorrelation as it produces a single value for the whole spatial pattern.

Values of Moran *I* lie in the range from $[-1, 1]$. If neighbours have similar values or classes, the Moran *I* is large. Values significantly below $\frac{-1}{n-1}$ indicate negative spatial autocorrelation and values significantly above $\frac{-1}{n-1}$ indicate positive spatial autocorrelation which means clusters exist in the graph. Cliff and Ord (1981) proved that Moran *I* statistic is asymptotically normally distributed as n tends to infinity with mean $E(I) = \frac{-1}{n-1}$, so for large $n, E(I) \approxeq 0$.

In our study, we choose the Moran *I* statistic as an indicator of clustering in graphs. If Moran *I* is large, this means the existence of clustering. While if Moran *I* is close to 0, it is an indicator of randomness between nodes classes distribution

Our experiment covers the same graph models with different values of w. We calculate the Moran *I* value for each w value in the range $(1, 50)$, and show the results in scatter plots in Figure 5.8. From these figures, we conclude that the values $w = \{1, \ldots, 50\}$ produce a range of Moran *I* values which correspond to different degrees of clustering.

We could notice from Figure 5.8 that the FF model and the BA model reach higher Moran *I* values (higher than 0.8) than the WS model (the higher value is approximately 0.7) and the ER model (the higher value is approximately 0.5). This is because the differences in the graph models structures, for example, in the ER model, it is hard to find a well separated clusters over the graph nodes due to higher transitivity in the graph model, while in the BA model, it is possible to find well separated clusters as the transitivity always equals 0.

In our study, we use BTD to distribute classes over graph nodes in our simulations.

5.7 Learning strategy for estimating the smoothing parameter

Most real network data have some labelled nodes, and we aim to predict the labels of the rest of the nodes with the best classification accuracy rate. Using a nonparametric kernel classification method requires a smoothing parameter, which plays an important role in classification

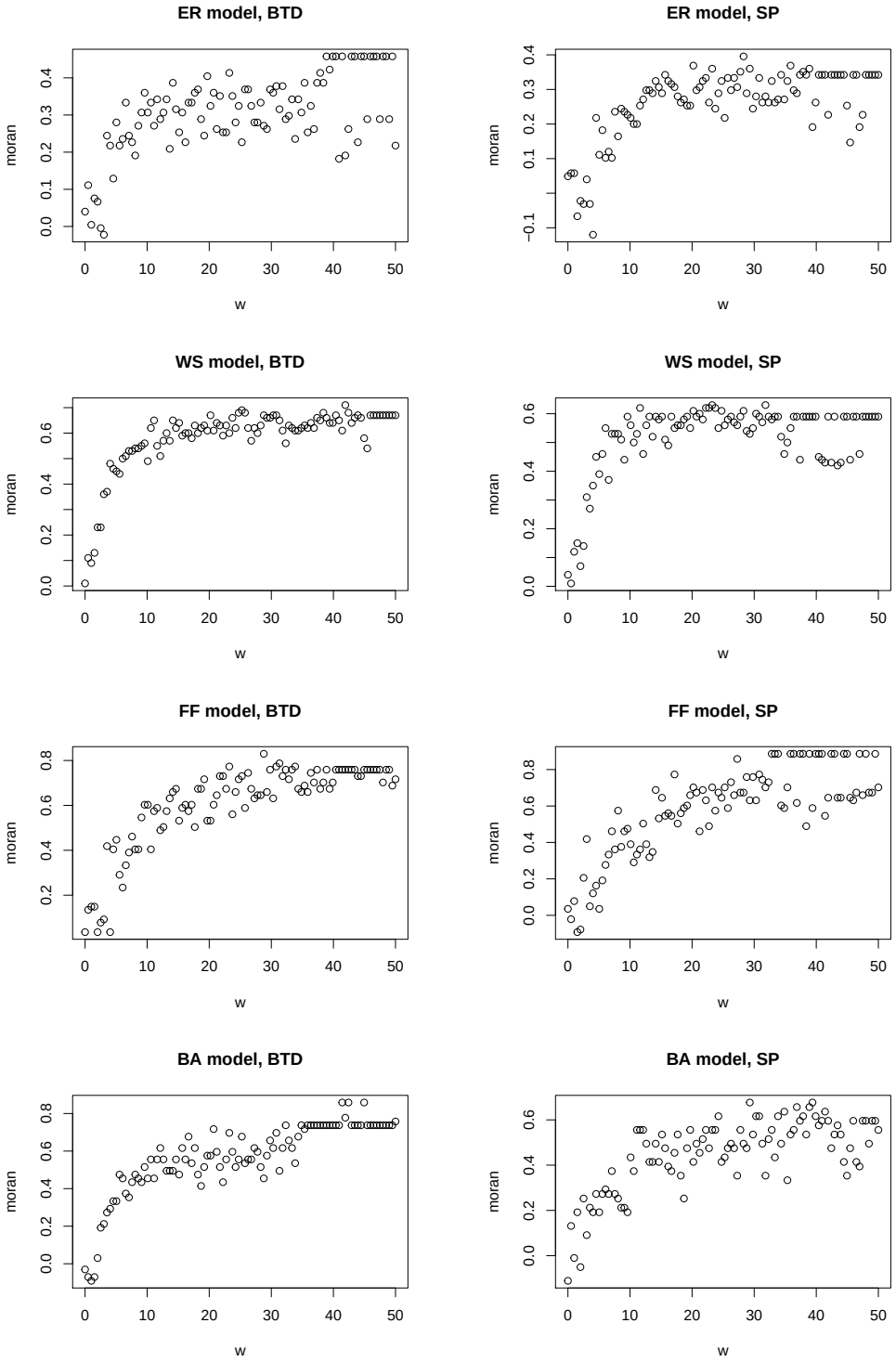


Figure 5.8: plots of the relationship between Moran I statistic and different values of w, which controls the degree of clustering in graphs.

accuracy. The best smoothing parameter is usually chosen by learning from the labelled nodes. Therefore, we have the possibility to imagine that some of the labelled nodes are unlabelled, and divide the labelled nodes into training and testing sets. We use the training set to learn and optimize the h smoothing parameter which then produces a good prediction of the testing set. We are looking to optimize the best smoothing parameter for this test set by learning over the training set. This could save the time of training h over all the data set as we did in our previous experiment. Dividing the data to training and testing sets could be based on different strategies.

In this section, we discuss different sizes and selection methods of the training and testing sets. First, we experiment with nine different sizes of the division starting from training set size equal to 10% of the nodes to 90% of nodes which corresponding to test sets of size starting from 90% of the nodes to 10% of nodes, and the results are given in Figures 5.9, 5.10, 5.11, and 5.12. Also, we apply three different methods of selecting the training sets: random choice, super nodes, or from the leaves. This also appears in the same figures. The experiments consist of these following steps:

1. Divide the network nodes into two sets: a training set and a testing set as described above. Then take a sub-matrix from the distance matrix based on the training set nodes.

2. Apply the kernel classification method on the training set with leave one out cross-validation over a sequence of smoothing parameters h, and choose the best h, which corresponds to the maximum probability of correct nodes classification.

3. Predict the classes in the testing set using these optimal h from the previous step by using the original distance matrix and the training set node classes.

4. Repeat these steps 10 times for each size, and calculate the averages of all accuracies over the testing sets.

Each average corresponds to a point on each curve, so we end up with nine points in each curve in each figure. The results are presented over Figures 5.9 - 5.12.

We know previously that each graph model acts in a different way from other models. Also, as the training set size increases, the classification accuracy also increases as we have

more information to use in the classification process. Another important note is that when the training set increases to 80% or 90%, this means most of the graph nodes are involved in the training set, so in the case of choosing super nodes as training, most of the entire graph nodes participate in the training and the remaining nodes are leaf nodes (or the small degree nodes). This may explain the behaviour of the (super) curves in the FF and the BA models as they have leaves (and small nodes' degrees) in their structures unlike the WS and ER models which do not have leaf nodes. Choosing super nodes as training set in the FF and the BA models may reverse the direction of the curves to decrease after the training set exceeds size 60-70%. The opposite case is when choosing the leaves as a training set and increasing the training size to 50% or more of the nodes which means the super nodes will remain in the testing set. In this situation, the (leaves) curves could stay stable or start to decrease in the FF and the BA models. However, in the WS and ER models, the curves increase from the minimum training set size until the maximum training size.

In the FF model, we can conclude that choosing super nodes for large size of training set is recommended over choosing random nodes or leaves nodes, while for small sizes of training set (e,g. 10%), we may recommend to choose leaves as the training set.

In the BA model, we conclude that choosing the training set from leaf nodes (small nodes degree) could guarantee higher accuracy classification rates than other selecting methods.

In the WS model, we do not prefer any method over other methods as they all behave similarly. However, in the ER model, we highly recommend to choose the training set from super and high degree nodes as they guarantee the highest accuracy classification rates for any training set size.

However, in real network node classification examples, for example in a static network, where there is no change in the graph structure, we recommend to check which model is most similar to the network under investigation, and choose the best way of selecting the training set with respect to the above results for different models. On the other hand, if the network is dynamic, which means the network structure is updated and changed by adding new nodes or changing nodes or edges positions, the classification strategy should be given more thought. Real networks can grow in different ways: some of them gain more density by attaching new

nodes to the super nodes. In this situation, the graph could follow the ER model. So, we may recommend to select the super nodes as training set which might produce higher accuracy classification rate. Other types of real networks are more likely to expand by adding nodes to the leaves, so the old leaves may become super nodes over time. In this situation, the network tends to produce some cycles between nodes, so it may look similar to the FF model. In this situation, choosing the higher nodes degree as the training set may help to achieve a higher accuracy classification rate. In contrast, if the network grows as a tree, it will be similar to the BA model, and here we suggest to select as we can from the lower degree's nodes as the training set to get a high chance of achieving a high accuracy classification rate.

Moreover, we recognize that most of the real network tends to have some clusters between nodes, as most social networks behave. In this situation, a different strategy of selecting training sets could applied, for example, selecting training nodes from the neighbours of the node needing prediction. Unfortunately, real networks have different and various structures and growing scenarios which make the classification task hard and difficult to follow using limited methods.

5.8 Comparison of classifiers

In this section, we show the simulation results of the classification accuracy rates of four different classifiers: the NN classifier, the fix and adaptive kernel classifiers, and SVM with the kernel trick, by using two distance measures BTD and SP. The goal of these simulations is to compare the performance of BTD and SP in each classifier, and compare between the performance of the classifiers.

We simulate from four different graph models, the FF model, the WS model , the ER model, and the BA model. From each model, we simulate 100 graphs, each with 100 nodes, and distribute the class labels as we described in Section 5.6 (by using BTD distance). We use LOOCV to estimate the smoothing parameter of each node in the graph our in all simulations.

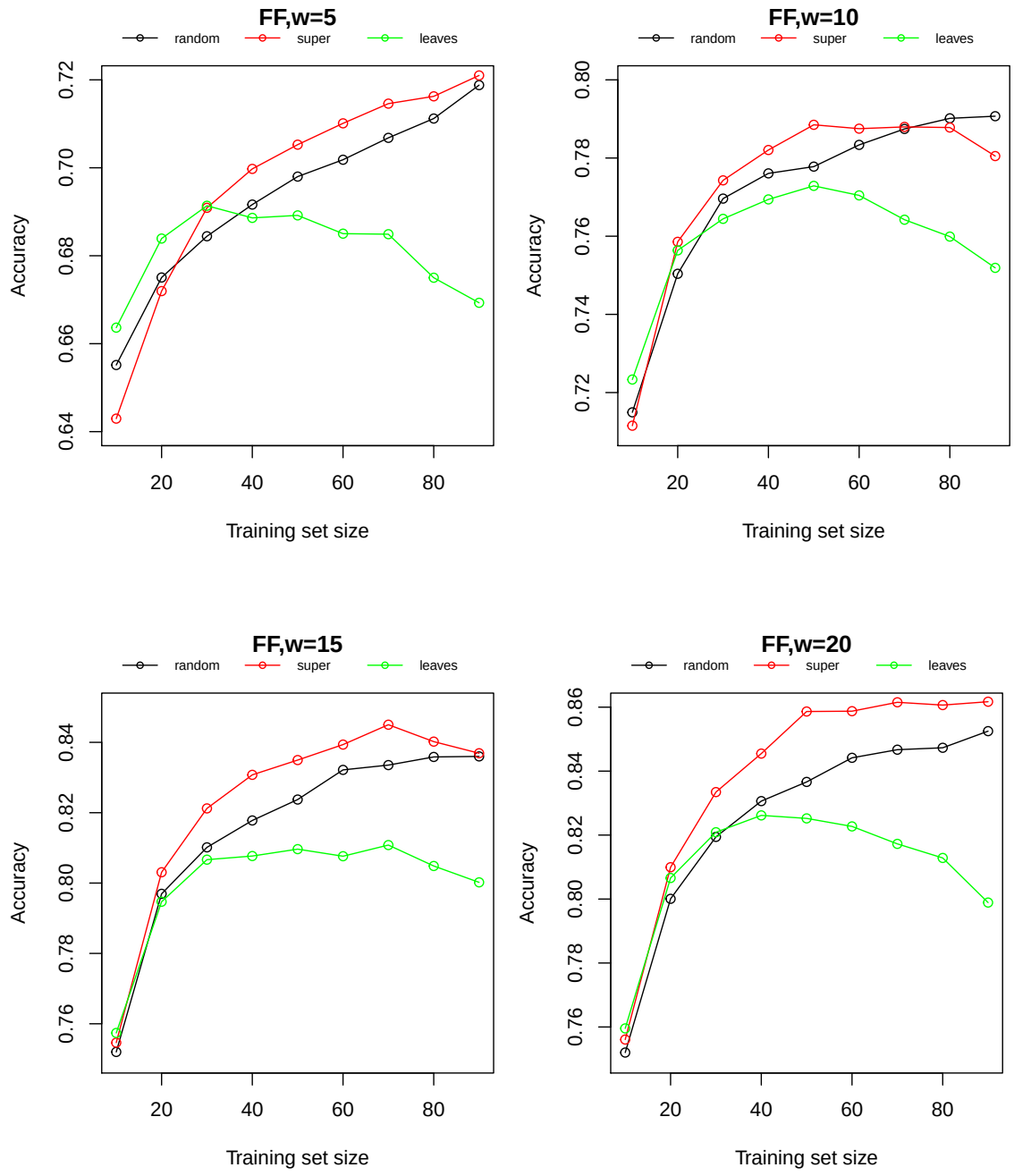


Figure 5.9: Four plots showing the behaviour of the classification accuracy with different training sets size over three different selection methods: random nodes, super nodes, and leaf nodes in FF graph model.

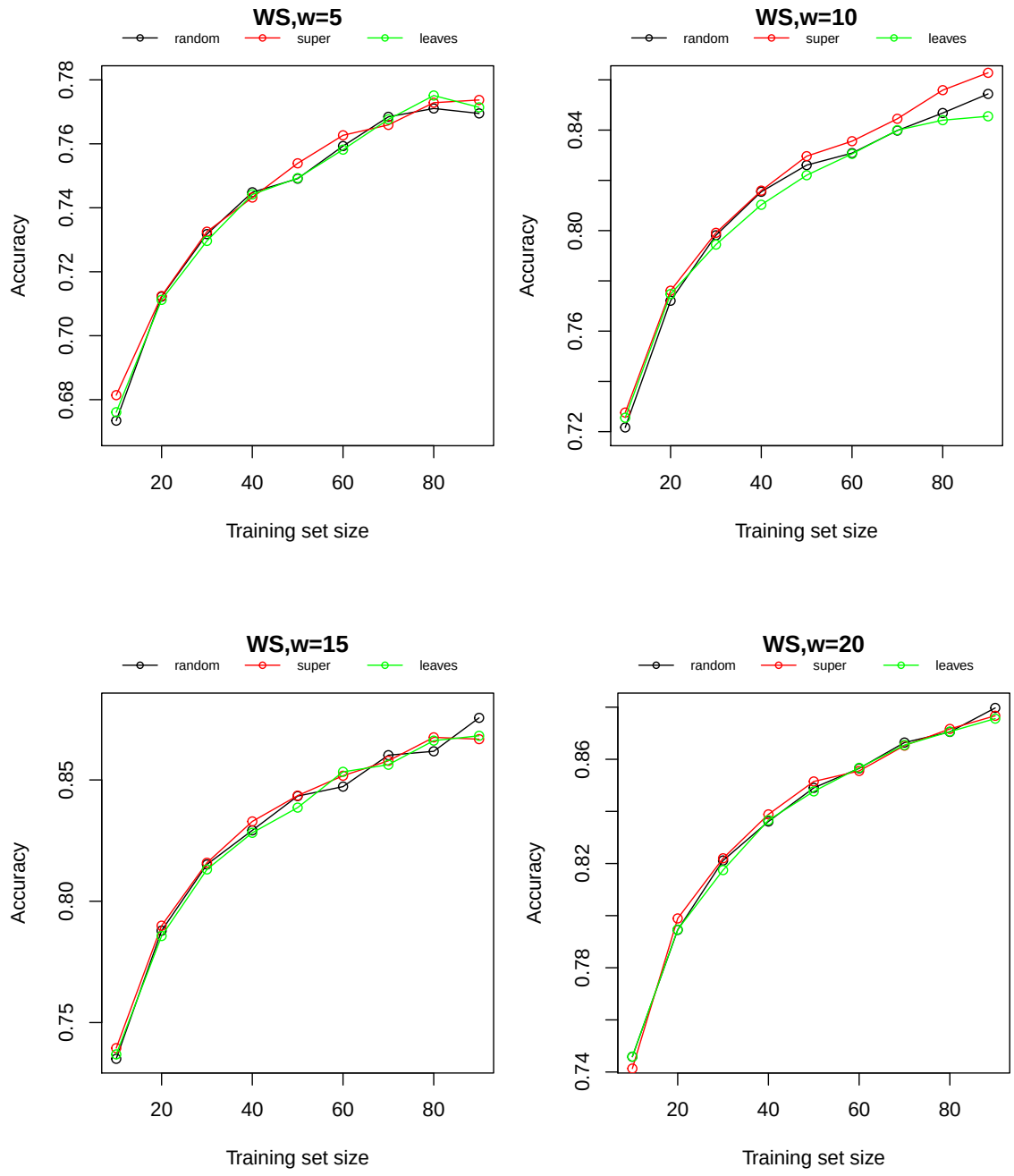


Figure 5.10: Four plots showing the behaviour of the classification accuracy with different training sets size over three different selection methods: random nodes, super nodes, and leaf nodes in WS graph model.

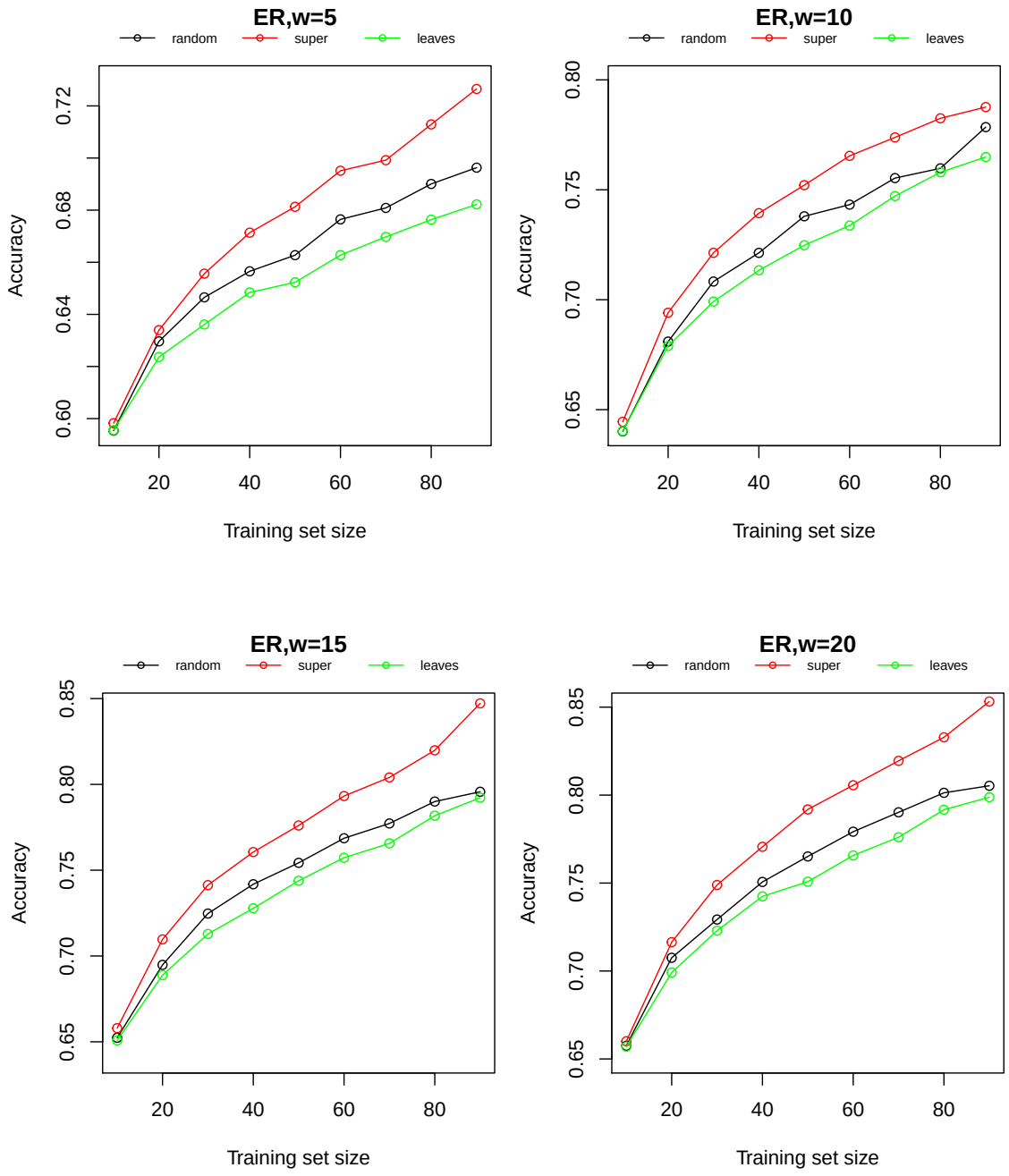


Figure 5.11: Four plots showing the behaviour of the classification accuracy with different training sets size over three different selection methods: random nodes, super nodes, and leaf nodes in ER graph model.

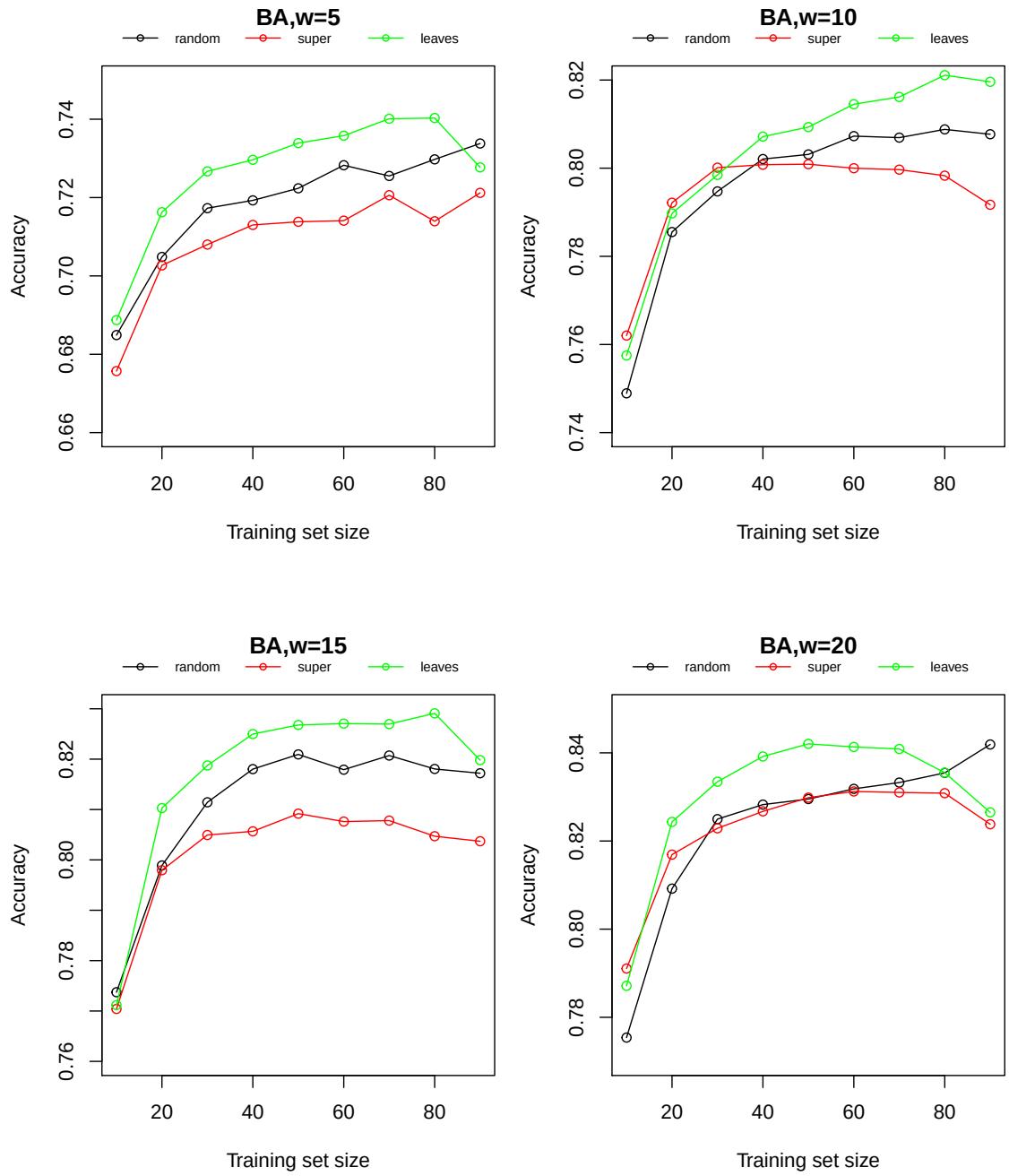


Figure 5.12: Four plots showing the behaviour of the classification accuracy with different training sets size over three different selection methods: random nodes, super nodes, and leaf nodes in BA graph model.

5.8.1 Comparison between the performance of BTD and SP in the nearest neighbour classifier

As we mentioned in Section 5.6, each simulation uses a graph model and a value of w. In each simulation, we compute the average number of neighbours for the graph nodes in each graph (*Avg*), the average nearest neighbour classifier accuracy rate (*NN*) over 100 simulated graphs, the standard deviation (*SD*) of the 100 accuracy rate in each simulation. We compare between the performance of BTD and SP in this classier by checking the p-value of the statistical t-test. Our hypotheses are

$$H_0 : \text{accuracy (BTD)} = \text{accuracy (SP)},$$

$$H_1 : \text{accuracy (BTD)} \neq \text{accuracy (SP)}.$$

From Table 5.1, we can see that, over all graph models, the average number of neighbours with BTD is always approximately 1, while with SP it is between 2 and 4. Therefore, we suggest with SP to repeat the random selection of the nearest neighbour for each node about 10 times, calculate the accuracy rate each time, then find the average of all these accuracy rates. This requires more time (about 10 times as much) as the BTD situation, which is undesirable for large graphs. In the average accuracy rates, there is no significant difference in the accuracy rates between BTD and SP amongst the FF, the ER, and the BA graph models. In this case, we recommend BTD in the nearest neighbour classifier as it is faster than SP. In the WS graph model, BTD produces higher accuracy rates compared with SP in most of the cases as we have significant p-values, so in this model, we have as evidence to reject the null hypothesis that the performance of the distances are not similar. Therefore, from these results, we can say that BTD is recommended in the nearest neighbour nodes classifier.

5.8.2 Comparing the performance of BTD and SP in fixed kernel classifier

In this section, we compare the performance of BTD and SP by using the kernel classifier over 100 simulated graphs from each model, so we consider a hypothesis to check the significance

	BTD			SP			
FF model	Avg	NN	SD	Avg	NN	SD	p-value
w=1	1.03	0.53	0.06	3.6	0.53	0.052	0.3
w=5	1.03	0.71	0.08	3.6	0.71	0.073	0.6
w=10	1.03	0.78	0.09	3.6	0.78	0.084	0.1
w=20	1.03	0.84	0.07	3.6	0.84	0.07	0.2
	BTD			SP			
WS model	Avg	NN	SD	Avg	NN	SD	p-value
w=1	1.01	0.53	0.056	4	0.53	0.05	0.8
w=5	1.01	**0.73**	0.057	4	0.72	0.05	**0.01**
w=10	1.01	**0.79**	0.05	4	0.78	0.04	**0.02**
w=20	1.01	**0.82**	0.05	4	0.81	0.04	**0.01**
	BTD			SP			
ER model	Avg	NN	SD	Avg	NN	SD	p-value
w=1	1.005	0.52	0.06	3.6	0.52	0.05	0.5
w=5	1.005	0.64	0.05	3.6	0.64	0.04	0.07
w=10	1.005	0.71	0.05	3.6	0.71	0.04	0.7
w=20	1.005	0.75	0.04	3.6	0.74	0.03	0.2
	BTD			SP			
BA model	Avg	NN	SD	Avg	NN	SD	p-value
w=1	1.001	0.52	0.05	1.98	0.51	0.06	0.16
w=5	1.001	0.68	0.07	1.98	0.69	0.07	0.06^*
w=10	1.001	0.78	0.08	1.98	0.79	0.08	0.05^*
w=20	1.001	0.83	0.09	1.98	0.84	0.09	0.1^*

Table 5.1: Simulation results of the comparison of the performance of BTD and SP in the nearest neighbour classifier among four different graph models. Avg corresponds to the average number of nearest neighbours, NN is the average accuracy rate of all simulations, and SD is the standard deviation of the accuracy rates of all simulations. $*$ in the p-value means that SP performs better than BTD.

of the difference between both distances.

$$H_0 : \text{accuracy (BTD)} = \text{accuracy (SP)}$$

$$H_1 : \text{accuracy (BTD)} \neq \text{accuracy (SP)}.$$

From the results in Table 5.2, the accuracy rate over most graph models show no significant difference between both distances as the accuracy rates are approximately similar. However, in the ER model, BTD produces higher accuracy rates than SP (with significant p-values) with small values of w. This is because BTD is able to break the ties between distances which are apparent (profusely) in the ER model.

The results lead us to not rejecting the null hypothesis, and accept both distances with this classifier.

5.8.3 Comparing the performance of the fixed kernel classifier and the adaptive kernel classifier based on BTD and SP

In this simulation study, we compare the performance of the fixed kernel classifier with the adaptive kernel using both BTD and SP, and we check the significance of the difference by testing this hypothesis

$$H_0 : \text{accuracy (Fixed)} = \text{accuracy (Adaptive)}$$

$$H_1 : \text{accuracy (Fixed)} \neq \text{accuracy (Adaptive)}.$$

Although the fixed kernel classifier is a special case of adaptive kernel, the results show that the adaptive kernel classifier does not always produce higher accuracy rates than fixed kernel. This is because some of the graph models' structures differ from the structure of the data in Euclidean space. In the Euclidean space, it is easy to see any noise or outliers in the data, so the adaptive kernel in this situation always works better than fixed kernel. The other reason is, that in LOOCV strategy, we can not guarantee to have the best smoothing parameter for the test node (unknown)based on the rest on the nodes. So, the fixed kernel might produce better accuracies compared with adaptive kernel.

However, it is worth trying the adaptive kernel with different graph models to discover its

FF model	BTD	SP	p-value
$w = 1$	0.53	0.53	0.09
$w = 5$	0.72	0.72	0.1^*
$w = 10$	0.79	0.78	0.06
$w = 20$	0.85	0.85	0.2
WS model	BTD	SP	p-value
$w = 1$	0.55	0.55	0.1^*
$w = 5$	0.77	0.77	0.05^*
$w = 10$	0.83	0.83	0.11
$w = 20$	0.87	0.87	0.56
ER model	BTD	SP	p-value
$w = 1$	**0.52**	0.51	0.04
$w = 5$	**0.62**	0.61	0.02
$w = 10$	0.76	0.75	0.1
$w = 20$	0.81	0.81	0.1
BA model	BTD	SP	p-value
$w = 1$	0.53	0.53	0.3
$w = 5$	0.70	0.71	0.1^*
$w = 10$	0.80	0.80	0.2
$w = 20$	0.83	0.83	0.1

Table 5.2: Simulation results of the comparison of kernel classifier accuracy rate between BTD and SP, graph size equal 100, simulation size =100. $*$ in the p-value means that SP performs better than BTD.

performance. Results are given in Table 5.3. We also notice from the results that the optimized values a and b are different among the nodes which means that each node chooses the best pair of a and b to have the optimal h which therefore leads to better classification accuracy rates.

In general, there is no significant difference between the performance of BTD and SP in most of the cases. Some models with some values of w produce higher accuracy rates with BTD compared with SP, while some other cases prefer SP to BTD. In the comparison between the fixed kernel and adaptive kernel, we notice that in the FF and ER models with BTD, the fixed kernel outperforms the adaptive kernel with $w = \{5, 10, 20\}$, while with $w = 1$, both methods perform similarly. However, for the WS model, both methods produce equal average accuracy rates. In the case of SP, the adaptive kernel performs better than the fixed kernel in the FF and WS models with $w = \{5, 10\}$ and in the BA model with $w = \{10, 20\}$, while in the ER model, the fixed kernel performs better than adaptive kernel with $w = \{1, 5\}$. Overall, the average accuracy rates among all models are varied, starting from lowest value in the ER model following by the FF, BA, and WS models respectively. This variation could be affected by different parameters, for example, the graph size, the number of classes, and the degree of the transitivity in the graph. In this situation, there is no obvious preference for fixed than adaptive kernel (or vice versa), but this situation might become clearer with a more thorough investigation.

5.8.4 Support vector machines with the kernel trick

In this simulation study, we use multidimensional scaling (MDS) to reduce the distance matrix dimension and represent the nodes in low-dimensional coordinate space (equal 4). This allow us to use the SVM classifier over the nodes in a graph (see Section 5.5.5).

The results given in Table 5.4 are based on NNMDS to 4 dimensions. In each simulation, the same graph is used for both distances BTD and SP.

	BTD			SP		
FF model	Adaptive	Fixed	p-value	Adaptive	Fixed	p-value
$w = 1$	0.53	0.53	0.3	0.54	0.53	0.06
$w = 5$	0.70	**0.71**	**0.005**	**0.73**	0.71	**0.0003**
$w = 10$	0.78	**0.79**	**0.002**	**0.79**	0.78	**0.01**
$w = 20$	0.84	**0.85**	**0.002**	0.85	0.85	0.1
WS model	Adaptive	Fixed	p-value	Adaptive	Fixed	p-value
$w = 1$	0.55	0.55	0.4	0.55	0.55	0.4
$w = 5$	0.77	0.77	0.1	**0.78**	0.77	**0.05**
$w = 10$	0.84	0.84	0.06	**0.85**	0.84	**0.01**
$w = 20$	0.87	0.87	0.06	0.87	0.87	0.1
ER model	Adaptive	Fixed	p-value	Adaptive	Fixed	p-value
$w = 1$	0.51	0.51	0.7	0.49	**0.51**	$\mathbf{1.3 \times 10^{-5}}$
$w = 5$	0.59	**0.61**	**0.01**	0.59	**0.60**	**0.006**
$w = 10$	0.65	**0.66**	**0.002**	0.65	0.65	0.7
$w = 20$	0.68	**0.69**	**0.001**	0.68	0.68	0.9
BA model	Adaptive	Fixed	p-value	Adaptive	Fixed	p-value
$w = 1$	0.54	0.54	0.1	0.54	0.54	0.2
$w = 5$	0.70	0.70	0.1	0.75	0.71	1.7×10^{-12}
$w = 10$	0.80	0.80	0.2	**0.82**	0.80	$\mathbf{5.7 \times 10^{-6}}$
$w = 20$	0.83	0.83	0.4	**0.85**	0.83	$\mathbf{9.5 \times 10^{-6}}$

Table 5.3: Simulation results of the comparison between fixed kernel classifier accuracy rate and adaptive kernel accuracy rate, $\alpha = 100$ (transformation parameter), graph size equal 100, simulation size equal 100. $*$ in the p-value means that SP performs better than BTD.

To check the significance of the difference between these distances we test this hypothesis

$$H_0 : \text{accuracy (BTD)} = \text{accuracy (SP)}$$

$$H_1 : \text{accuracy (BTD)} \neq \text{accuracy (SP)}$$

The results of the FF, WS and BA models show no significant differences between the performance of BTD and SP over most values of w. The ER model shows performance in favour of BTD over SP for all w values. From this result, we could suggest to check the graph structure for any given data, and if it appears to follow the ER graph model, so the BTD is better to use with the SVM classifier. Otherwise, there is no problem using any distance.

5.8.5 Comparing the performance of all methods based on BTD and SP

In this section, we compare the performance of the four different classifiers: NN, fixed kernel, adaptive kernel, and SVM with the kernel trick. We simulate 100 graphs as we describe in Section 5.6, and use the same simulated graph with all classifiers. In this experiment, we do not set any hypothesis to check the significant differences between the methods' performances, instead we consider the ANOVA test to check the differences between all these classifiers together. The results are presented in Table 5.5 and Figures 5.13 - 5.16.

- In the FF graph model, Figure 5.13 shows no significant differences between the classifiers with $w = 1$ over both distances. However, with $w = 10$ the SVM has significant improvement over the NN classifier for both distances. It reaches 0.79 (BTD), and 0.80 (SP) in Table 5.5. The adaptive kernel performs significantly better than NN classifier in SP (0.80). For $w = 20$, the differences are only significant between SVM and NN in SP and reaches 0.87.

- In the WS graph model, all classifiers outperform the NN classifier as we can see from Table 5.5 and the ANOVA test in Figure 5.14. For $w = 1$, the SVM accuracy reaches a higher value (0.56) over both distances. However, for $w = 10$, the differences are significant as all classifiers reach 0.85 compared with 0.79 with NN classifier. The 2^{nd} row in Figure 5.14 supports this result, that the first three lines from each graph are in the positive part of the real line. This is a significant sign of the superiority of all classifiers

FF model	BTD	SP	p-value
w=1	0.53	**0.54**	0.01^*
w=5	0.73	0.73	0.6
w=10	0.81	0.81	0.5
w=20	0.86	0.86	0.8
WS model	**BTD**	**SP**	**p-value**
w=1	0.55	0.55	0.1
w=5	0.78	0.78	0.3
w=10	0.85	0.85	0.4
w=20	0.87	0.87	0.4
ER model	**BTD**	**SP**	**p-value**
w=1	**0.52**	0.51	**0.05**
w=5	**0.70**	0.68	**0.003**
w=10	**0.78**	0.74	**0.004**
w=20	**0.84**	0.82	**0.001**
BA model	**BTD**	**SP**	**p-value**
w=1	0.55	0.55	0.7
w=5	0.72	0.71	0.06
w=10	0.81	**0.82**	**0.1***
w=20	0.85	**0.86**	**0.2***

Table 5.4: Simulation results of the comparison of kernel trick classifier accuracy rates by BTD and SP, where w is a parameter which controls the distribution of the classes in the graph, graph size equal 100, simulation size equal 100, number of dimensions in MDS equal 4. $*$ in the p-value means that SP performs better than BTD.

compared with the NN classifier. For $w = 20$, the results show the superiority of all classifiers compared with NN classier (similar to $w = 10$).

- In the ER model, the overall results from Table 5.5 show that BTD produce better accuracies compared with SP over all w values. For $w = 1$, there is no significant difference between the classifiers as we can see from Figure 5.15 (1^{st} row), that the mean of all lines is around 0 in the real line. For $w = 10$, the 2^{nd} row of Figure 5.15 shows significant differences between all classifiers compared to NN classifier over both distances, also SVM exceeds kernel classifiers in BTD which reaches 0.80.

  For $w = 20$ the behaviour of the classifiers are identical to the case with $w = 10$, which appears in Figure 5.15 (3^{rd} row).

- Finally, in the BA model, SVM always exceeds the NN classifier in SP over all w values which reaches $0.56, 0.83$,and 0.87 respectively as we can see in Table 5.5. However, for $w = 10$, the adaptive kernel classifier outperforms the NN classifier in SP.

The overall conclusion is that these results show a preference to SVM compared to the rest of classifiers. However, choosing the appropriate classifier/distance in any network data could consider more investigation. For example, if the data transitivity is considered as high, we expect more ties between nodes and the structure may vary between the WS and ER models. In this situation, we could recommend using SVM classifier with BTD. If the transitivity is low or 0, we could recommend using SVM with SP. More investigations should be done with large networks and real data before suggesting any classifier.

	BTD				SP			
FF model	NN	Kernel	Adaptive	SVM	NN	Kernel	Adaptive	SVM
w=1	0.53	0.53	0.53	0.53	0.52	0.53	**0.54**	0.53
w=10	0.76	0.78	0.78	0.79	0.76	0.78	**0.80**	**0.80**
w=20	0.83	0.85	0.85	0.86	0.83	0.85	0.86	**0.87**

	BTD				SP			
WS model	NN	Kernel	Adaptive	SVM	NN	Kernel	Adaptive	SVM
w=1	0.53	0.55	0.55	**0.56**	0.53	0.55	0.55	**0.56**
w=10	0.79	**0.85**	**0.85**	**0.85**	0.79	**0.85**	**0.85**	**0.85**
w=20	0.82	**0.88**	**0.88**	**0.88**	0.82	0.87	**0.88**	0.87

	BTD				SP			
ER model	NN	Kernel	Adaptive	SVM	NN	Kernel	Adaptive	SVM
w=1	**0.52**	**0.52**	**0.52**	0.51	**0.52**	0.51	0.51	0.51
w=10	0.71	0.78	0.77	**0.80**	0.71	0.77	0.78	0.78
w=20	0.75	0.82	0.81	**0.84**	0.75	0.81	0.81	0.82

	BTD				SP			
BA model	NN	Kernel	Adaptive	SVM	NN	Kernel	Adaptive	SVM
w=1	0.52	0.53	0.53	0.55	0.52	0.53	0.54	**0.56**
w=10	0.79	0.80	0.80	0.82	0.79	0.80	**0.83**	**0.83**
w=20	0.83	0.84	0.84	0.86	0.83	0.84	**0.87**	**0.87**

Table 5.5: Simulation accuracy rates results of comparison between the performance of BTD and SP and comparison between four different classifiers among four different graph models. Each graph has 100 nodes, and there are 100 simulations per model.

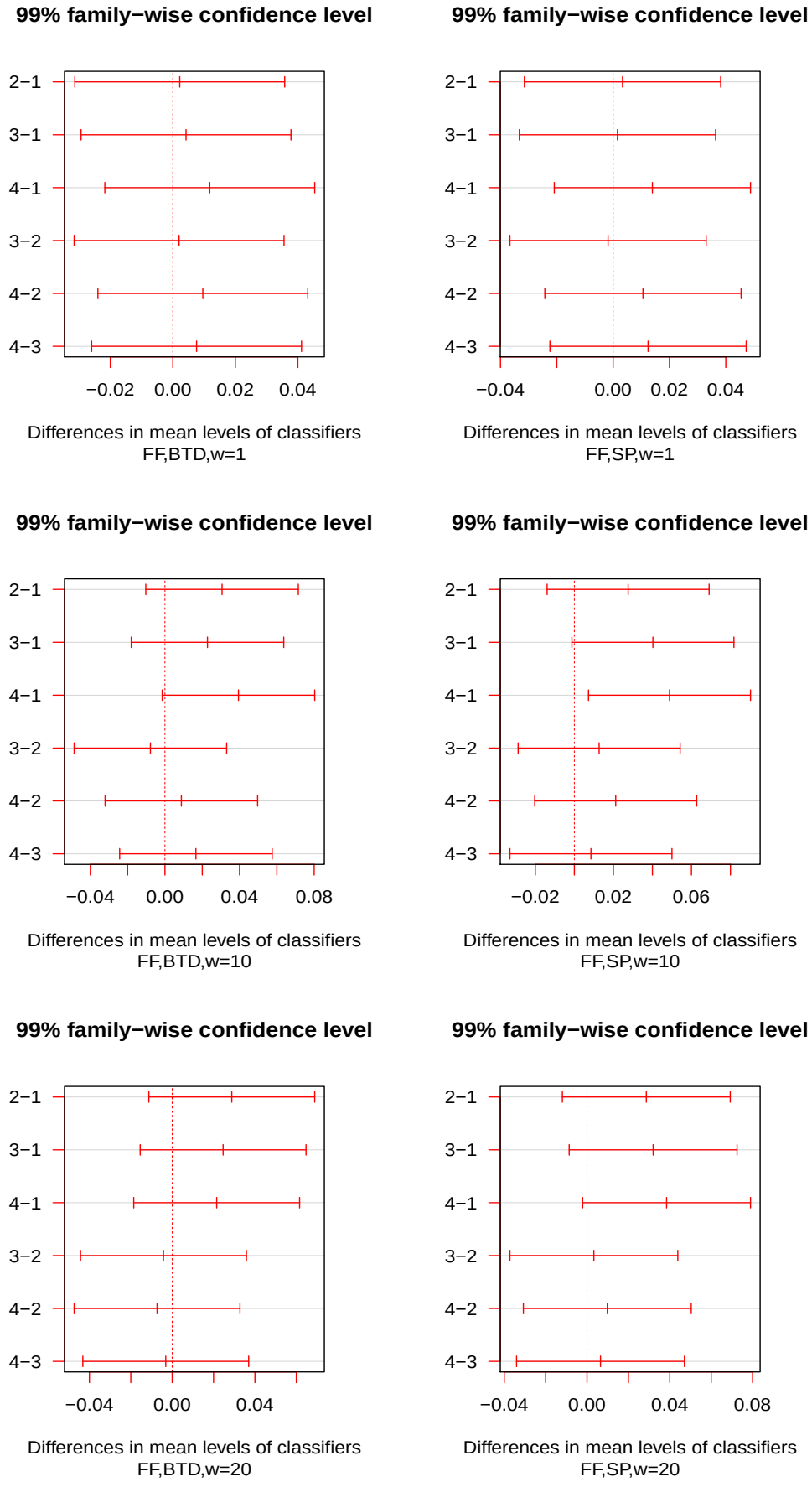


Figure 5.13: ANOVA test for the FF model. the first plots in the first column are related to BTD, and the second column are related to SP for different values of w. The vertical axis labels correspond to the classifier order in Table 5.5.

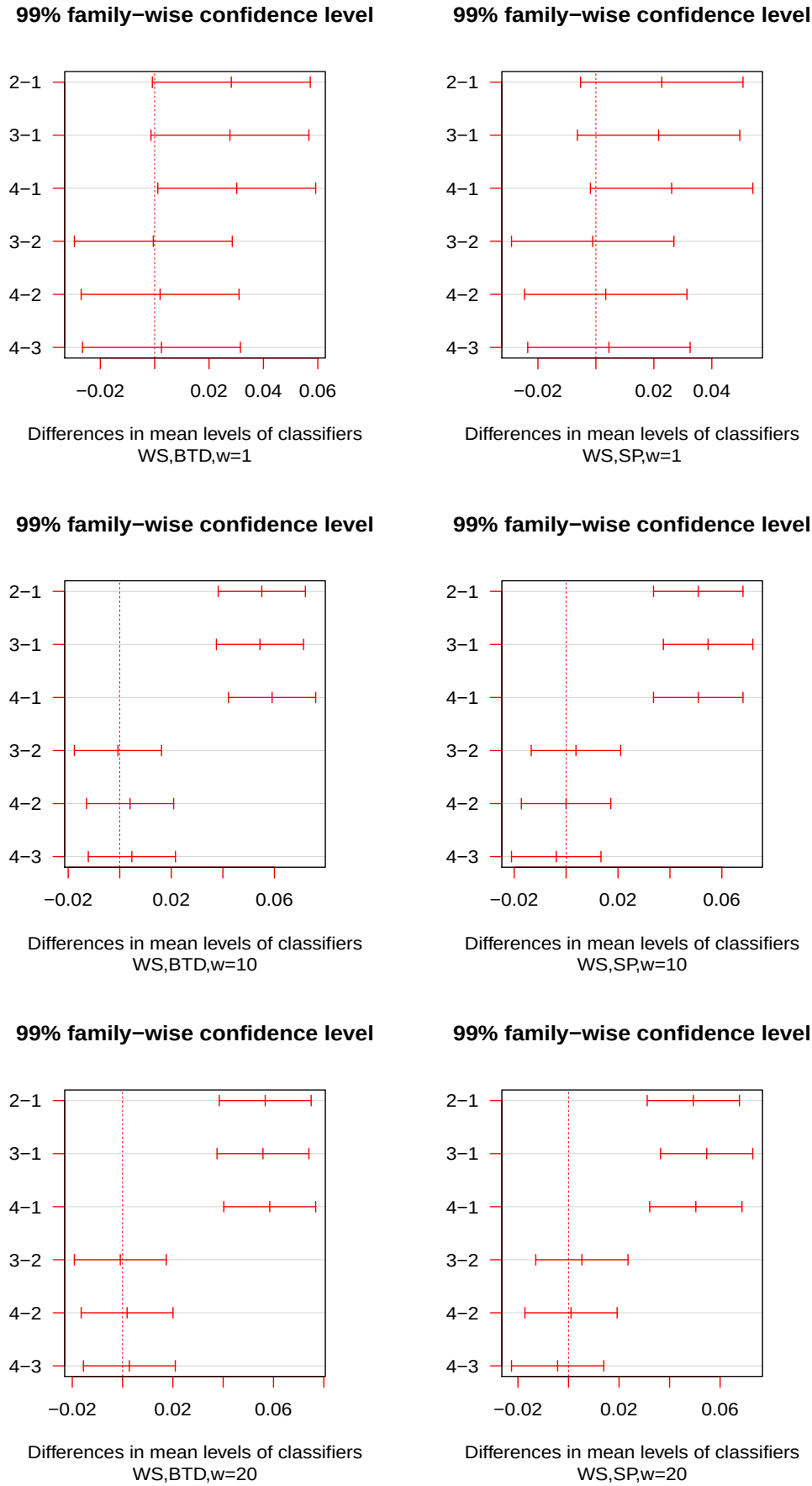


Figure 5.14: ANOVA test for the WS model. the first plots in the first column are related to BTD, and the second column are related to SP for different values of w. The vertical axis labels correspond to the classifier order in Table 5.5.

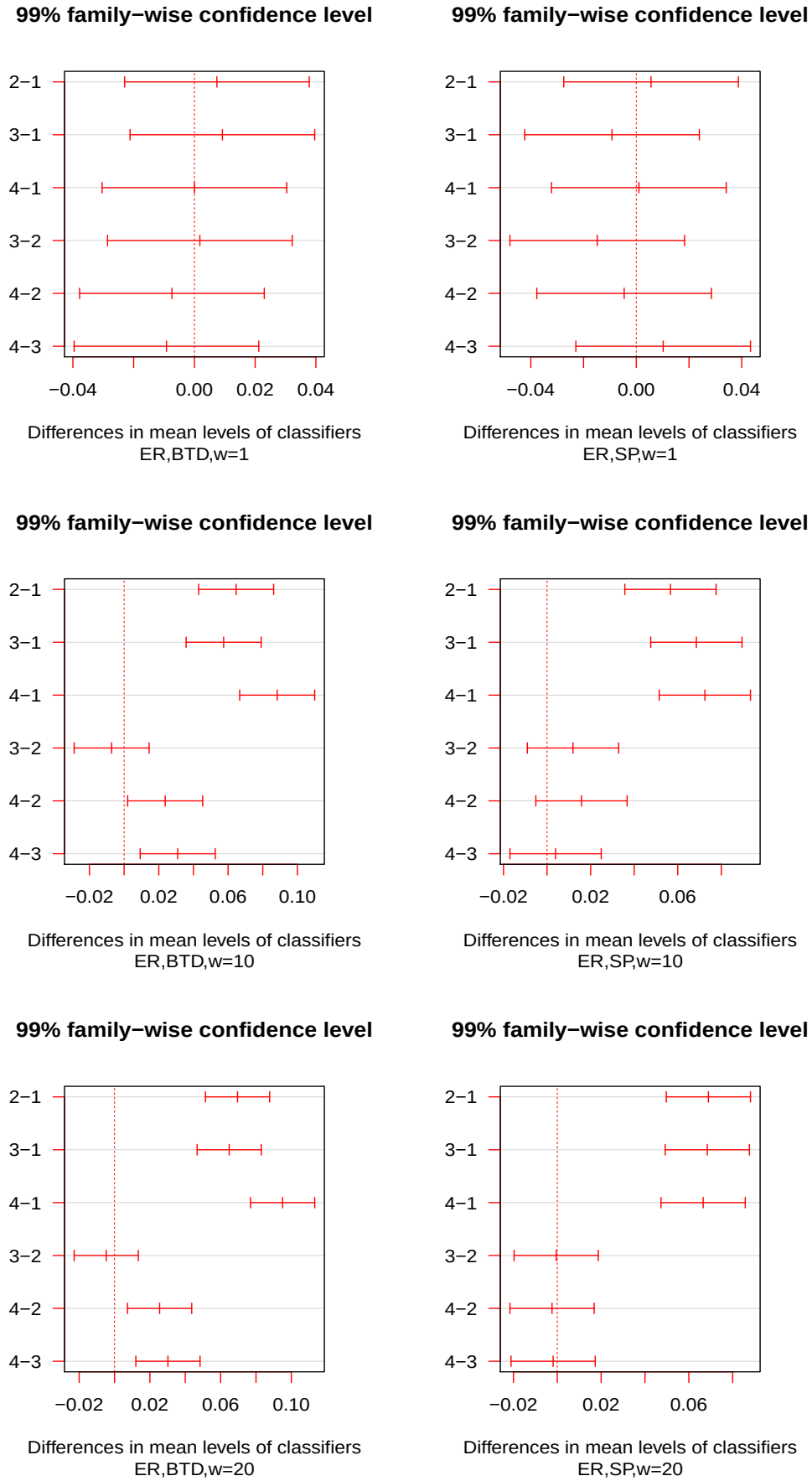


Figure 5.15: ANOVA test for the ER model. the first plots in the first column are related to BTD, and the second column are related to SP for different values of w. The vertical axis labels correspond to the classifier order in Table 5.5.

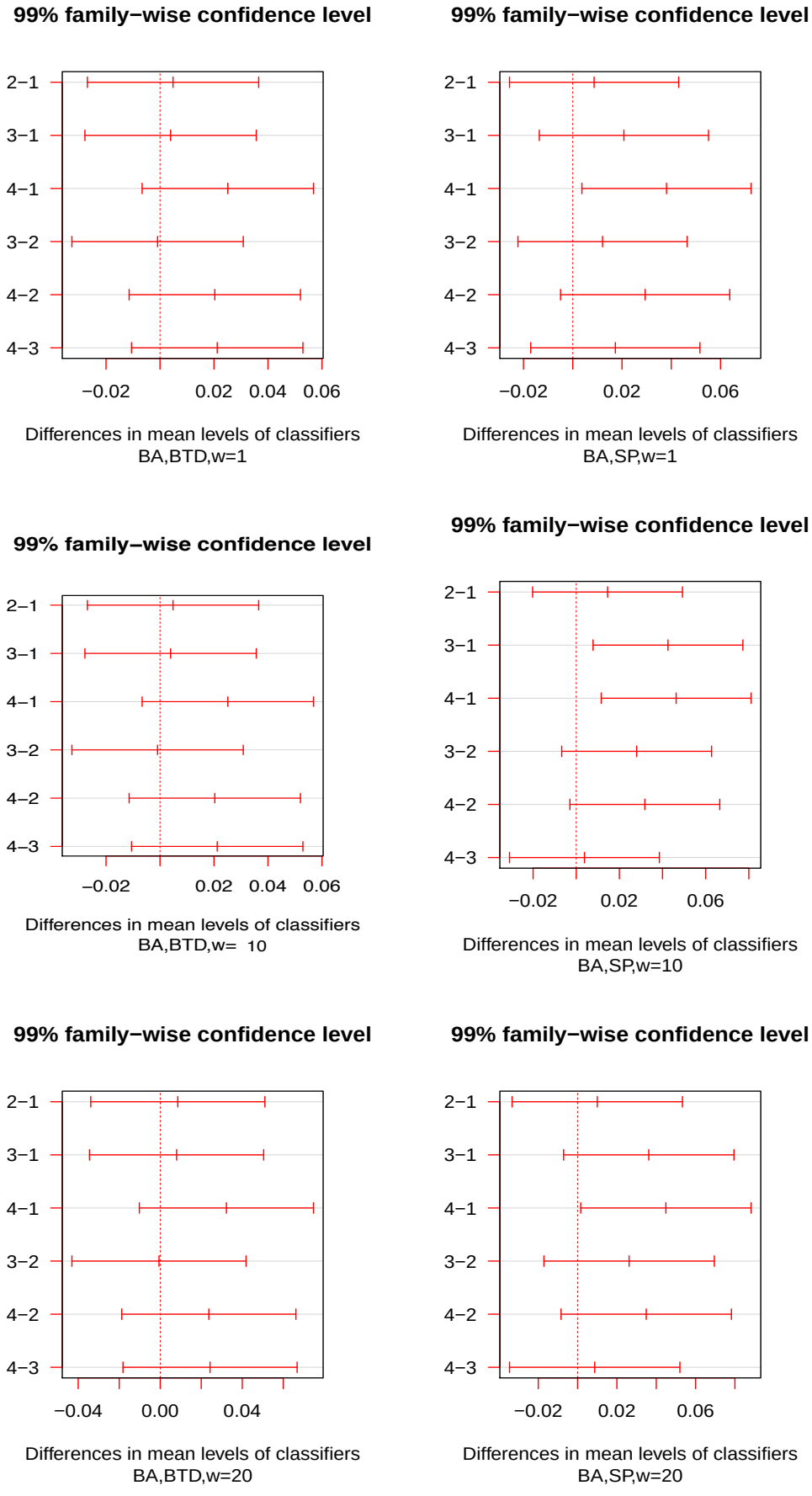


Figure 5.16: ANOVA test for the BA model. the first plots in the first column are related to BTD, and the second column are related to SP for different values of w. The vertical axis labels correspond to the classifier order in Table 5.5.

5.9 Application to real data

5.9.1 Facebook network

In this section, we choose a connected component from the Facebook network which has 547 nodes and 5706 edges. Figure 5.17 shows the structure of the network (in minimum scale) with gender classes in two different colours, it has 380 nodes for the 1^{st} class and 167 nodes for the 2^{nd} class.

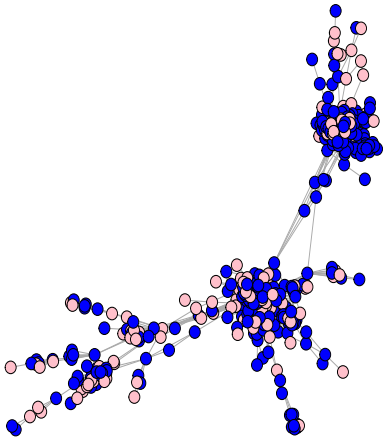


Figure 5.17: Connected component from the Facebook network which consists of the first and second egos. Blue and pink colours correspond to the gender classes.

We check the classification accuracy rates of our four classifiers; NN, fixed and adaptive kernel, and SVM with kernel trick on this graph, compared with the naive classifier where the accuracy rate is simply equal to the majority class percentage $380/547 = 0.694$.

Starting with the NN classifier, we find that by using SP, each node has a lot of adjacent nodes which means the existence of ties between each node and its neighbour. We examine the scaled standard deviation between the number of nearest neighbours over all graph nodes and make a comparison between BTD and SP variations. This means we measure each distance on this network, then find the number of nearest neighbours $|l^{(i)}|$ for each node v_i which satisfies $d(v_i, l^{(i)}) = \min d(v_i, v_j), \forall v_i, v_j \in G$, where $l^{(i)}$ means the nearest neighbours of node v_i. In

SP, $|l^{(i)}|$ means the number of adjacent nodes to v_i. By looking for $|l^{(i)}|$ in each distance, in BTD $|l^{(i)}|$ is always equal to 1, or 2 in some extreme cases, and rarely reaches 3, while in SP $|l^{(i)}|$ always lies in $[1, 99]$. By calculating the scaled standard deviation of $|l^{(i)}|$ for each distance, we find that it equals to 0.32 with BTD, and equals to 0.95 with SP. This massive difference (approximately triple) is evidence of the BTD power in breaking ties and distinguishing between equal distances in SP.

As before, to calculate the accuracy rate of the NN classifier, we repeat the NN 10 times and each time select one neighbour randomly for each node. We get 10 different accuracy rates, then find the mean of these rates as the NN classifier accuracy rate. The NN accuracy rate with BTD is equal to 0.676, and with SP is equal to 0.639, which suggests BTD with the NN classifier is recommended for other connected components in the Facebook network or any other similar real networks. However, there is another reason which makes BTD favourable compared to SP, that is for very large networks, repeating the method several times is expensive as it consumes more time with SP, while with BTD, there is no need to repeat the method as it detects the correct nearest neighbour.

Although these results are still below the naive estimator's accuracy rate 0.694, the NN accuracy may be improved by using more than one neighbour, for example, using $K = 2$ or 3 or even more as in KNN.

For the fixed kernel classifier, the accuracy rate is 0.724, with BTD and SP (the h is chosen by LOOCV method as we described in Section 5.4). While for the adaptive kernel classifier, the accuracy rate is 0.720 with BTD, and 0.718 by SP.

The results for SVM with the kernel trick seem less accurate than the kernel classifier, the accuracy rates with BTD for different dimensions $k = 2, 3, 4$ are $0.678, 0.703$, and 0.674 respectively, and with SP for the same dimensions are $0.674, 0.702$, and 0.689. All results are shown in Table 5.6.

Therefore, we could recommend using the fixed kernel classifier as to achieve higher accuracy rate in other Facebook connected components or any other similar network structures.

	Naive	NN	Fixed Kernel	Adaptive Kernel	SVM (k=3)
BTD	0.694	0.676	**0.724**	0.720	0.703
SP		0.639	**0.724**	0.718	0.702

Table 5.6: Accuracy rates of different node classification methods applied on connected component from Facebook network.

5.9.2 Amazon network

In this section, we choose four different connected components from the Amazon network (see Figure 5.18). The description of these graphs are in Chapter 2. We apply the four different classifiers and the results are shown in Table 5.7. The result is that all classifiers generally achieve higher accuracy rates compared with the Naive classifier. However, the SVM classifier produces higher accuracy in example graphs 1 with BTD (0.49) and in example 2 with SP (0.53). In example 3, the fixed kernel has higher accuracy with SP (0.63), while in example 4, the higher accuracy is achieved by the NN classifier with SP distance (0.70).

The overall conclusion is that no classifier is recommended more than others in this experiment because the differences in the performance are small. This could be because that the graphs' sizes are small which makes the decision much harder. Another reason is the multi-classes in these graph, in examples 1, 2, and 3 there are four different (unequal) classes distributed over nodes, which makes the classification task much harder. We also check the second standard error for the naive classifier for each example, and just the third example shows significant difference between the naive classifier and the other classifiers (except SVM with SP) as it has a second standard error equal 0.137.

In similar networks, as the graphs are considered as small, we believe that the user could apply more than one classifier and compare the results between them, which would only take a short time. However, this also depends on the aim of the classification and how much accuracy is requested. If high accuracy is needed, there may be a need to check different classifiers.

	n	Naive	NN		Fixed Kernel		Adaptive Kernel		SVM(k=6)	
			BTD	SP	BTD	SP	BTD	SP	BTD	SP
Example 1	66	30/66=0.45	0.48	0.42	0.45	0.45	0.47	0.45	**0.49**	0.43
Example 2	46	18/46=0.39	0.50	0.47	0.48	0.48	0.48	0.48	0.52	**0.53**
Example 3	51	21/51=0.41	0.61	0.56	0.61	**0.63**	0.61	0.61	0.57	0.49
Example 4	69	43/69=0.63	0.69	**0.70**	0.69	0.69	0.69	0.69	0.65	0.65

Table 5.7: Accuracy rates of different node classification methods applied on four connected components examples from the Amazon network.

5.10 Conclusion

This chapter is about classification in graphs, specifically, classification of the node labels. We briefly cover some common approaches in node classification, and focus on four different classifiers: the NN, the fixed kernel, the adaptive kernel, and the SVM with kernel trick. We experiment with both simulated graphs and real networks. The simulation study covers four different graph models: the FF, WS, ER, and BA graph models, and shows that the SVM with kernel trick produces more accurate classification accuracy rates compared with other classifiers over all graph models. We also discuss the idea of different training size sets and the selection criteria over the different models. The results could help with choosing the best selection criteria of the training sets, and good size of the training set. However, these result could differ from the results in the real networks where we can not recommend a classifier to use in any real data until we do more investigations due to different reasons:

- Each graph has different structure from other graphs, even those from the same model. They could look different as we change the model parameters. It is important to check and analyse the graphs before starting the classification task. This could help to know which model is similar to this graph, so based on this investigation, we could recommend an appropriate classifier to use, and also the recommended distance method.

- The graph size is important information in the analysis. Large graphs usually achieve

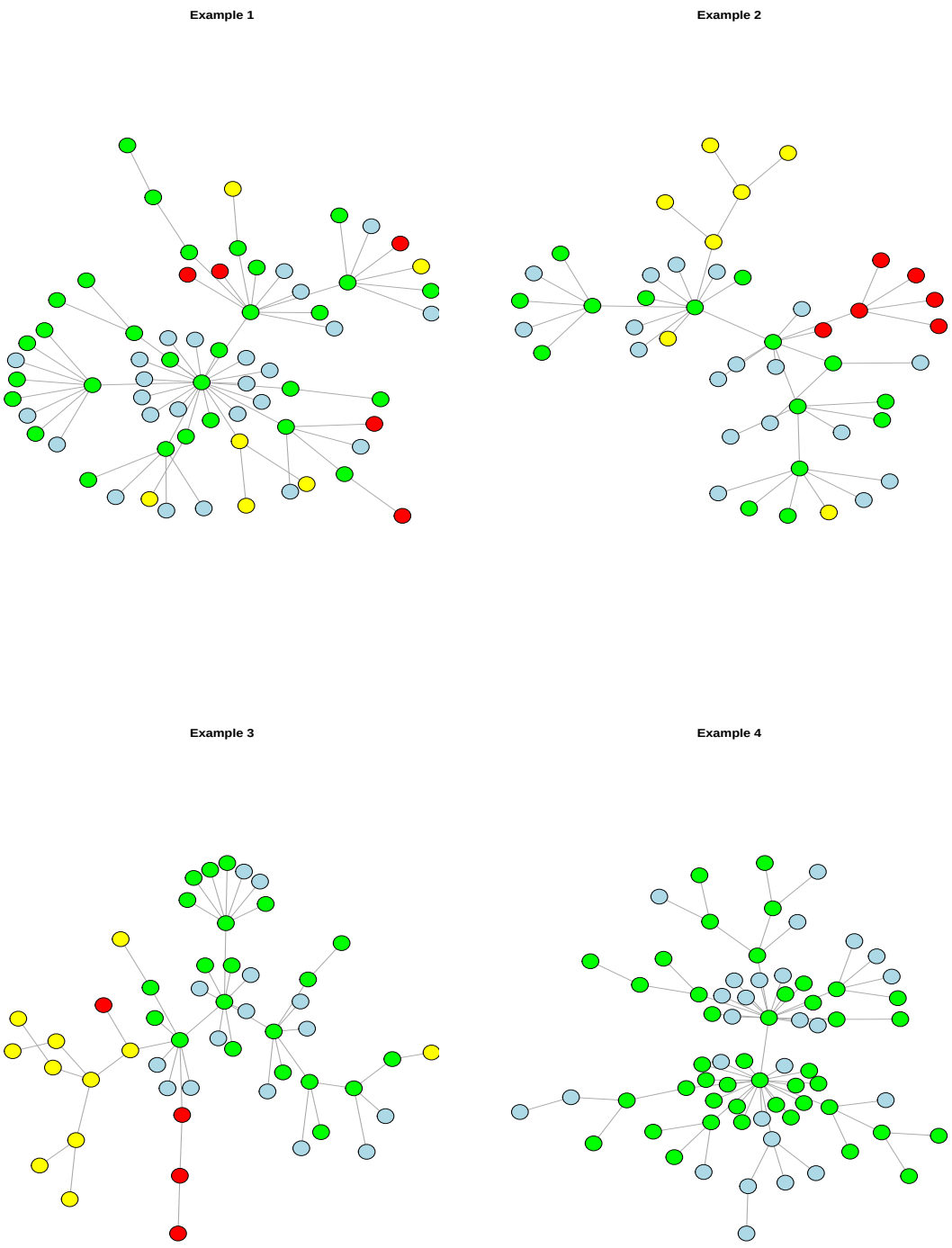


Figure 5.18: Four connected components examples from Amazon network. The colours correspond to the different classes.

higher classification accuracy than small graphs. This is because the richness of data that we can use during classification. Another reason is that in the case of existence of missing data, the large graphs could deal with these missing without any effect (or with little effect) on the results, while in small graphs, it may hard to deal with any missing information.

- The other important component in the classification task is the number of classes in the network under investigation. More classes may be harder than binary classes especially in small graphs. The relationships between classes are also important, as clustering between nodes always leads to higher accuracy rates than randomness.

- On the other hand, the classification strategy may depend on the researcher's aims and goals. Some of them look for faster results regardless the accuracy, and some look for high accuracy rate regardless of time consumption.

There are still unresolved issues involved in classification, some of which we discuss in Chapter 6.

Chapter 6

Graph nodes prediction

6.1 Introduction

We discussed in the previous chapters some machine learning tasks: clustering as unsupervised learning in Chapter 4 and classification as supervised learning in Chapter 5, where we applied some classification methods to predict node labels in some connected components in the Facebook and the Amazon networks.

During the analysis of the Amazon network, we notice that it has a richness of data and information regarding the node features, which give us plenty of scope to do specific further analysis. For example, total number of customer reviews, number of times downloaded, product ratings, and average rating of the the ratings (in this study, we refer to the average rating by "rating" to avoid any confusion). This information could provide indicators of the product quality given by the customers. Predicting the missing label for products, or for new products, could help in different fields, for example in marketing purposes, and is therefore a useful idea to be tackle.

The Amazon network could be considered as a social network, and we believe that there are relationships between adjacent products, and also between the product features (e.g. classes and ratings) themselves. We also think that there might be a relationship between the class of product and its rating, and we could use this relationship to predict the rating of any product

with respect to its class. For example, Amazon could be interested in predicting the rating for a new product (let's say a new part of a movie), where the first part of the same movie has a high rating score and a lot of reviewers. The new part has a connection with the old part due to some paired purchases between them. We could predict the rating of the new part based on the immediate neighbours (first part), then advertise the new part and recommend it to the users. Moreover, as the connected components in the Amazon tend to gather similar classes together in one component (as we discuss in Chapter 2), the new part of the movie may connect to (be co-purchased with) other movies which may have similar actors or from similar categories. This could make the prediction more accurate by knowing more information about the nearby products (nodes).

Moreover, this may be useful in a user recommendation system. For example, user X is interesting in purchasing science fiction books for some authors, so the recommendation system could recommend a new episode by the same author, and request a rating and review (from the user and previous users) for it.

Therefore, we focus in this chapter on predicting a product's rating, taking account of its class in a new and different way. We discover how strong this relation is by using a weight parameter to minimize the distances between similar classes' products and maximize the distances between different classes' products as we describe in Section 6.4. However, we notice that the ratings of the majority of products are limited within small range of values, so a simulation study is presented with a simulated graph from the BA model that will be similar to the Amazon components, in which we distribute the classes and ratings in different and new ways as we show in Section 6.5.

6.2 Data mining in graphs

Both classification and prediction are considered as supervised learning. In general, classification is a way of predicting the categorical class label by constructing a classifier, while prediction is used to predict continuous or ordered label values as opposed to categorical labels. In some research areas, prediction is called estimation, and lots of studies that we have found

referred to classification as prediction (Lu and Zhou, 2011; Jiang et al., 2015; Liben-Nowell and Kleinberg, 2007).

Most of the prediction studies in graphs focus mainly on link prediction between nodes rather than node features prediction (specially in social networks), which makes our idea of research infrequent in this field. More details about links prediction can be found in (Leskovec et al., 2010).

Few studies were found in the literature which used prediction methods. For example, Kovac and Smith (2011) discussed penalized regression in noisy images. As each pixel in the image is considered a vertex in a graph and the edges connect neighbouring pixels. They discussed penalized regression on graphs to estimate the underlying signal image.

Another study by Richard et al. (2012) addressed two prediction problems in dynamic graphs: predicting links between nodes and predicting functions on graph nodes, by introducing a hybrid approach based on the graph structure and the change in the edges and nodes features over time. They tested their method on both synthetic and real data.

We notice from the literature that some classification methods are proper for prediction tasks, for example KNN (see Section 5.5.2), but it has a problem with ties when we apply it with SP. Also, if K is more than one, it is hard to choose the predicted value in the case of equal frequencies among the neighbours.

In our study, we discuss the Nadaraya-Watson estimator as one of the most common kernel regression estimators in regression analysis, in Section 6.4.

We predict the rating of the Amazon network products with respect to the class label (product groups: book, music, movie, DVD) by considering a weight parameter between products which depends on the their classes. The main reason of this work is to check the dependency of the rating scores of the nodes on the nodes classes in the Amazon network data, in particular, and in simulated graphs which have similar structure to the Amazon network. As the Amazon network's connected components have a tree based structure, our simulated graphs are from the BA model. This is not to abandon other (non-tree) models which have similar structure to a lot of real networks, but to focus and concentrate on one basic graph model, similar to the Amazon

network.

The weight idea could be applied in different ways between nodes. For example, in the extreme case, we could suggest a binary weight $\{0, 1\}$ to be equal 0 between nodes from different classes and equal 1 between nodes from similar class. This ignores any effect of nodes from different classes and respects only the nodes from the same class. Another simple weight idea is to consider a weight between nodes from similar classes, which could be dependent or independent from the class itself without any considerations between nodes from different classes.

Another flexible weight idea is minimizing the distances between nodes from similar class and maximizing the distances between nodes from different classes. This procedure relies on the relationships between similar product's class and different product's classes without any dependency on the class itself. we consider this idea in our study in Section 6.4.

One more extreme case suggests a unique weight within each class's nodes which may depend on the class's nodes frequency, and different unique weights between pairs of different classes which may depend on each class frequency.

Another possibility is to update all distances between similar classes to be less than all distances between different classes.

6.3 Evaluation of prediction results

As we are predicting a continuous value (product rating), it is not straightforward to decide whether the predicted value is correct or not, so instead of looking for exact matching between the actual and predicted values v_i, $\hat{v}_i$, we focus on how far $\hat{v}_i$ is away from v_i. Different measures can be used in this type of evaluation (Han and Kamber, 2006). The most common measures are

- Mean absolute prediction error $\frac{\sum_{i=1}^{n} |v_i - \hat{v}_i|}{n}$.
- Mean squared prediction error $\frac{\sum_{i=1}^{n} (v_i - \hat{v}_i)^2}{n}$,

where $\hat{v}_i$ is the predicted value of v_i using all information except label and value at node v_i. In our study, we use the mean squared prediction error (MSPE) to evaluate our prediction results as it has been used in most similar studies, so we can compare our results directly with those of other researches.

6.4 Nonparametric regression

The aim of regression analysis is to find a relationship between a predictor variable x and response variable y by a function of x. In the simple case, this function constructs a linear relationship of y on x, as

$$y = \alpha + \beta x,$$

and in the case of multiple predictors (x_1, x_2), as

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2.$$

However, in some circumstances, this linear model might not fit well because of the non-linearity in the data under exploration. Nonparametric regression works more efficiently with these data by relaxing the linear assumption and substituting it with a smooth function $m(x)$, which is useful to model the underlying data by a smooth curve without parametric model.

Suppose we have a random sample of bivariate data $\{(x_i, y_i) : i = 1, ..., n\}$, the common nonparametric regression model has the form

$$y_i = m(x_i) + \varepsilon_i, \qquad\qquad i = 1, ..., n,$$

where $m(.)$ is an unknown smooth function, and the errors ε_i are independent and identically distributed, show the variation of y around $m(x)$, and satisfy

$$E(\varepsilon_i) = 0, \qquad Var(\varepsilon_i) = \sigma_\varepsilon^2, \qquad Cov(\varepsilon_i, \varepsilon_j) = 0 \quad for \quad i \neq j.$$

The idea is to find the local average of response variable Y values near a point of x, which means trying to estimate the conditional mean curve

$$m(x) = E(Y|X = x). \tag{6.1}$$

Nonparametric density estimation methods (as mentioned in previous chapter) consider an

interval around x, which depends on a bandwidth h and contains the neighbouring points around x, and nonparametric regression depends on the same technique. It calculates the weighted average of the response variable Y of the neighbourhood points of x. This can be written as

$$\hat{m}_h(x) = \frac{1}{n}\sum_{i=1}^{n} W_h(x; x_1, \ldots, x_n) y_i \tag{6.2}$$

where $W(.)$ is the weight function which depends on the sample $x_1, \ldots, x_n$ and the smoothing parameter h. The nonparametric regression smoother can be written as

$$\hat{m}_h(x) = \frac{1}{n}\sum_{i=1}^{n} W_h(x_i) y_i, \tag{6.3}$$

which was first proposed by Nadaraya (1964) and Watson (1964) and called Nadaraya-Watson estimator (Härdle, 1990). The weights $W_h(x)$ can be determined by the kernel function $K\left(\frac{x-x_i}{h}\right)$ to have the form

$$W(x_i) = \frac{K\left(\frac{x-x_i}{h}\right)}{\sum_{i=1}^{n}, K\left(\frac{x-x_i}{h}\right)}, \tag{6.4}$$

so by substituting Eq. (6.4) in Eq. (6.3), we have

$$\hat{m}_h(x) = \frac{\sum_{i=1}^{n} K_h(x - x_i) y_i}{\sum_{i=1}^{n} K_h(x - x_i)}. \tag{6.5}$$

The kernel function in Eq. (6.5) can be chosen as the common Gaussian kernel as

$$K_h(x, x_i) = \frac{1}{\sqrt{2\pi}h} exp\left(\frac{-(x - x_i)^2}{2h^2}\right) \qquad x \in \{x_1, ...x_n\}. \tag{6.6}$$

However, in graphs, we do not have any point $x \in \mathbb{R}$, instead we have nodes and distances between them, which can be used in kernel function as

$$K_h(v, v_i) = \frac{1}{\sqrt{2\pi}h} exp\left(\frac{-d(v, v_i)^2}{2h^2}\right) \qquad v \in \{v_1, ...v_n\}. \tag{6.7}$$

6.4.1 The proposed weight

In this section, we introduce a weight $(1-q)$ to modify the distance between nodes from similar class and $(1+q)$ between nodes from different classes, the weighted distance is updated as

$$d_q(v, v_i) = \begin{cases} (1-q)d(v, v_i) & \text{if } v, v_i \text{ belong to same class,} \\ (1+q)d(v, v_i) & \text{if } v, v_i \text{ belong to different classes.} \end{cases} \tag{6.8}$$

where $0 \leqslant q < 1$. This transformation allows us to decrease the distances between nodes from similar class as $0 < 1 - q \leqslant 1$, and increase the distances between nodes from different classes

as $1 \leqslant 1+q < 2$. We could use different smoothing parameters h between similar and different classes which could produce similar results. However, the parameter q in expression (6.8) can interpret and reflect the significant of the relationships between similar and different classes.

The smoothing parameter h may also play important roles in the success of the prediction. In our study, we test the smoothing parameter h over a range of values and choose the best h which corresponds to minimum MSPE (mMSPE). Another possible way is to choose h with respect to the node degree, which means that h is considered as an adaptive parameter (not fixed) for all nodes. Also, we could choose h with respect to the node class and the frequency of each class. For example, in the Amazon example 1 (see Figure 6.1), we could use a larger value of h to predict book ratings than other node classes because the book node class has less frequency than other classes. Choosing one of these ideas depends on the researcher and the aim of the study. In this study, we apply the idea which we think is more suitable to our data.

The purpose of this experiment is to investigate the relationship between node classes and ratings in the Amazon connected components, and the strength of this relationship. We apply the Nadaraya-Watson estimator over four different graph examples from the Amazon data network, whose nodes have a variety of class labels and rating scores. We describe them in Tables 6.1 - 6.4, and Figures 6.1 - 6.4. We also carry out an experiment for simulated graphs which have similar structure to Amazon graphs. The ideas can also be used to detect a relationship between any pair of features that may be related.

6.4.2 Prediction algorithm

We summarize our experiment in Algorithm 3. For a given $h > 0$, and $q \in (0, 1)$. The goal is to find h and q which minimize the MSPE. We also plot a surface which shows minimum of the surface of mean squared prediction error mMSPE over range of parameters h and q. If the mMSPE value occurs for a large value of q, this means that there is a relationship between the node classes and ratings.

We could also say that, we have a deterministic relationship between h, q, hence one inde-

Algorithm 3. Prediction of the nodes label values in $G(V, E)$

1: Compute the $(n \times n)$ distance matrix based on BTD.
2: Create a vector of node classes, and a vector of rating score $y_i, i = 1, \ldots, n$.
3: Define the kernel function

$$K_h(v_i, v_j) = \phi(v_i, v_j) = \exp\left(\frac{-d_q^2(v_j, v_j)}{2h^2}\right) \qquad 1 \leqslant i, j \leqslant n, \tag{6.9}$$

4: Set the diagonal of $K_h(v) = 0$ to predict the rating of each node on the diagonal based on the rest of the nodes ratings (LOO-CV).
5: Calculate Eq. (6.5) where y_i is the vector of rating of the nodes.
6: Calculate the Mean squared prediction error $\text{MSPE}_{h,q}$ between the predicted and original rating scores.

pendent parameter as follows:

$$\frac{d_q^2(v_i, v_j)}{2h^2} = \begin{cases} \frac{(1-q)^2 d^2(v_i,v_j)}{2h^2} = \frac{d^2(v_i,v_j)}{2h_0^2} \to h_0 = \frac{h}{1-q} & \text{if } v_i, v_j \text{ belong to same class,} \\ \frac{(1+q)^2 d^2(v_i,v_j)}{2h^2} = \frac{d^2(v_i,v_j)}{2h_1^2} \to h_1 = \frac{h}{1+q} & \text{if } v_i, v_j \text{ belong to different classes.} \end{cases}$$

However, further analysis has been considered to check the relationships between the node classes and ratings with the topological structure of the graph. So, we consider different hypotheses which are focused on such relationships as follows:

1. H_0 : Rating is independent of class.

   H_1 : Ratings distribution depends on classes.

2. H_0 : Ratings are independently distributed over graph nodes.

   H_1 : Ratings have a spatial autocorrelation.

3. H_0 : Nodes association with classes and ratings are independent of graph nodes positions.

   H_1 : Nodes association with classes and ratings have spatial autocorrelation.

4. H_0 : Nodes features (classes and ratings) are independently and randomly distributed over graph nodes.

   H_1 : Nodes features (classes and ratings) have spatial autocorrelation.

As our analysis is non-parametric, we do not have a statistical test to assess our hypothesis.

Hence, we consider different criteria to accept or reject these hypotheses based on a comparison between the mMSPE of the original structure, and the same measure in the same graphs with suggested permutations. We suggest four different permutations types, then predict the each ratings in each permuted graph. Each permutation type is repeated 100 times to to produce an average MSPE surface, which corresponds to the average MSPE values over all these 100 permutations. Then, we find the mMSPE of this surface, with the corresponding values of h and q to check our hypotheses.

We introduce two indicators based on the permutation idea which could help to build some decisions and accept or reject the assumed hypothesis. The first is the 90% quantile interval of the average MSPE values over the 100 permutations which is called QI1. The second is the 90% quantile interval of the average MSPE values over 100 permutations in the case we use the optimal pair of h and q from the original structure in each permutation, which is called QI2.

In this our study, we build our conclusions based on both indicators QI1 and QI2 (with respect to q and h values), however, it is possible to favour either QI1 or QI2 according to the researcher opinion. The permutations are described as follows:

Permutation A

In this permutation, we permute the node classes, and calculate the mMSPE with corresponding values h and q. The aim of this permutation is to check the dependency of the rating scores on the classes distribution over the graph nodes. If the mMSPE value of the original structure lies in the QI1/QI2 of this permutation, it means that the rating scores do not significantly depend on classes.

Permutation B

In this permutation, we permute the node rating scores, and calculate the mMSPE. From this experiment, we can determine if the rating scores are independently distributed over graph nodes or they have a spatial autocorrelation. If the mMSPE value of the original structure lies in the QI1/QI2 of this permutation, it means that the rating distribution do not have a spatial

correlation with the graph structure.

Permutation C

In this permutation, we permute both the node classes and ratings (as linked pairs). We can determine from this permutation if the ratings association with the nodes' classes has a spatial autocorrelation over the graph nodes by checking if the mMSPE value of the original structure lies in the QI1/QI2 of this permutation.

Permutation D

In this permutation, we permute node classes and ratings independently. If the mMSPE value of the original structure lies in the QI1/QI2 of this permutation, the node features are randomly and independently distributed over the nodes.

Assessing the hypotheses based on permutations in simulated graphs

- We test the 1^{st} hypothesis by analysing the results of permutation A. If the mMSPE value of the original structure lies in the of QI1 or QI2 of permutation A, we do not reject the null hypothesis of the 1^{st} hypothesis.
- We test the 2^{nd} hypothesis by analysing the results of permutation B. If the mMSPE value of the original structure lies in the QI1 or QI2 of permutation B, we do not reject the null hypothesis of the 2^{nd} hypothesis. So the rating scores are independently distributed.
- We test the 3^{rd} hypothesis by analysing the results of permutation C. If the mMSPE value of the original structure lies in the QI1 or QI2 of permutation C, we do not reject the null hypothesis of the 3^{rd} hypothesis.
- We test the 4^{th} hypothesis by analysing the results of permutation D. If the mMSPE value of the original structure lies in the QI1 or QI2 of permutation D, we do not reject the null hypothesis of the 4^{th} hypothesis. So, the nodes features are independently and randomly distributed over graph nodes.

The experiment results of the Amazon examples are shown in Table 6.5, each set of permutations corresponds to a row in the table. Also, the Figures 6.5 - 6.8 show 3D plot for each graph example with each permutation type.

Before starting our analysis, we mention that the 4^{th} hypothesis is more general than the others, so even if we accept the H_0 of the 4^{th} hypothesis, this does not mean that every thing is completely random, so we still have to check other hypotheses.

Rating	Book	music	DVD	Video
1.5			1	
3		1	2	3
3.5			7	3
4		2	11	11
4.5	3	3	9	8
5	2			
Average	4.7	4.08	3.88	3.98

Table 6.1: Table of Example 1 ratings frequency for each class.

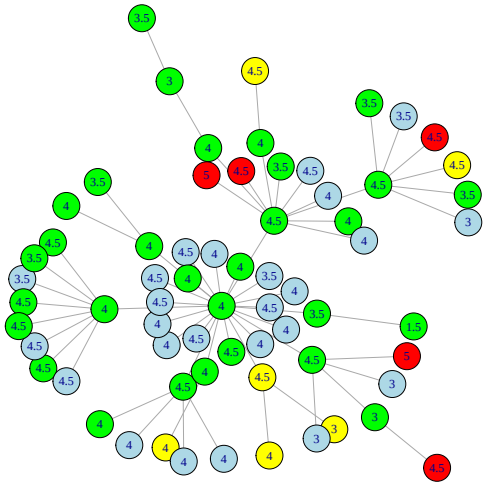


Figure 6.1: Example 1 structure with different classes as colours: book, music, DVD and video as red, yellow, green, and blue respectively, and node ratings are the labels.

Rating	Book	music	DVD	Video
1	1			
3	3		1	1
3.5	2		2	5
4		1	7	5
4.5		2	5	7
5		4		
Average	2.8	4.71	4.03	4

Table 6.2: Table of Example 2 ratings frequency for each class.

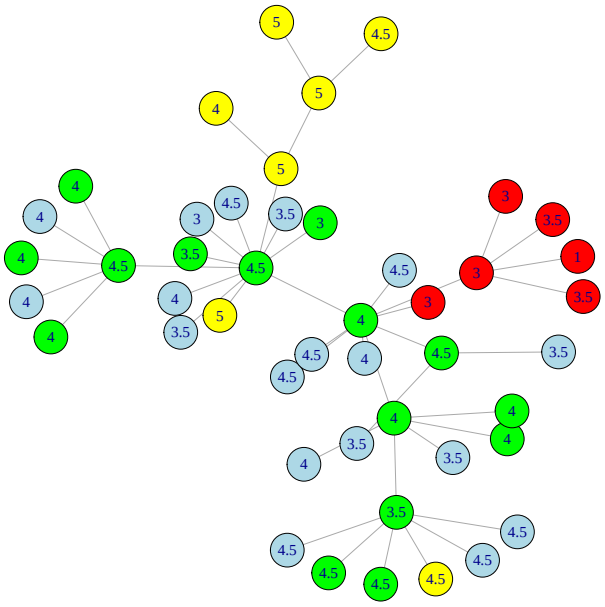


Figure 6.2: Example 2 structure with different classes as colours: book, music, DVD and video as red, yellow, green, and blue respectively, with node labels as ratings.

Rating	Book	music	DVD	Video
2			1	1
2.5		1		
3			2	1
3.5		1	2	2
4	1	2	4	4
4.5	1	3	12	7
5	1	4		
Average	4.5	4.32	4.04	3.96

Table 6.3: Table of Example 3 ratings frequency for each class.

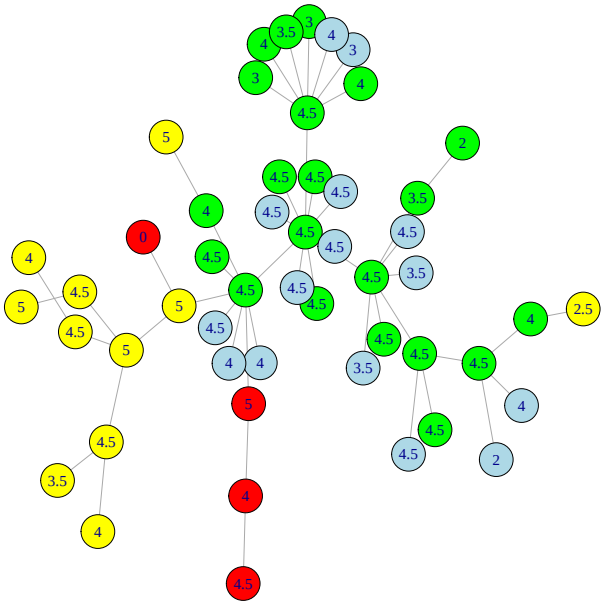


Figure 6.3: Example 3 structure with different classes as colours: book, music, DVD and video as red, yellow, green, and blue respectively, with node labels as ratings.

Rating	DVD	Video
2	1	1
2.5	2	
3	5	1
3.5	14	7
4	15	12
4.5	5	5
Average	3.65	3.84

Table 6.4: Table of Example 4 ratings frequency for each class.

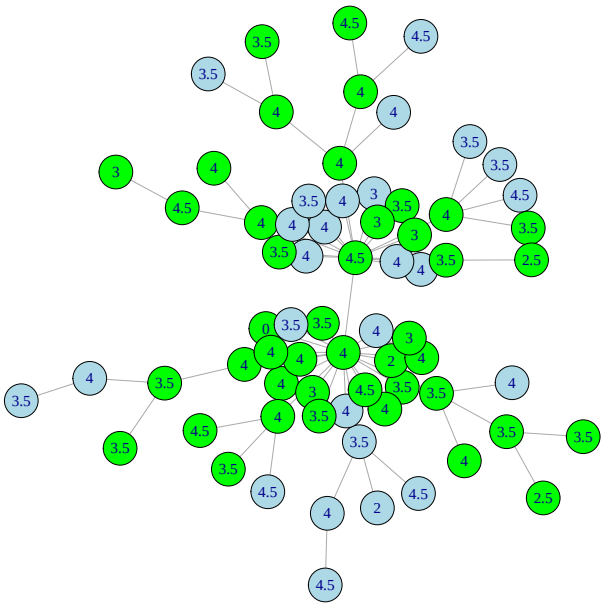


Figure 6.4: Example 4 structure with different classes as colours: book, music, DVD and video as red, yellow, green, and blue respectively, with node labels as ratings.

	Example 1			Example 2			Example 3			Example 4		
	mMSPE	h	q	mMSPE	h	q	mMSPE	h	q	mMSPE	h	q
Original	0.32	0.42	0.9	0.33	3.5	0.9	0.41	0.02	0.2	0.32	9.5	0.9
Original (over h)	0.35	20		0.41	5.02		0.44	4.02		0.32	20	
A	0.36	13.52	0.9	0.40	4.52	0	**0.44**	4.02	0	**0.32**	20	0
QI1	0.28-0.36			0.29-0.40			0.25-0.44			0.29-0.32		
QI2	0.38-0.61			0.48-0.56			0.28-1.05			0.31-0.33		
B	0.35	20	0	0.54	20	0	0.52	20	0	**0.32**	20	0
QI1	0.26-0.36			0.45-0.55			0.42-0.54			0.29-0.33		
QI2	0.39-0.81			0.51-0.62			0.42-1.37			0.31-0.33		
C	**0.33**	5.5	0.9	**0.34**	4.02	0.9	0.53	9.5	0.9	**0.32**	9.5	0.9
QI1	0.27-0.34			0.18-0.37			0.29-0.55			0.29-0.32		
QI2	0.37-0.73			0.31-0.40			0.36-1.78			0.31-0.33		
D	0.35	20	0	0.54	20	0	0.52	20	0	**0.32**	20	0
QI1	0.28-0.36			0.30-0.55			0.30-0.53			0.28-0.33		
QI2	0.40-0.78			0.51-0.63			0.37-1.75			0.31-0.33		

Table 6.5: Results of 4 different permutations A,B,C,D for 4 different graph examples by doing kernel regression over 100 permutations. The bold font values refer to the minimum mMSPE over all permutations for each example. The notations h, q, QI1, and QI2 are as we explain in the text.

Comments on Table 6.5 results

The results in Table 6.5 are produced form four different Amazon graph examples, each example has its own structure and distribution of classes and ratings so, the results will vary between one example and another.

We believe that the node features in most of the real graphs are not randomly distributed, and most of them have some topological structure. However, we notice that for examples 1 and 2, the mMSPE values from the original structures lie in QI1 of permutation D, and for examples 3 and 4, the mMSPE values from the original structures lie in QI1 and QI2 of permutation D. This result means that we can not reject the null hypothesis of the 4^{th} hypothesis, so the node classes and ratings are independently and randomly distributed.

This occurs because we do not have enough data to analyse and reject the null hypotheses as the size of the graph examples is small. So, we could not detect a structure in such graphs. Therefore, we still need to check other hypotheses to investigate any relationship between the node features and the topological structure of the graphs.

In general, we check the MSPE of the naive estimator in each example. We find that all examples have higher MSPE of the naive estimator ($0.35, 0.53, 0.50,$ and 0.32 respectively) compared with our mMSPE when considering the relationship of parameters h and q, ($0.32, 0.33, 0.41,$ and 0.32 respectively).

- In **Example 1**, as we can see from Table 6.1 and Figure 6.1, the majority classes are video and DVD, and most of the nodes have rating score between 3 and 4.5 with very few small and high ratings. We can see in Table 6.5 that the original structure has smaller mMSPE (equal 0.32) than any permutation with q equal 0.9, which is a significant sign of the relationship between nodes' classes and rating in this graph example. This result is supported by the mMSPE value (over h), which reaches $0.35(> 0.32)$, with h equal 20. Moreover, the mMSPE value from the original structure lies in QI1 interval of all permutations but not in QI2, which gives a chance to reject all the null hypotheses and accept that there is a spatial correlation between node features and the graph topological structure. From Figure 6.5, we can see the effect of the q range in the plot of original

structure and the plot of permutation C, both surfaces reach the minimum with q equal 0.9, while in other permutations, the surfaces do not respond to the q range.

- In **Example 2**, the distribution of the classes has some clusters (see book and music nodes in Figure 6.2), but the majority classes are DVD and video with overlap between their ratings as we shown in Table 6.2. From the results in Table 6.5, we could decide to not reject the H_0 of the 3^{rd} hypothesis because the mMSPE of the original structure lies in both QI1 and QI2 of permutation C. So the nodes features are independent from the graph structure. However, we could agree that the node ratings distribution depend on the node classes for two reasons; first, the mMSPE of the original structure lies in QI1 of permutation A but not in QI2. So, we could reject the H_0 of the 1^{st} hypothesis. The second is that both surfaces in Figure 6.6 (the original and C) reach the minimum values with q equal 0.9. Also, we could reject the H_0 of the 2^{nd} as the mMSPE of the original structure does not lie in QI1 and QI2 of permutation B, so the ratings have a spatial correlation over the graph structure.

- In **Example 3**, the results in Table 6.5 show that the mMSPE of the original structure lies in QI1 and QI2 of all permutations except permutation B. So we could not reject all null hypotheses except the H_0 of the 2^{nd} hypothesis which leads us to accept that the ratings have spatial autocorrelation. This result is assured by the prediction error of the original structure which reaches the minimum (equal 0.41) corresponding to h equal 0.02 as evidence of the relationship between the nearby nodes.

- In **Example 4**, the graph has two different classes, and the ratings are mostly between 3 to 4.5 with high overlap between class ratings (see Table 6.4). Therefore, the mMSPE of the original structure is equal to all mMSPE of all permutations and lies in QI1 and QI2 (of all permutations). This result leads us to not reject all the null hypotheses and accept the randomness in the graph nodes. The reason for this is that the graph has just two classes and overlap in the ratings distribution, which makes the analysis more difficult.

To conclude, we can say that there is a relationship between the node classes and rating scores in the Amazon network especially in examples 1 and 2, as there is a difference between mMSPE in the 1^{st} and 2^{nd} rows with high value of h. However, it is hard to check such relation-

ships or permutations with a small graph sizes, so for more investigation, we did a simulation study over tree structure graph with different distributions of classes and ratings.

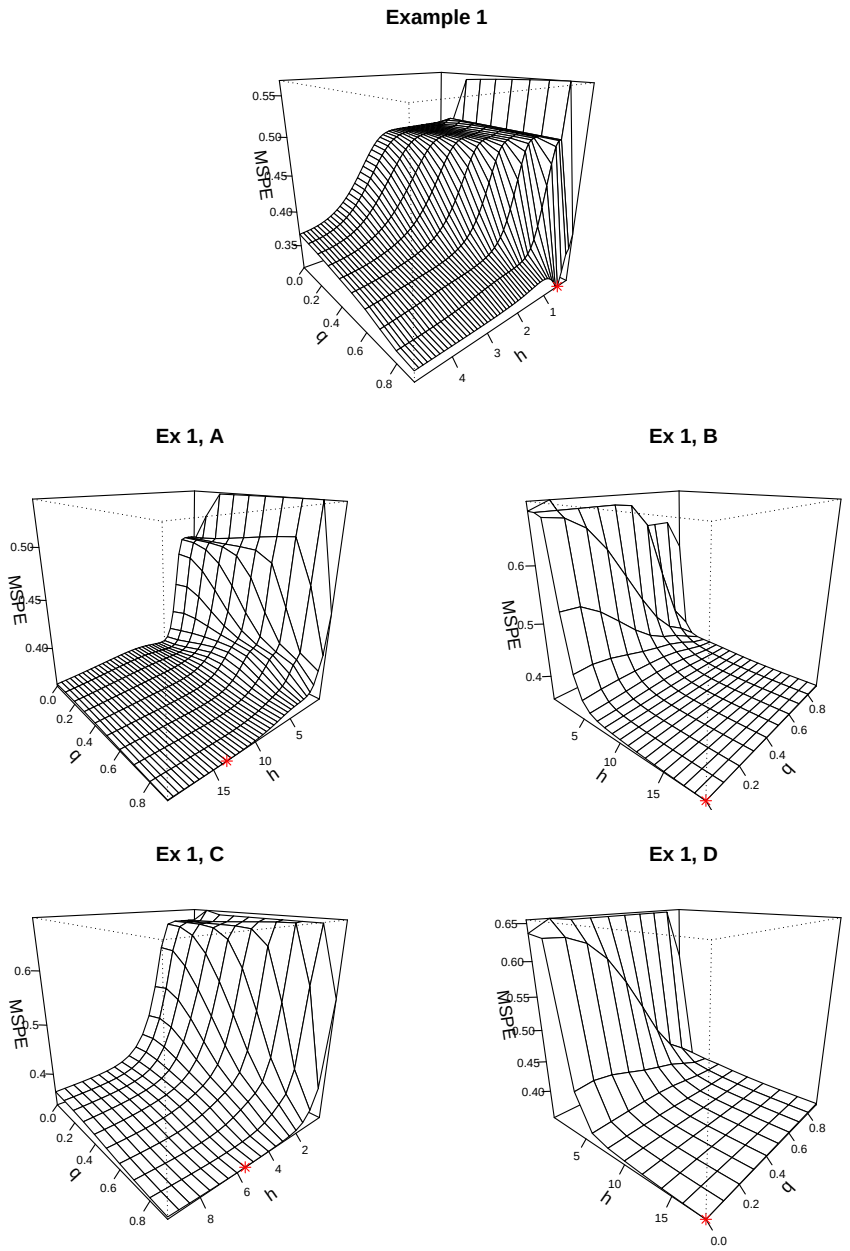


Figure 6.5: 3D curve plot of the MSPE values for the binary values of h and q parameters for the original structure in the top and for each permutation A, B, C, and D in Example 1. The red star point is the minimum point in the curve.

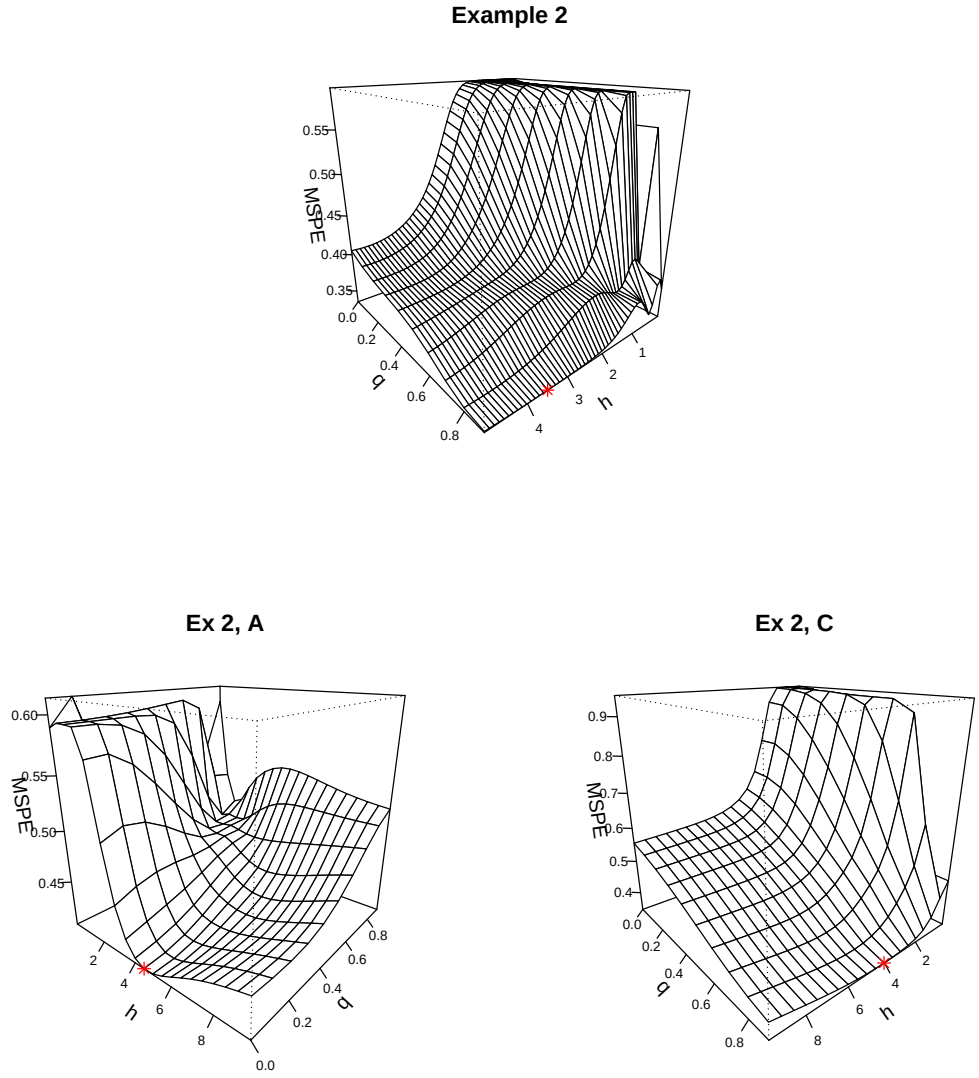


Figure 6.6: 3D curve plot of the MSPE values for the binary values of h and q parameters for the original structure in the top and for each permutation A, and C in Example 2. The red star point is the minimum point in the curve.

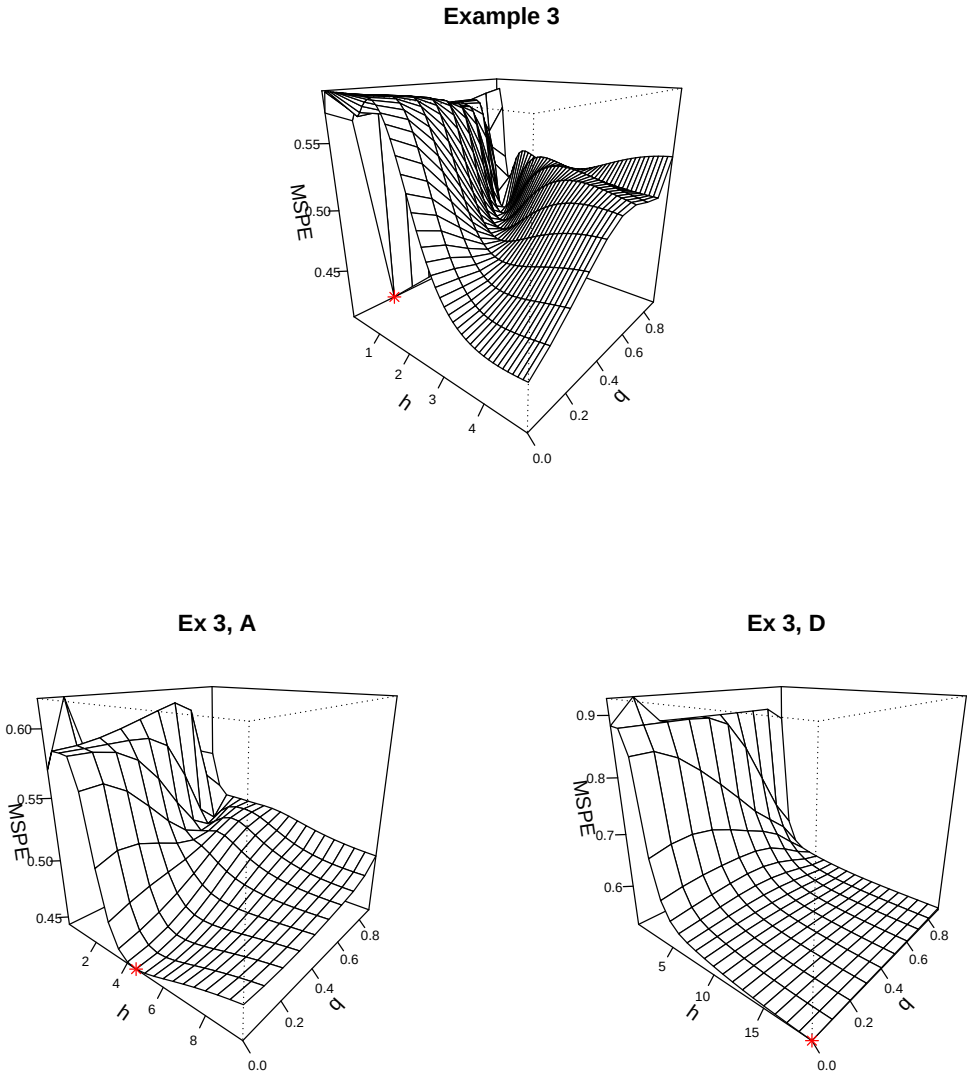


Figure 6.7: 3D curve plot of the MSPE values for the binary values of h and q parameters for the original structure in the top and for each permutation A, and D in Example 3. The red star point is the minimum point in the curve.

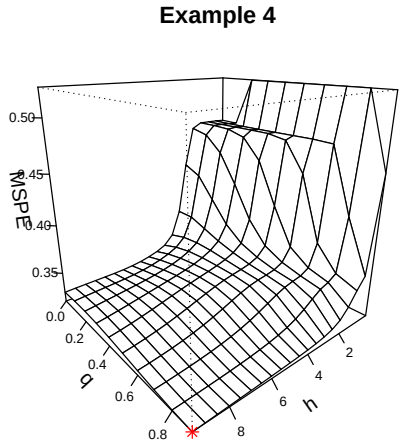


Figure 6.8: 3D curve plot of the MSPE values for the binary values of h and q parameters for the original structure. The red star point is the minimum point in the curve.

6.5 Simulation study

In this section, we check the value of our permutations A, B, C, and D in simulated graphs, and compare between the results of real networks from the Amazon and the simulated graph.

Another aim is to identify the structure and the distribution of the node features of any given graph (similar to our simulated graph) based on the mMSPE results of the original structure and the permutations, by connecting these results with our hypotheses.

We simulate a graph from the BA model which has similar model structure (tree structure) to the Amazon graphs, with 80 nodes. Each node has a class from four different classes and rating score from nine different discrete scores similar to the Amazon network. The experiment could be done with a different number of classes and different rating scores, but we choose the node features similar to the Amazon classes and rating scores to make the comparison easier.

The distribution of the classes and ratings is based on different strategies, the classes and ratings could be independent in some cases and could be completely dependent in other cases. The distribution criteria include three different ways of distributing classes C1, C2, and C3, and seven different ways of distributing rating scores (described below), R1 - R3 depend on the

classes, R4 - R7 independent from classes. The combination between them results in 12 different situations C1R1, C1R2, C1R3, C1R4, C2R1, C2R2, C2R3, C2R4, C3R5, C3R6, C3R7, and C3R4. The description of each case is:

- C1, the classes are equally independently distributed, and each class has 20 nodes.
- C2, the classes are distributed based on four unique clusters.
- C3, the classes are independently distributed with unequal group size.
- R1, each class has a unique rating score for all its nodes from these values $\{1, 2.5, 3.5, 5\}$.
- R2, each class has possible ratings from these groups $\{1, 1.5\}, \{2.5, 3\}, \{3.5, 4\}, \{4.5, 5\}$.
- R3, each class has possible rating scores (with overlap between them) from these groups $\{1, 1.5, 2\}, \{2, 2.5, 3\}, \{3, 3.5, 4\}, \{4, 4.5, 5\}$ for all its nodes.
- R4, the rating scores have a uniform distribution over the graph nodes from the whole rating scores range.
- R5, the rating scores are distributed based on four unique clusters, and each nodes' cluster have one value of rating.
- R6, the rating scores are distributed based on four unique clusters, and each nodes' cluster have group of ratings (similar to R2).
- R7, the rating scores are distributed based on four unique clusters, and each nodes' cluster have group of ratings (similar to R3).

We consider combinations which have different final structures. Other combinations are similar (to some extend) to these chosen combinations. For example, the structure of C1R5, C1R6, C1R7 are similar to the structure of C1R1, C1R2, C1R3 respectively, when we do permutation B. The structure of combinations C2R5, C2R6, and C2R7 have exactly the same structure of C2R1, C2R2, and C2R3. The structure of combinations C3R1, C3R2, and C3R3 are similar to C2R1, C2R2, and C2R3, when we do permutation A.

However, there are lots of choices and possibilities to distribute classes and rating scores which could be done with graphs. In this study, we look at a limited subset of cases which

cover the aims of the experiment. Figures 6.9, 6.10 and 6.11 show examples of these mixed distributions.

The prediction results are in Tables 6.6, 6.7, and 6.8. The corresponding 3D surface plots of some situations are shown in Figures 6.12, 6.13 and 6.14.

The results in these tables are based on one simulated graph from the BA graph model. We tried the same experiment with several different simulated graphs from the same model and they produced similar results.

Comments on Tables 6.6, 6.7, and 6.8 results

In these tables, we arrange the mMSPE results of the rating scores prediction in simulated graphs. The results are over 12 different distributions of classes and rating scores, and over four different permutations as we discussed in Section 6.4. Pair values of parameters h and q are determined in each case, and the QI1/QI2 ranges are also calculated in each permutation. These QI's allow us to check if the original mMSPE lie in these intervals which are used as indicators for rejecting or not rejecting our hypotheses. Another important indicator of the relationship between h and q is the mMSPE of the optimization over h without considering q.

- In **Table 6.6**, the distribution of the classes is independent over the nodes with equal size groups over the four classes. The rating scores distribution is varied starting from R1 to R4, which are shown in Figure 6.9. The effect of this variation appears in the mMSPE results (first row) where it reaches the minimum value in C1R1 with 1.6×10^{-32} and increases to 1.68 in C1R4. On the other hand, the mMSPE values reach high values of q (equal 0.9) in all cases except C1R4, which means that the mMSPE is reached by decreasing the distances between similar class nodes and increasing the distances between different class nodes.

  We can see this result in Figure 6.12 (top) and the top two plots in Figure 6.13, where the q slope of the surface is sharp in C1R1 ($0 <$ mMSPE < 1.5 on q side) and flatter in in C1R2 and C1R3, so C1R1 combination shows higher dependency of the ratings prediction on the node classes than C1R2 and C1R3.

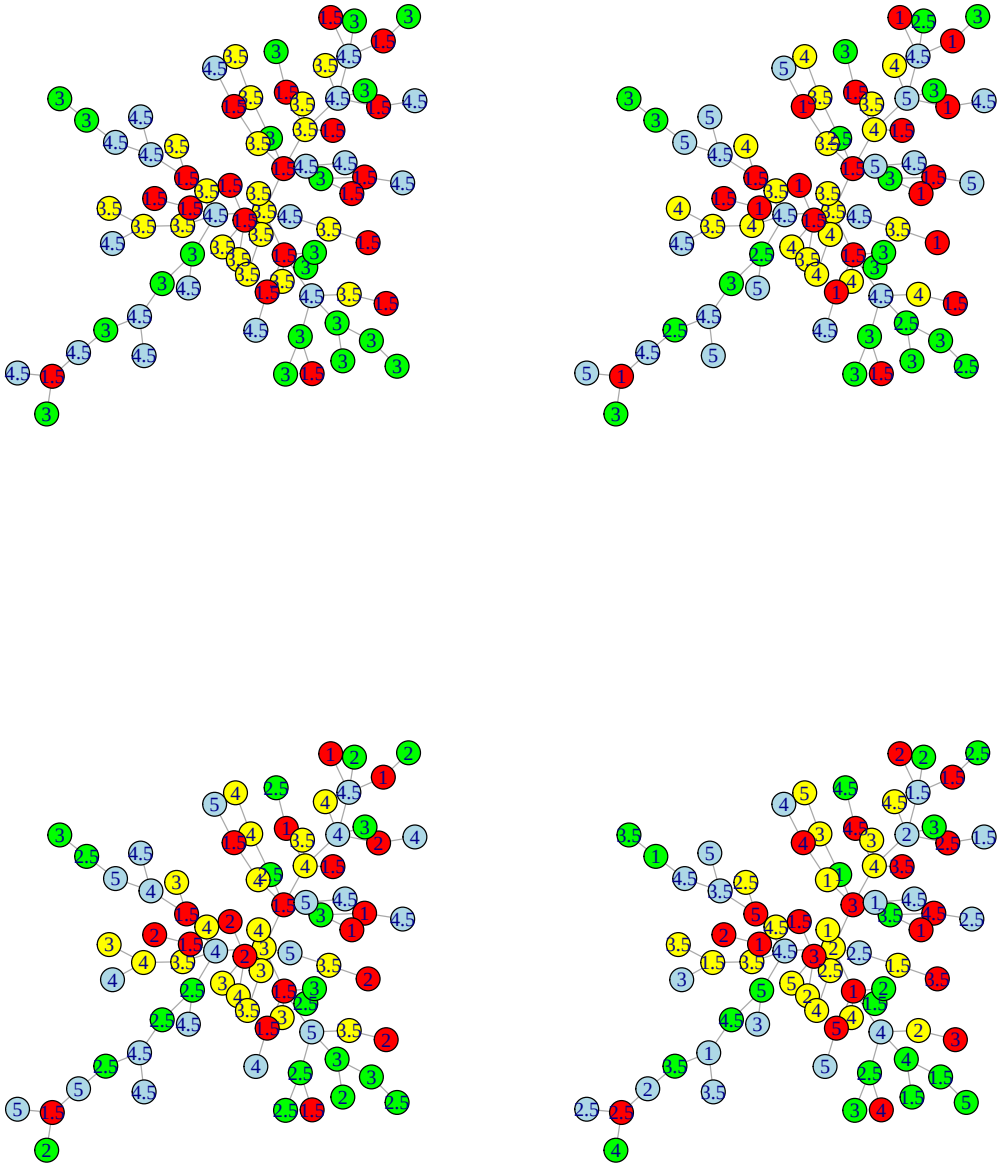


Figure 6.9: Four plots showing four different combinations of classes and ratings distribution, in the top left C1R1, top right C1R2, bottom left C1R3, and bottom right C1R4.

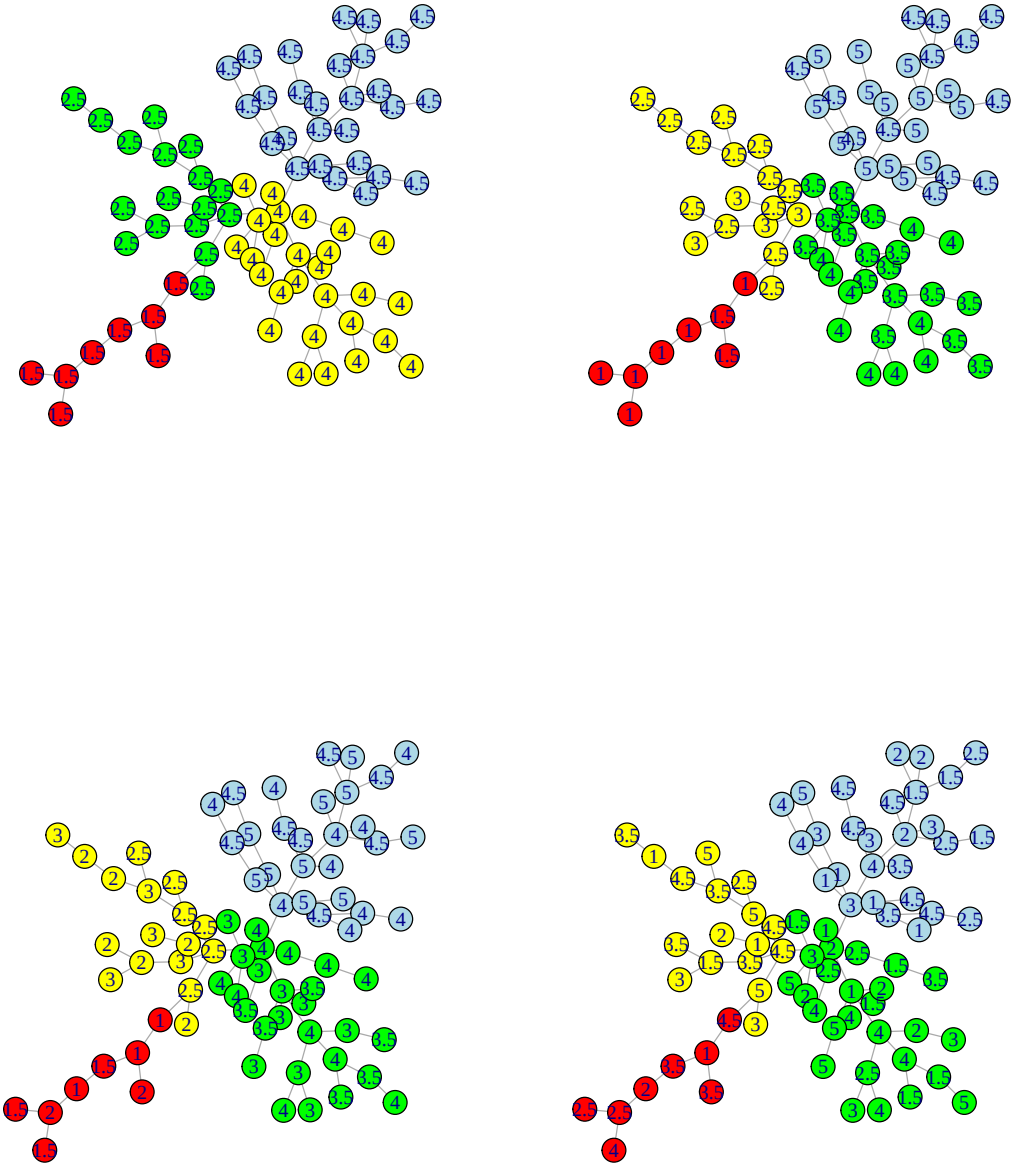


Figure 6.10: Four plots showing four different combinations of classes and ratings distribution, in the top left C2R1, top right C2R2, bottom left C2R3, and bottom right C2R4.

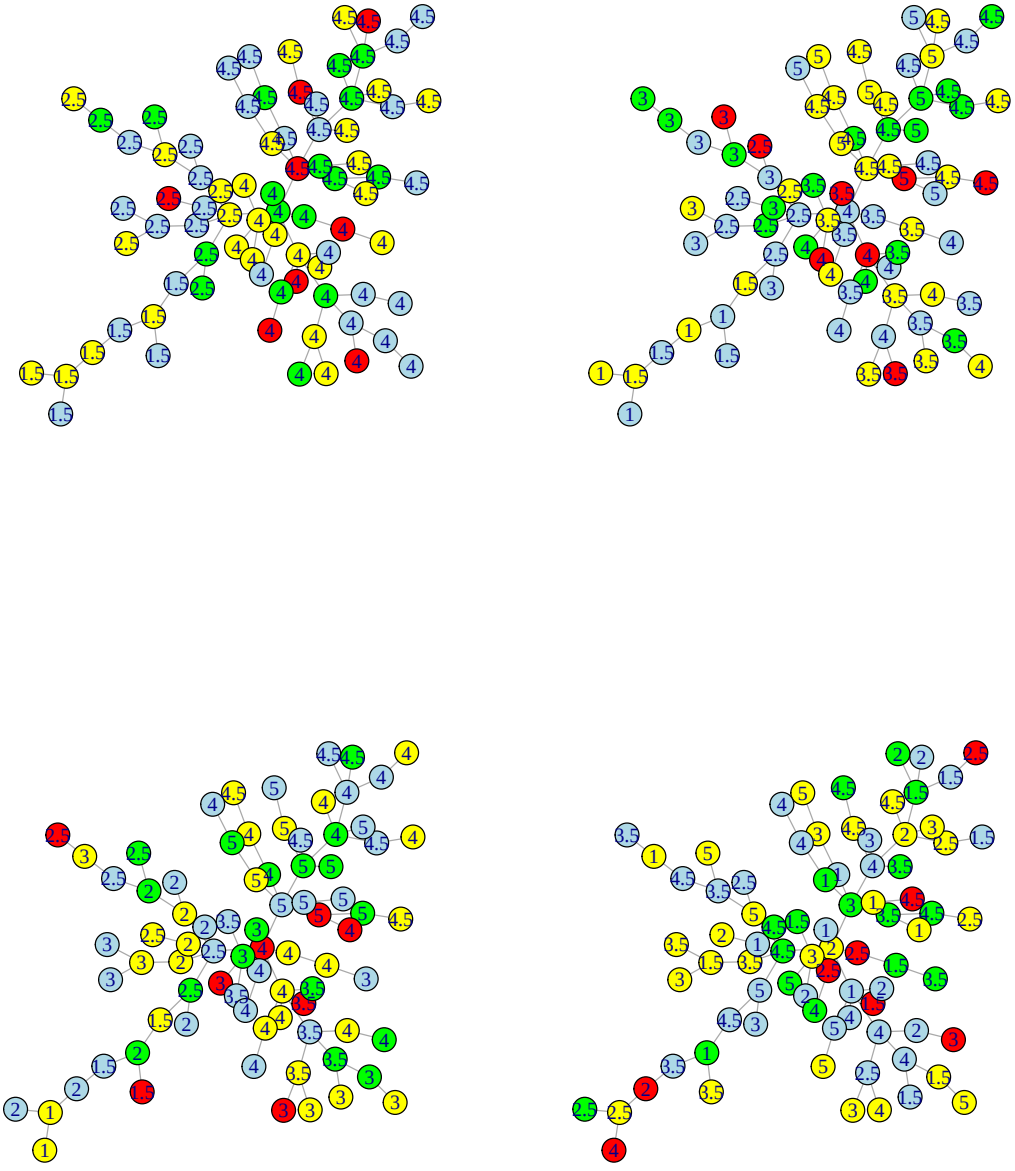


Figure 6.11: Four plots showing four different combinations of classes and ratings distribution, in the top left C3R5, top right C3R6, bottom left C3R7, and bottom right C3R4.

	C1R1			C1R2			C1R3			C1R4		
	mMSPE	h	q	mMSPE	h	q	mMSPE	h	q	mMSPE	h	q
Original	1.6×10^{-32}	0.02	0.8	0.19	5	0.9	0.22	6	0.9	1.68	99	0
Original (over h)	1.27	99		1.36	99		1.30	99		1.68	99	
A	1.19	99	0	1.36	99	0	1.31	99	0	1.68	99	0
QI1	1.16 - 1.19			1.32 - 1.36			1.24 - 1.31			1.61 - 1.68		
QI2	1.9 - 3.8			1.39 - 1.53			1.29 - 1.44			1.68 - 1.68		
B	1.21	99	0	1.37	99	0	1.31	99	0	1.68	99	0
QI1	1.09 - 1.27			1.26 - 1.38			1.18 - 1.33			1.53 - 1.70		
QI2	1.6 - 3.4			1.31 - 1.52			1.25 - 1.44			1.65 - 1.71		
C	$\mathbf{2.4 \times 10^{-32}}$	0.02	0.8	**0.21**	6	0.9	**0.22**	6	0.9	1.68	99	0
QI1	0 - 2.6×10^{-32}			0.13 - 0.23			0.18 - 0.24			1.56 - 1.71		
QI2	(0.14 - 4.1)$\times 10^{-32}$			0.18 - 0.24			0.19 - 0.25			1.63 - 1.72		
D	1.20	99	0	1.37	99	0	1.31	99	0	1.68	99	0
QI1	1.9 - 1.22			1.25 - 1.38			1.22 - 1.33			1.51 - 1.70		
QI2	1.37 - 3.3			1.33 - 1.51			1.27 - 1.43			1.64 - 1.71		

Table 6.6: Summaries of mMSPE, h, and q of prediction results of simulated BA graph with four different combinations C1R1, C1R2, C1R3, and C1R4, and 4 different permutations A,B,C,D, by doing kernel regression over 100 permutations in each setting.

	C2R1			C2R2			C2R3			C2R4		
	mMSPE	h	q	mMSPE	h	q	mMSPE	h	q	mMSPE	h	q
Original	3.4×10^{-32}	0.02	0.9	0.06	2.5	0.9	0.17	7	0.9	1.68	99	0
Original (over h)	0.005	1.02		0.10	4		0.20	6		1.68	99	
A	0.005	1.02	0	**0.10**	4	0	**0.20**	6	0	1.68	99	0
QI1	0 - 0.005			0.07 - 0.10			0.18 - 0.20			1.59 - 1.68		
QI2	0.009 - 0.09			0.30 - 0.37			0.62 - 0.70			1.68 - 1.68		
B	1.06	99	0	1.26	99	0	1.05	99	0	1.68	99	0
QI1	0.99 - 1.08			1.18 - 1.27			1 - 1.06			1.58 - 1.71		
QI2	1.33 - 2.27			1.23 - 1.38			1.01 - 1.12			1.65 - 1.71		
C	$\mathbf{1.2 \times 10^{-32}}$	0.02	0.8	0.12	0.02	0.9	0.24	5	0.9	1.68	99	0
QI1	0 - 0.86×10^{-32}			0.07 - 0.14			0.18- 0.26			1.60 - 1.70		
QI2	$(0.7 - 6.9) \times 10^{-32}$			0.16 - 0.38			0.21 - 0.28			1.63 - 1.71		
D	1.06	99	0	1.26	99	0	1.05	99	0	1.68	99	0
QI1	0.96 - 1.07			1.51 - 1.27			0.96 - 1.07			1.54 - 1.70		
QI2	1.4 - 2.5			1.23 - 1.50			1.01 - 1.14			1.64 - 1.71		

Table 6.7: Summaries of mMSPE, h, q of prediction results of simulated BA graph with four different combinations C2R1, C2R2, C2R3, and C2R4, and 4 different permutations A,B,C,D, by doing kernel regression over 100 permutations in each setting.

	C3R5			C3R6			C3R7			C3R4		
	mMSPE	h	q	mMSPE	h	q	mMSPE	h	q	mMSPE	h	q
Original	0.005	1	0	0.08	4	0	0.18	3.5	0.2	1.68	99	0
Original (over h)	0.005	1		0.08	4		0.18	3.5		1.68	99	
A	**0.005**	1	0	**0.08**	4	0	**0.18**	3.5	0	1.68	99	0
QI1	0 - 0.005			0.07 - 0.08			0.13 - 0.18			1.60 - 1.68		
QI2	0.005 - 0.005			0.08 - 0.09			0.17 - 0.19			1.68 - 1.68		
B	1.06	99	0	1.19	99	0	1.21	99	0	1.68	99	0
QI1	0.93 - 1.07			1.02 - 1.21			1.11 - 1.23			1.54 - 1.68		
QI2	1.36 - 2.13			1.15 - 1.64			1.29 - 1.83			1.62 - 1.71		
C	1.06	99	0	1.19	99	0.8	1.21	99	0	1.68	99	0
QI1	0.95 - 1.07			1.04 - 1.2			1.13 - 1.23			1.59 - 1.70		
QI2	1.3 - 2.23			1.25 - 1.68			1.29 - 1.8			1.64 - 1.71		
D	1.06	99	0	1.19	99	0	1.21	99	0	1.68	99	0
QI1	0.96 - 1.08			1.07 - 1.21			1.11 - 1.23			1.54 - 1.71		
QI2	1.34 - 2.33			1.23 - 1.74			1.37 - 2.04			1.64 - 1.71		

Table 6.8: Summaries of mMSPE, h, q of prediction results of simulated BA graph with four different combinations C3R5, C3R6, C3R7, and C3R4, and 4 different permutations A,B,C,D, by doing kernel regression over 100 permutations in each setting.

One of the most significant results in **Table 6.6** is that permutation C always matches the original results. This is because we permute the node positions attached with their classes and rating scores. We can check this result by looking at the plots in Figure 6.12, which are nearly identical (the same with combinations C1R2 and C1R3). Also, the original mMSPE value for all combinations lies in QI1 and QI2 of permutation C which indicates that there is no significant relationships between the graph nodes (with their features) and their positions in the graph structure. Subsequently, we can not reject the H_0 of the 3^{rd} hypothesis that the graph has independent distribution of its nodes association with classes and ratings.

Another important result is that all other permutations A, B, and D reach high values of mMSPE. This is because these permutations change the relationships between classes and rating scores in all combinations, so the prediction is not affected by parameter q and it always equals 0.

The 2^{nd} row in **Table 6.6** shows the mMSPE results when we ignore the differences between classes by considering $q = 0$ and varying h. We can see that for all combinations except C1R4, the mMSPE values are large compared with the values in the 1^{st} row where parameter q is involved. This leads to the conclusion that parameter q is important in these combinations.

From the results in **Table 6.6**, we can reject the H_0 in the 1^{st} hypothesis and accept that the rating scores depend on the classes as the mMSPE values of the original structure do not lie in the QI1 and QI2 of permutation A in all combinations except C1R4.

We can also reject the H_0 in the 2^{nd} hypothesis and accept that the nodes ratings have spatial autocorrelation, this is because the mMSPE results of the 1^{st} row do not lie in the QI1 and QI2 of permutation B, so when swapping the rating scores among the nodes, the mMSPE is large with a large value of parameter h.

In C1R4, we expect this result of high mMSPE in the original structure and all permutations as the distribution of classes and rating scores are completely independent. So there are no differences between mMSPE in the original structure and any other permutations, so we can not reject any H_0 of our hypotheses in this combination.

- In **Table 6.7**, the distribution of the classes is restricted over four non-overlapping and

non-equal size clusters. The rating scores are distributed according to four strategies: R1, R2, R3, and R4. It is worth mentioning that q is equal to 0.9 in the first combination C2R1 in the original distribution, but this large q has a small effect on the prediction accuracy as we can see from the surface in Figure 6.14 (top) that the surface does not respond with the q parameter range. However, in combinations C2R2 and C2R3, the parameter q has a significant effect on the prediction accuracy error rate, which can be seen clearly from the Figure 6.14 (middle curves).

One of the important results in **Table 6.7** is that the mMSPE values of the original distribution of C2R2 and C2R3 do not lie in the QI1 and QI2 of permutation A. This leads us to reject the H_0 of the 1^{st} hypothesis and accept that the ratings distribution depend on nodes classes in these combinations. Although in combination C2R1, the mMSPE value of the original distribution of lies in the QI1 and not in QI2 of permutation A. We still reject the H_0 of the 1^{st} hypothesis, and accept the dependency of the ratings on the classes. This is because the chosen value h from permutation A is small (equal 1.02), which means the rating prediction depends on the nearby nodes that has smaller error rate, as in R1 the ratings has four separated clusters.

The other important result is the permutation C results. In C2R1, the mMSPE of the original distribution do not lie in QI1. However, it lies in QI2 as we use $q = 0.9$ from the original distribution results, which means decreasing the distances between similar classes and increasing the distances between different classes. So, wherever we permute the nodes, $q = 0.9$ keeps the prediction error low. In this situation, we could reject the H_0 of the 3^{rd} hypothesis and accept that the nodes association with their features have spatial autocorrelation in the graph.

In combinations C2R2 and C2R3, we reject the H_0 of the 3^{rd} hypothesis as the mMSPE of the original distribution do not lie in QI1 and QI2 in permutation C. Figure 6.15 shows the surfaces of this permutation with combinations C2R1, C2R2, and C2R3, which appear the slope of the surface in the q side.

The H_0 of the 2^{nd} hypothesis is rejected in all cases as the mMSPE values of the original distributions do not lie in QI1 and QI2. This means that the ratings have spatial autocorrelation over the graph nodes.

However, we can not reject the null hypothesis of all hypotheses in the combination C2R4, this is because that the mMSPE value of the original structure is large and lies in the QI1 and QI2 of all permutations with a large value of h.

- **Table 6.8** shows different rating scores distributions: R5, R6, R7 with R4. The results show identical mMSPE values between the 1^{st} and 2^{nd} rows with permutation A results, with q equal to 0. This is because randomness in the classes distribution C3 which leads us to not reject the H_0 in the 1^{st} hypothesis for all combinations, and accept that the ratings are independent from classes. However, we may not reject the H_0 in the 2^{nd} hypothesis for the combination C3R4 as the mMSPE of the original structure lies in the QI1 and QI2 of permutation B. While we could reject the same H_0 in the rest of the combinations, and accept that the ratings have a spatial autocorrelation (as we expected from the ratings distributions R5, R6, and R7).

  However, although the mMSPE of the original structure does not lie in the QI1 and QI2 of permutation C (in all combinations except C3R4), we may not reject the H_0 of the 3^{rd} hypothesis because the MSPE of permutation C reaches the minimum with low dependency on h and q ($h = 99$ and low $q = 0$).

Therefore, in the case where we do not have knowledge about the graph structure, these hypotheses could help to discover any relationship between node features and the topological structure of the graph. Also, we can detect any dependency between features, by testing our hypotheses and checking the results of the corresponding permutations.

6.6 Conclusion

We have discussed the idea of predicting the label values of the graph nodes with respect to categorical labels by using nonparametric regression. We applied our methods to real data from the Amazon network and simulated graphs from the BA graph model. The prediction error rate varied between these different data due to the differences in the graph size and the node features structure.

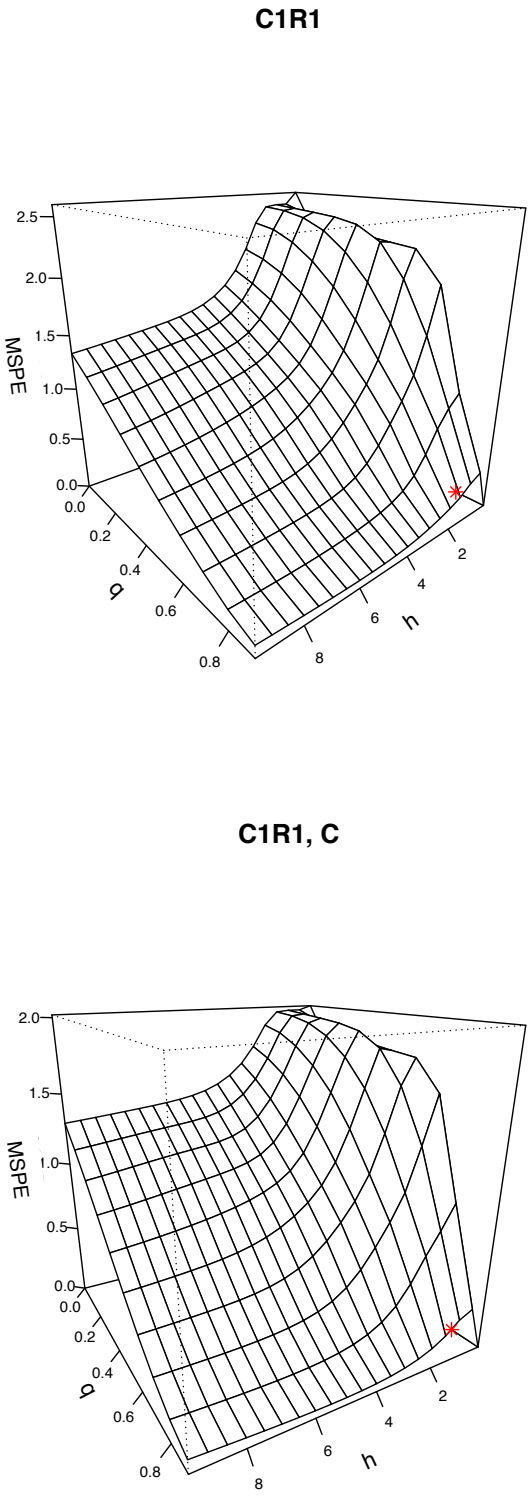


Figure 6.12: Two figures showing the MSPE surface in combination C1R1 and its permutation C, the star red point is the minimum point in the surface.

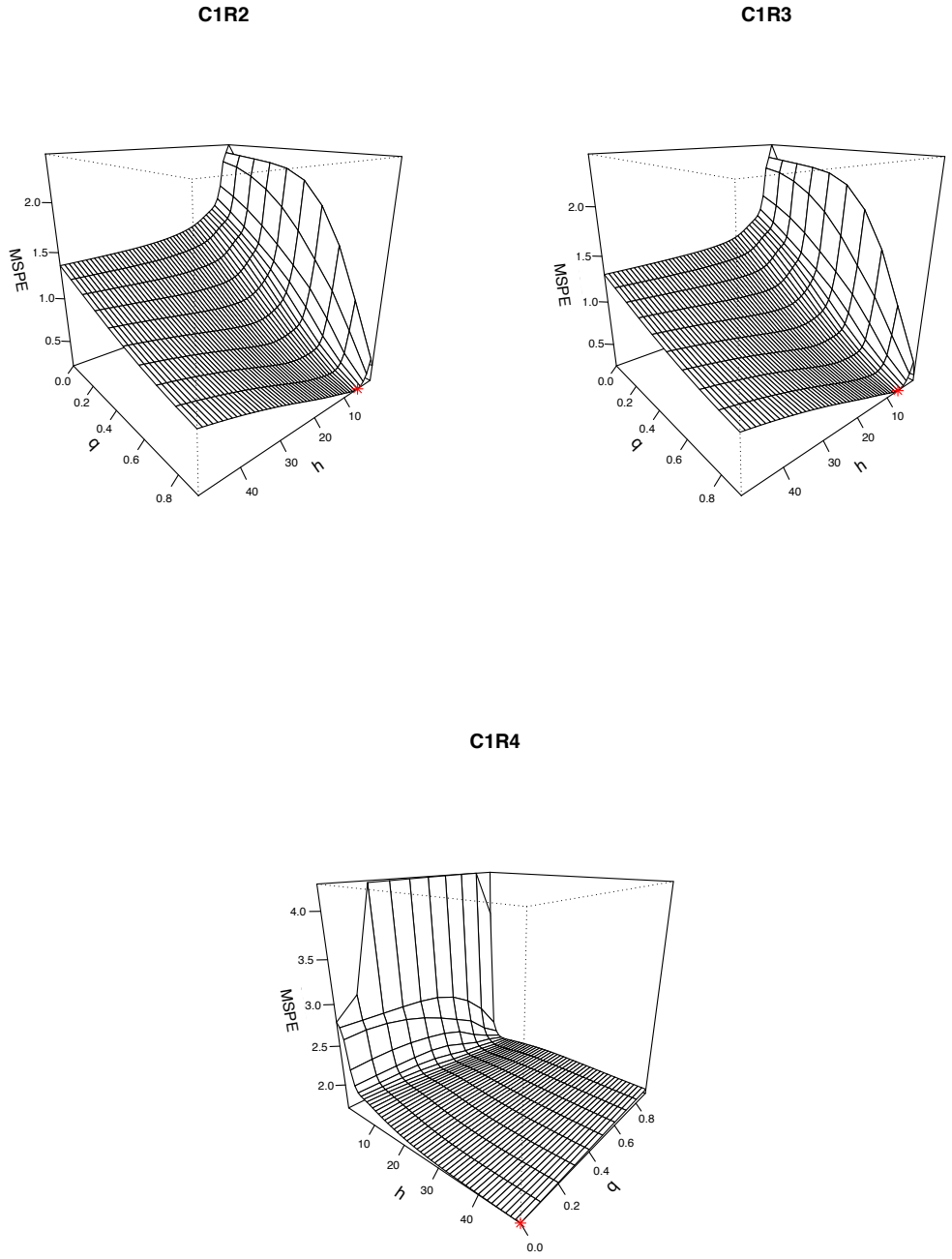


Figure 6.13: Three figures showing the MSPE surface in combination C1R2, C1R3, and C1R4. The star red point is the minimum point in the surface.

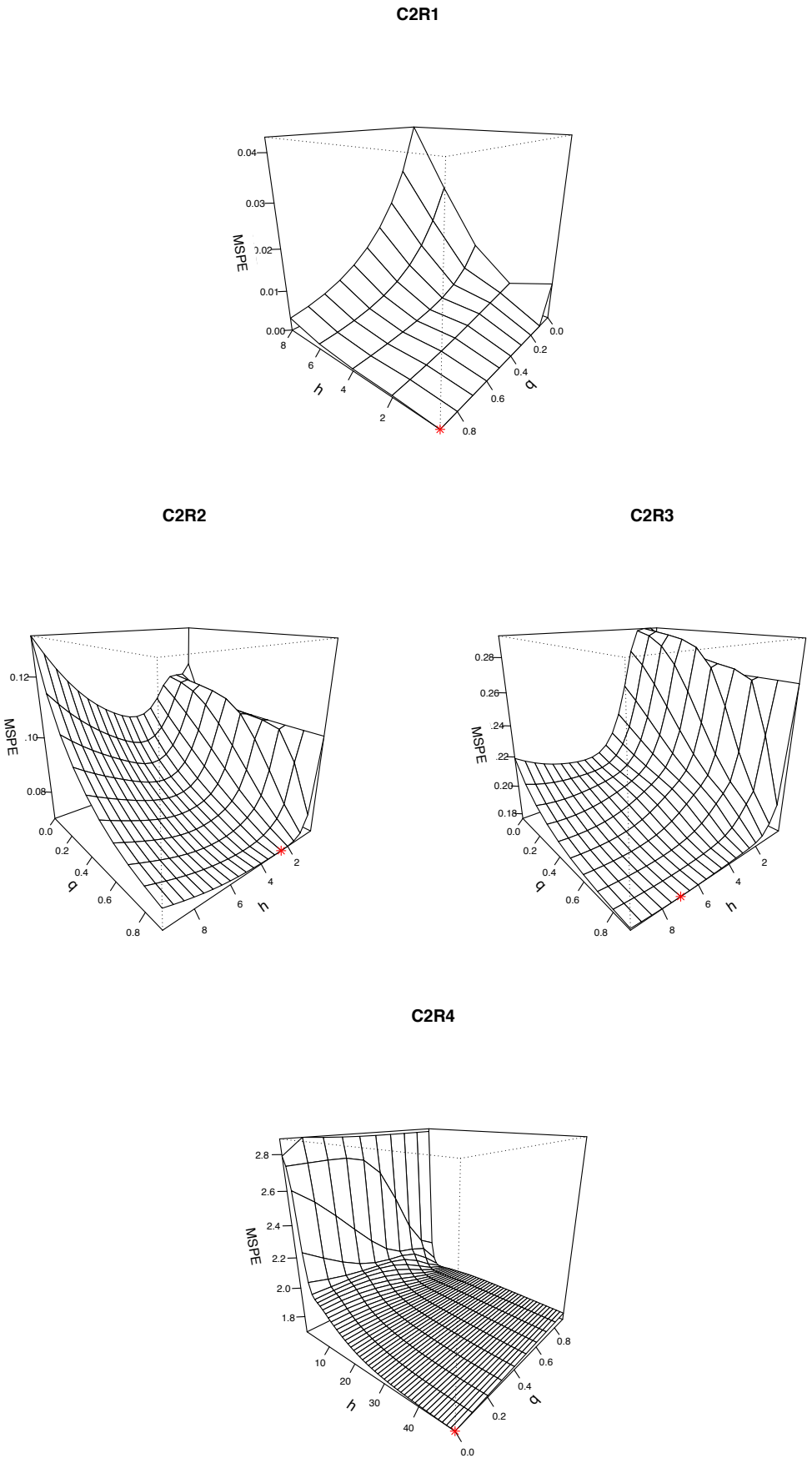


Figure 6.14: Four figures showing the MSPE surface in combination C2R1, C2R2, C2R3, and C2R4. The star red point is the minimum point in the surface.

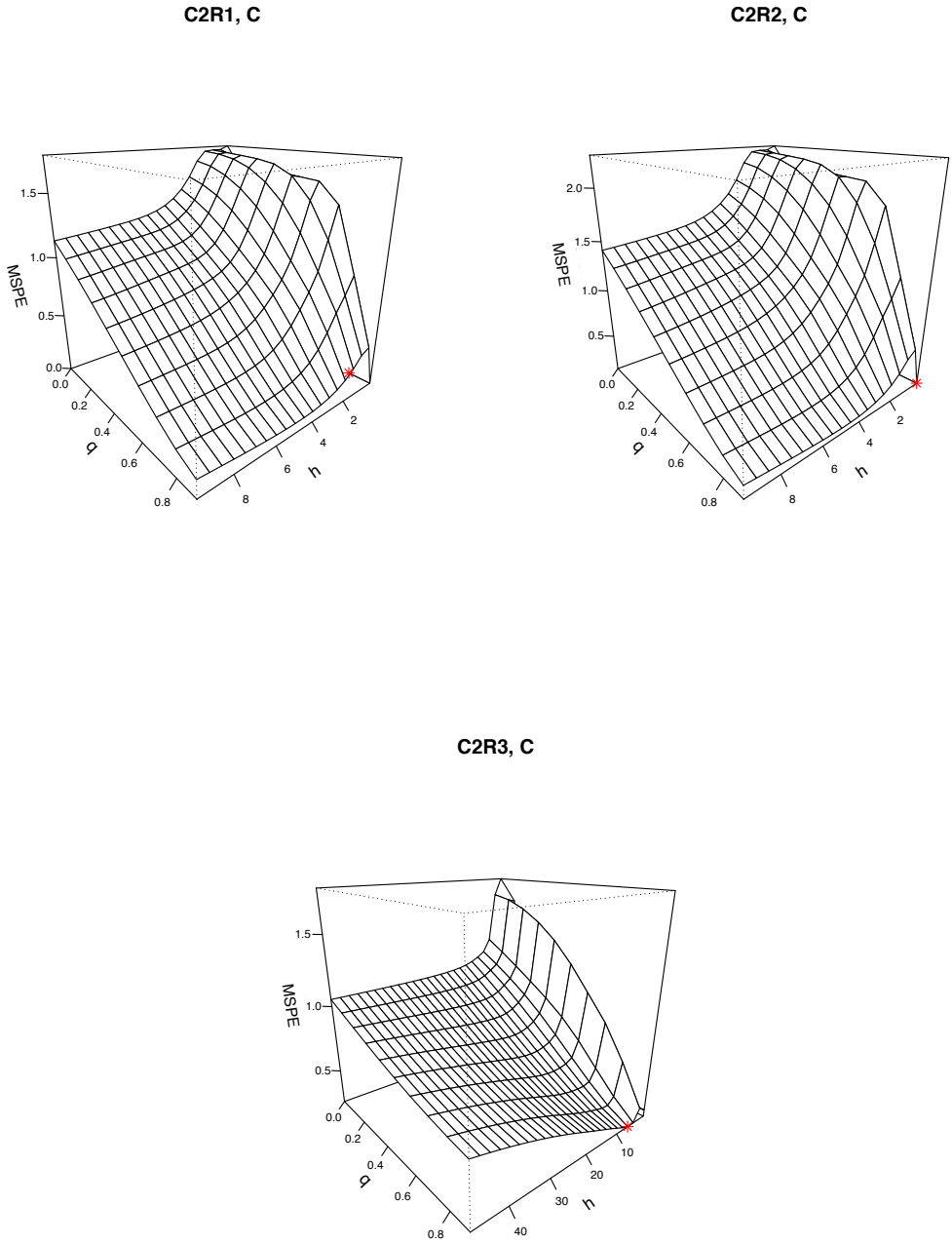


Figure 6.15: Three figures showing the MSPE surface in permutation C in combinations C2R1, C2R2, and C2R3. The star red point is the minimum point in the surface.

One of the well known facts is that the Amazon network and most of the real networks, specially the social networks, have relationships between their nodes. Some of these relations are as simple as the situations we consider in our combinations in the simulation study, and some may be more complicated. For example when the graph nodes have a group of features with a transitive relation between them. However, in dynamic networks, which grow and change with time, these types of relationships could be more complicated.

We also discuss a strategy of learning and predicting the relationships between the node features in a graph and how these features are related and distributed among the nodes. This could be done by comparing the mMSPE results of the original structure and some possible permutations. For example, if the difference between mMSPE with and without q is large with low h value. This could be a sign of a relationship between the feature under prediction and the weighted feature.

Moreover, we consider four different hypotheses to check the behaviour of the predicted feature values. We introduced four different permutations which provide us with more information about the entire graph structure, and can be used to reject or accept these hypotheses. However, the graph size and the features distribution may also play an important role in the accuracy of the prediction. For example, large graph size produces more significant results than small graph size. Also the number of classes and the rating scores range are important parameters that affect the results' quality, so large number of classes leads to lower accuracy rates.

Chapter 7

Conclusion and future work

The first point in this study is about measuring distances between nodes in undirected and unweighted graphs, and how the strength of the relationships between nodes affects the distances between them. To our knowledge, none of the current distance measures, in the literature, is designed to meet our condition, that the equal distances should be distinguished depending on the plurality of the paths between nodes. This philosophical idea is motivated by the existence of ties between pairs that have equal shortest path lengths, and results in a novel formula of measuring distances between nodes. Based on experiments and comparisons, we conclude that the new measure BTD can distinguish between equal path lengths over different graph models. However, as BTD is based essentially on matrix multiplication, this might be difficult with large graphs. In this situation, a large memory software is needed to tackle the big data.

The second point is applying the new distance measure in different techniques in machine learning, and comparing the results with the classical distance measure over different simulated graphs and real networks (the Facebook network and Amazon network). We show an improvement in the clustering results when using our proposed distance measure. We also introduce an adaptation of the K-means algorithm by suggesting two ways of selecting the initial centroids which show their effectiveness in reducing the clustering process time. More experiments may be applied with other cluster methods to check the efficiency of BTD compared with SP in the future.

We also suggest some nonparametric classification techniques to classify the node labels.

We produce a new strategy of approximating the piecewise constant function to choose the smoothing parameter in kernel density classification, which is efficient in choosing the optimal smoothing parameter.

Another new strategy is in choosing the smoothing parameter in the balloon estimator for the adaptive kernel classifier, which depends on the node degree. The results show that this strategy allows the node to control its smoothing parameter based on its relationships with other nodes.

We also apply a transformation for the distances between nodes to a low-dimensional coordinate system by using the multidimensional scaling. This is required to be able to perform the SVM classification technique over graph nodes.

The results here are based on simulated graphs from different models and real networks, which compare the accuracy rates between the selected classifiers. The results could help in choosing the best classifier for specific data in the case when we know the model structure.

The last machine learning technique in this study is prediction, where we aim to predict the node label values by using nonparametric regression. We introduce a modification of the distances which takes into account the similar and different class labels. The experimental results show an improvement in the prediction accuracy with this idea. We also suggest a strategy to discover the graph structure of any given graph by testing proposed hypotheses and permutations. The results in this chapter are based on simulated graphs from the BA graph model and the Amazon network.

The extension of this study could take several directions:

- In this study, we focus on undirected and unweighted graphs. A possible development is to generalize the distance measure formula to be useful in weighted graphs, where the edges have weights which represent the strength of the relationship between nodes. Another generalization may be in directed graphs, where the edges have a direction from the tail node to the head node. Such extension is not straightforward as there are some difficulties. For example, the adjacency matrix of any directed graph is not symmetric except in the case of double edges between nodes, which leads to a non-symmetric dis-

tance matrix. However, more complications in the weighted graphs, where the adjacency matrix could have integer and real values instead of ones. Also, multiple edges and loops which need to be considered in the distance measure.

Some of the interesting directed and weighted real networks are financial networks. The nodes can be considered as bank accounts, and the edges as the money transactions between accounts, where the edge weights represent the transferred money volume between accounts. It is possible to represent this network in an undirected and unweighted graph without losing the edges information by representing this information as node features. For example, the volume of the money in an account by the node degree, the amount of money entering the account in specific time by the in-degree value, and the amount of money leaving the account in specific time by out-degree value. We could also use double edges between any pair of nodes in the directed graph as an additional weight to the distance between the same nodes in the undirected graph.

This idea could help in the analysis of this network, for example, in clustering, it increases the chance to have this pair of nodes in the same cluster. In this situation, the clustering results not only depend on the topological structure of the graph but also depend on the nodes' (accounts') behaviours. Also, by analysing the nodes degree distribution over a period of time, it could help in recognizing the transactions pattern and predicting the future transactions over some accounts (nodes) which may help in detecting anomalies. These anomalies make unusual transactions between them with unusual money volume, which appear as a spike in the transactions behaviour pattern. This can also help in detecting financial fraud, which is one of the interesting problems in financial transaction networks as fraud causes big losses to banks and customers.

- Another possible idea is to consider a dynamic real network which considers time as an important parameter which controls the graph topology, for example, the World Wide Web (WWW), which is one of the most ubiquitous (Costa et al., 2011). It has a huge size with billions of pages from different types, such as text, image, video, or computer program. The dynamic structure of WWW is mostly based on the user behaviours, which depends on the frequently visited pages and the links describe the relations between these pages. However, the distribution of the nodes' in-degree changes with time, as the rate

of visiting new pages could follow the power law distribution, while stable pages have a constant rate of visiting. This could help (for example) with online marketing, where analysing the rate of the "visiting" distribution could help in detecting the top ranking web pages, and targeting these pages in the advertisements. Moreover, the idea of missing information or labels is more complex in WWW network, as is some cases, a link may be broken or a web page is unreachable or removed due to damage or disruption to the server at a specific time.

All these observations make the WWW network more complex and harder to analyse at deeper levels, and this could attract more researchers to test and experiment new tools and investigate more information from this massive network.

- In Chapter 4, we consider the modularity measure as an optimization function of the quality of clustering results. The idea may be extended to create a penalized modularity function, which takes a maximum value corresponding to the best number of clusters.

- Another possible extension is to consider more than two characteristics in the prediction technique. This could be done in some networks which have a wealth of information as in the Amazon network, where the nodes (products) have a multivariate labelling. For example, the number of reviews, number of times downloaded, ratings, number of votes and number of helpful votes. We could investigate the relationships between these features, and find a suitable combination to predict a feature score based on multivariate predictors, by using different weights to scale these predictors. The chosen weights could be based on different strategies; for example, some products have ratings based on one review from a customer, where other products have ratings based on many reviews from different customers. In this case, higher weight may be added to the product ratings which have many reviews. Furthermore, we could predict more information beyond the product characteristics, for example, with books, we may predict the total number of copies that could sell in a period of time based on early sales statistics and other features.

- Some recent research in network analysis is concerned with multilayer networks. There are two common types of multilayer networks: it could be two different layers of networks belonging to two different systems, for example the first layer from Twitter and the second layer from Facebook, with connections between users within both layers and between

layers. Each layer may have similar users (nodes), but the links (edges) between users do not coincide for both layers. This means the set of followers and friends for each user in both layers is not necessary to be identical, which leads to a different adjacency matrix for each layer. One of the interesting ideas is to study the flow of information between layers based on the links between different users or the same user. Another possible idea is to investigate the transitivity between layers. To what extent cycles could appear between layers. The other type of multilayer network is when each layer has a different set of nodes with interlaced connections between them. The complexity in this type is that the behaviour of the nodes could have different patterns within each layer and between layers. The use of multilayer networks could produce promising results in studying complex systems, for example in biology, ecology, and human brain networks.

Appendix A

Single Source Shortest Path Algorithms for Directed Weighted Graphs

In the case of nonnegative directed graphs, Dijkstra's algorithm is one of the most popular methods for finding shortest paths; it was introduced by Dijkstra (1959). The text below describes the strategy of the algorithm.

Let $G(V, E)$ be a graph or a digraph (as defined in Chapter 2) with nonnegative edge weights $E(e)$, and let $Adj[v]$ be the adjacency list of a vertex v. The pseudo code in Algorithm 4 and the example in Figure A.1 present Dijkstra's algorithm.

Dijkstra's algorithm marks the vertices sequentially, It starts by initializing a source node s and discovers all s's neighbours k which are adjacent nodes to s. It visits the neighbour from nodes k (say k_1) which has a minimum distance $d(s, k_1) = w(s, k_1)$ where $w(s, k_1)$ is the weight of the edge $e(s, k_1)$, then it moves from k_1 to one of the k_1's neighbours (adjacent nodes) which has a minimum edge weight. It continues searches and marks next nodes which have minimum edge weights until the destination is reached, and updates the distances from s to each reached node. This procedure is called the minimum priority queue Q, and it allows one to walk from a node to another by passing through the edges with minimum weights. In the pseudo code below, the operation EXTRACT-MIN (Q,w) controls the arrangement of the nodes in Q, and RELAX releases the edge with minimum weight in each iteration. Therefore, it is described as a greedy method that finds the shortest paths faster than other methods. Dijkstra's algorithm is widely

used in transportation networks to determine shortest paths. Dijkstra's algorithm running time is $O(|E| + |V| \log |V|)$ (Cormen et al., 2009; Douglas, 2001).

Algorithm 4. DIJKSTRA (G,w,s) (Cormen et al., 2009)

```
1: INITIALIZED-SINGLE-SOURCE (G, s)
2: Q = G.V
3: while Q ≠ ∅
4:     u=EXTRACT-MIN (Q, w)
5:     for each vertex v ∈ G.Adj[u]
6:         RELAX (u, v, w)
```

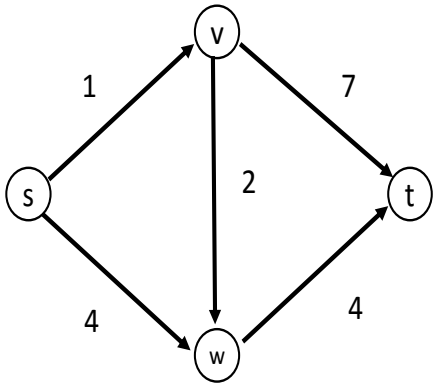


Figure A.1: Directed weighted graph with four vertices and five edges. To calculate the shortest path from the source s to the vertex t with Dijkstra's algorithm, the first step is to discover s's neighbours and visit the neighbour v that has a minimum edge weight from $w(s, v)$. The second step is to move to one of the v's neighbours and visit the vertex which has minimum edge weight from v until the destination t is reached. The $Q = s - v - w - t$, so the distance $d(s, t) = 7$.

In the case of directed weighted graphs with negative edges (which means some edges' weights $E(e)$ are negative), Richard Bellman and Ford introduced the Bellman-Ford algorithm in 1958 (Ford and Lester, 1956; Bellman, 1958). As the algorithm was designed for weighted graphs with negative edges, it can be used to detect negative cycles in graphs. The negative cycle is a cycle that has negative sum of its edges. More precisely, if any negative cycle is detected, the algorithm output is an impossible solution of the shortest paths in this graph.

Given a weighted directed graph $G(V, E)$ with source node s and edge weight function $w : E \rightarrow \mathbb{R}$, the algorithm calculates the distances between s and s's neighbours k to be $d(s, k) = w(s, k)$. Then, it moves from k vertices to k's neighbours (k's neighbours have two edges away from s) after visiting all the k vertices. At each stage, the method updates the distances from s to the reached nodes to be the minimum distances between any pair of vertices.

Figure A.2 shows an example of calculating the minimum-weighted paths with the Bellman-Ford algorithm and describes the algorithm's process. Noticeably, the distance could be infinity if there in no path between a pair of vertices.

The pseudo code in Algorithm 5 shows the Bellman-Ford steps; first, the algorithm initializes the source node s and the distance between each pair of adjacent nodes. Second, the **for** loop traverses $N - 1$ nodes and each iteration enters the **for** loop in lines 3-4 to relax each edge once. By relaxing each edge, the algorithm updates the distances between vertices by testing $v.d > u.d + w(u, v)$, if the condition is satisfied then $v.d = u.d + w(u, v)$ or moving to another edge . After calculating the distances in the graph in $(N-1)^{th}$ iteration and assigning $v.d$ to the shortest path between the source s and any $v \in V$, the algorithm checks for any negative weight cycles in the N^{th} iteration by checking the condition $v.d > u.d + w(u, v)$. If this condition holds for any pair in this iteration then the algorithm returns FALSE and a negative cycle is detected.

The complexity of the Bellman-Ford algorithm is $O(|V||E|)$ because it repeats for every vertex as a source point, which takes $O(|V|)$ time. Furthermore, in each iteration, it passes over all the edges $|E|$ so the loop over all E runs in $O(|E|)$ time. It is slower than Dijkstra's algorithm, which makes using it desirable only in negative weighted graphs (Cormen et al., 2009).

Algorithm 5. BELLMAN-FORD (G,w,s)(Cormen et al., 2009)

```
1: INITIALIZED-SINGLE-SOURCE (G, s)
2: for i = 1 to |G.V| − 1
3:     for each edge (u, v) ∈ G.E
4:         RELAX (u, v, w)
5: for each edge (u, v) ∈ G.E
6:     if v.d > u.d + w(u, v)
7:         RETURN FALSE
8: RETURN TRUE
```

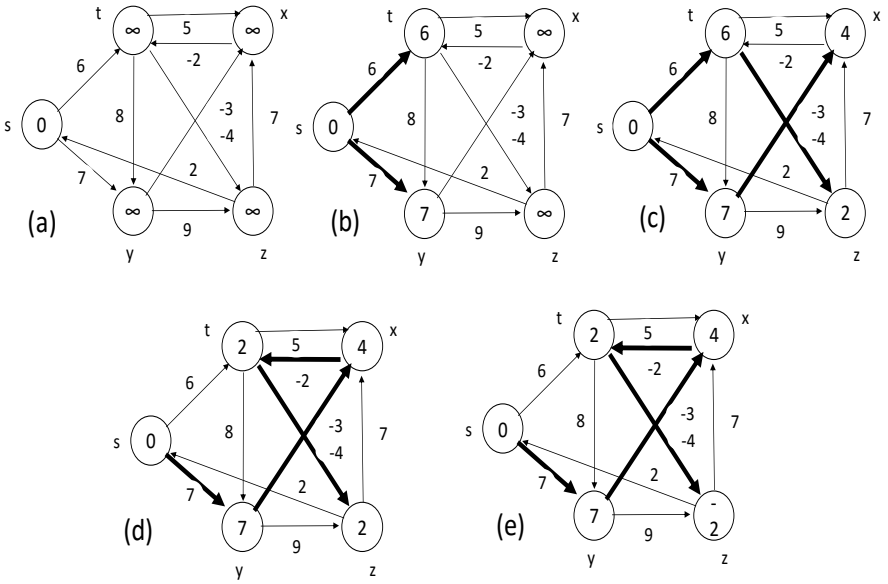


Figure A.2: The execution of the Bellman-Ford algorithm. The vertex source is s. The d values appear between the vertices, and the bold edges indicate predecessors: if edge (u, v) is bold, then $v.\pi = u$. In this example, each pass relaxes the edges in the order $(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$. (a) The graph situation before the first pass over the edges. (b)–(e) The graph situation after each successive pass over the edges. The d and π values in part (e) are the final values (Cormen et al., 2009).

Appendix B

All-Pairs Shortest Path Algorithm for Directed Weighted Graphs

B.1 Solving APSP through SSSP

Solving APSP problems can be done easily by running one of the SSSP algorithms $|V|$ times ($|V|$ is the total number of vertices), i.e. repeating the algorithm for each vertex v as a source. In the case of directed nonnegative weighted graph, Dijkstra's algorithm is the best algorithm as it runs in $|V|O(|V|^2) = O(|V|^3)$ times. Finally, if the graph has negative weight edges, we use the Bellman-Ford algorithm, which has a slower running time equal to $O(|V|^2|E|)$ and in dense graphs it reaches $O(|V|^4)$ due to Max$(|E|) = \binom{|V|}{2} = |V|(|V|-1)/2 = O(|V|^2)$.

Therefore, new algorithms have been introduced, which are faster than repeating the SSSP for all graph nodes. The first of these is the Floyd-Warshall algorithm (Floyd, 1962; Warshall, 1962) followed by Johnson's algorithm (Johnson, 1977).

B.2 Floyd-Warshall algorithm

Floyd (1962) introduced an algorithm that computed the length of the shortest path between all pairs of graph vertices in directed weighted graphs based on Warshall's theorem which com-

putes the transitive closure of a graph, where transitive closure means that there is at least one path between any pair of vertices (v_i, v_j) in the graph. Floyd algorithm is suitable for positive and negative edge weights. But in the case of negative weight cycles, the algorithm returns FALSE in the pseudo code, which means no shortest path results. It runs in $O(|V|^3)$ time as shown in Theorem B.2.1, which is equal to the time complexity of running Dijkstra's algorithm from each vertex v_i (Floyd, 1962; Warshall, 1962).

The algorithm assumes a shortest path p from vertex v_0 to v_j (in the case where v_0 and v_j are not adjacent vertices) in graph G, which passes through all intermediate vertices $p = \langle v_0, v_2, ...v_j \rangle$ rather than v_0 and v_j to end with a minimum-weight path.

The following lemma in Cormen et al. (2009), (p.645) provides the optimal substructure for the shortest path.

Lemma B.2.1 *Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$, let $p = \langle v_0, v_1, ..., v_m \rangle$ be a shortest path from vertex v_0 to vertex v_m and, for any i and j such that $0 \leqslant i \leqslant j \leqslant m$, let $p_{ij} = \langle v_i, v_{i+1}, ..., v_j \rangle$ be the subpath of p from vertex v_i to vertex v_j. Then, p_{ij} is a shortest path from v_i to v_j.*

Based on this property, Floyd defined the quantity $d_{ij}^{(k)}$ as the length of the shortest path p_{ij} between any pair of vertices (v_i, v_j) (k is the number of intermediate vertices in this path) by considering edges' weights that connect the intermediate vertices of the path $\{1, 2, ..., k\}$. If $k = 0$, that means there is no intermediate vertex in p_{ij} and that the vertices v_i and v_j are adjacent. Thus, $d_{ij}^{(k)} = w_{ij}$. If $k \geqslant 1$, he defined $d_{ij}^{(k)}$ by the recurrence relationship as

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if} \quad k = 0, \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if} \quad k \geqslant 1. \end{cases}$$

For APSP in a graph G of size n, the intermediate vertices are a subset of the set $\{1, 2, ..., n\}$ where n in the number of nodes. Therefore, the matrix $D^n = \left(d_{ij}^{(n)}\right)$ gives the length of all shortest paths between each pair (i, j). To compute the values $d_{ij}^{(k-1)}$, Floyd used the above bottom-up procedure with input $W = w_{ij}$ where:

$$w_{ij} = \begin{cases} 0 & \text{if} \quad i = j \\ \text{the weight of directed edge}(i,j) & \text{if} \quad i \neq j \quad \text{and} \quad (i,j) \in E \\ \infty & \text{if} \quad i \neq j \quad \text{and} \quad (i,j) \notin E \end{cases}$$

Here, we present the pseudo code of the Floyd-Warshall algorithm, followed by an example of a directed weighted graph in Figure B.1 with a sequence of matrices $D^{(k)}$ computed by the algorithm.

Algorithm 6. FLOYD-WARSHALL (G,W)(Cormen et al., 2009)

```
1: n = |V|
2: D^(0) = W
3: for k = 1 to n
4:    let D^(k) = (d_ij^(k)) be a new n × n matrix
5:    for i = 1 to n
6:       for j = 1 to n
7:          d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
8: return D^(n)
```

Theorem B.2.1 (Jungnickel, 2013) *The Floyd-Warshall algorithm computes the distance matrix D for (G, w) with complexity $O(|V|^3)$.*

Example B.2.1

$$D^{(0)} = \begin{pmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & 3 & \infty \\ \infty & \infty & 0 & 2 \\ \infty & -1 & \infty & 0 \end{pmatrix} \quad D^{(1)} = \begin{pmatrix} 0 & \infty & -2 & 0 \\ 4 & 0 & 2 & 5 \\ \infty & 1 & 0 & 2 \\ 3 & -1 & 2 & 0 \end{pmatrix} \quad D^{(2)} = \begin{pmatrix} 0 & -1 & -2 & 0 \\ 4 & 0 & 2 & 4 \\ 5 & 1 & 0 & 2 \\ 3 & -1 & 1 & 0 \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & -1 & -2 & 0 \\ 4 & 0 & 2 & 4 \\ 5 & 1 & 0 & 2 \\ 3 & -1 & 1 & 0 \end{pmatrix} \quad D^{(4)} = \begin{pmatrix} 0 & -1 & -2 & 0 \\ 4 & 0 & 2 & 4 \\ 5 & 1 & 0 & 2 \\ 3 & -1 & 1 & 0 \end{pmatrix}$$

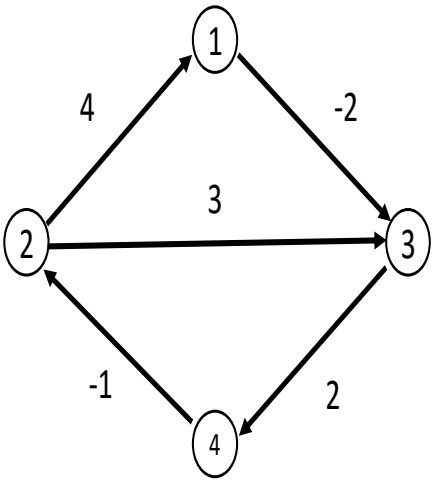


Figure B.1: Example of the Floyd-Warshall algorithm on a directed weighted graph.

The Floyd-Warshall algorithm does not keep any information about the intermediate vertices of any path, so it can not refer to the shortest path route. However, with a simple modification of the algorithm, the shortest path route can be presented, which is considered as an advantage of the Floyd-Warshall algorithm

Moreover, another strong point of this algorithm is, that it can be used as a detector of negative cycles in the graph (similarly to the Bellman-Ford algorithm).

In addition, the Floyd-Warshall algorithm can check the transitive closure of the graph G. This can be checked easily by assigning a weight equal to 1 to all graph edges and then computing the Floyd-Warshall algorithm. If each entry of the matrix D satisfies the condition $d_{ij} < n$, there is at least one path between all vertices.

Since this algorithm appeared in 1962, a series of updates have been introduced to reduce the complexity time: for example, the modifications by Fredman in 1976, and Han and Takaoka in 2012 (Cormen et al., 2009; Madkour et al., 2017).

B.3 Johnson's algorithm

In the case of sparse graphs where $|E| = O(|V|)$, Johnson (1977) produced an algorithm to find the shortest path's length in directed weighted graphs in $O(|V|^2 \log |V| + |V||E|)$ time, which works faster than the Floyd-Warshall algorithm. Johnson's algorithm accepts either positive or negative edge weights (except when there are negative edge weight cycles, which can be reported if detected). The main idea of this technique is to re-weight the negative edge weights to be positive (described below). Then, it uses both Dijkstra's algorithm and the Bellman-Ford algorithm as subroutines. If the graph $G(V, E)$ has nonnegative edge weights (after the re-weighting step), then it runs Dijkstra's algorithm from each vertex to conclude all the shortest paths lengths. Otherwise,if the graph $G(V, E)$ has negative edge weights (after the re-weighting step), then it associates a new weight $\hat{w}$ which satisfies two conditions:

1. For any $v, u \in G$, $p\langle v, u\rangle$ is the shortest path from u to v with the weight function w if and only if $p\langle v, u\rangle$ is the shortest path from u to v with the weight function $\hat{w}$.

2. $\hat{w}$ is nonnegative for any edge in the graph.

The following lemma shows that a certain re-weighting of the negative edge weights in the graph satisfies condition 1. The function δ is the shortest path weights using weight function w and $\hat{\delta}$ is the shortest path weights using weight function $\hat{w}$.

Lemma B.3.1 (Cormen et al., 2009) *Given a weighted, directed graph $G(V, E)$ with weight function $w : E \to \mathbb{R}$, let $h : V \to \mathbb{R}$ be any function mapping vertices to real numbers. For each edge $(u, v) \in E$, define:*

$$\hat{w}(u, v) = w(u, v) + h(u) - h(v) \tag{B.1}$$

Let $p = \langle v_0, v_1, ..., v_k\rangle$ be any path from vertex v_0 to vertex v_k. Then, p is a shortest path from v_0 to v_k with weight function w if and only if it is a shortest path with weight function $\hat{w}$. That is, $w(p) = \delta(v_0, v_k)$ if and only if $\hat{w}(p) = \hat{\delta}(v_0, v_k)$. Furthermore, G has a negative weight cycle using weight function w if and only if G has a negative weight cycle using the weight function $\hat{w}$.

A proof of this lemma is in Cormen et al. (2009).

To check that condition 2 is satisfied, we assume $G(V, E)$ is weighted and directed graph with weight function $w : E \to \mathbb{R}$. Then, construct a new graph $G^{'} = (V^{'}, E^{'})$ by adding a new node s to V to be adjacent to all V nodes, so $E^{'} = E \cup \{(s, v) : v \in V\}$ and $w(s, v) = 0$, $\forall v \in V$. This step is illustrated in Figure B.2. Since $w(s, v) = 0$, $G^{'}$ has no negative weight cycles if and only if G has no negative weight cycles, so by the triangle inequality, it defines $h(v) = \delta(s, v), \forall v \in V^{'}$, and we have $h(v) \leqslant h(u) + w(u, v), \forall u, v$, such that edge$(u, v) \in E^{'}$. Therefore, the new weights $\hat{w}$ in formula B.1 satisfy $\hat{w}(u, v) = w(u, v) + h(u) - h(v) \geqslant 0$ which assures condition 2 for the h function.

The following algorithm shows the re-weighting steps and computing of APSP by Johnson's algorithm followed by an example in Figure B.2, which illustrates the re-weighting idea.

Algorithm 7. JOHNSON (G,w) (Cormen et al., 2009)

Compute G', where $G'.V = G.V \cup \{s\}$
  $G'.E = G.E \cup \{(s, v) : v \in G.V\}$, and
  $w(s, v) = 0$ for all $v \in G.V$
if BELLMAN-FORD $(G', w, s) == FALSE$
  print "the input graph contains a negative-weight cycle"
else for each vertex $v \in G'.V$
    set $h(v)$ to the value of $\delta(s, v)$ computed by the BELLMAN-FORD algorithm
  for each edge $(u, v) \in G'.E$
    $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$
  let $D = (d_{uv})$ be a new $n \times n$ matrix
  for each vertex $u \in G.V$
    run DIJKSTRA$(G, \hat{w}, u)$ to compute $\hat{\delta}(u, v)$ for all $v \in G.V$
    for each vertex $v \in G.V$
      $d_{uv} = \delta(u, v) = \hat{\delta}(u, v) + h(v) - h(u)$
return D

It starts by the first step of Johnson's algorithm which builds $G^{'}$ by adding new vertex s to G and new edges from s to all $v \in V$ with weight equal to 0. Then, it runs the Bellman-Ford algorithm on $G^{'}$ and reports the result. If the algorithm returns FALSE, this means a negative cycle is detected in $G^{'}$. Otherwise, the algorithm continues to the re-weighting step. It computes $h(v)$ as a shortest path $\delta(s, v)$ for all $v \in V'$ by the Bellman-Ford algorithm, followed by computing the new weight $\hat{w}$ for each edge in E'. Finally, it computes the shortest path weight $\hat{\delta}(u, v)$ (with respect to the new weight edges) by Dijkstra's algorithm from each $v \in V$,

then stores the correct shortest path weight $\delta(u, v)$ (with respect to the original weight edges) in matrix D.

In recent years, a series of updates on these essential methods has been produced to minimize the complexity time, making the algorithms faster and more efficient. However, in our research, we focus on undirected unweighted graphs and the distances between vertices in connected components. Thus, any distance between separated components is considered to be infinite.

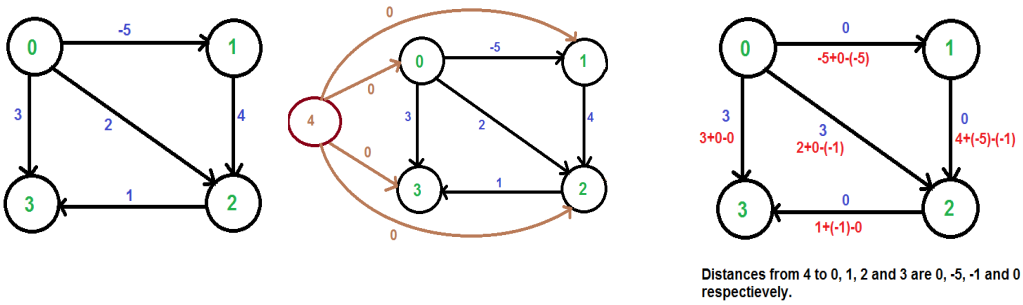


Figure B.2: An example of a directed weighted graph with the calculation of Johnson's algorithm.

Bibliography

Aggarwal, C. C. (2011). *Social network data analytics*. Springer, New York.

Aggarwal, C. C. (2015). *Data classification: algorithms and applications*. CRC Press.

Aggarwal, C. C. and Reddy, C. K. (2014). *DATA CLUSTERING: Algorithms and Applications*. Taylor and Francis Group, LLC, London.

Aggarwal, C. C. and Wang, H. (2010). *Managing and Mining Graph Data*. Springer, first edition.

Aingworth, D., Chekuri, C., and Motwani, R. (1996). Fast estimation of diameter and shortest paths (without matrix multiplication). In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 547–553, Philadelphia, PA, USA. SIAM REVIEW, Society for Industrial and Applied Mathematics.

Albert, R. and Barabási, A. (2002). Statistical mechanics of complex networks. *REVIEWS OF MODERN PHYSICS*, 74(1):47–97.

Almulhim, F. A., Thwaites, P. A., and Taylor, C. C. (2019). A new approach to measuring distances in dense graphs. In Nicosia, G., Pardalos, P., Giuffrida, G., Umeton, R., and Sciacca, V., editors, *Machine Learning, Optimization, and Data Science*, pages 204–216, Cham. Springer International Publishing.

Bansode, S. M. and Bhusare, B. B. (2014). Centroids initialization for k-means clustering using improved pillar algorithm. *International Journal of Advanced Research in Computer Engineering & Technology*, 3(4):1317–1322.

Barabási, A. and Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286(5439):509–512.

Baswade, A. M. and Nalwade, P. S. (2013). Selection of initial centroids for k-means algorithm. *International Journal of Computer Science and Mobile Computing*, 2(7):161–164.

Bellman, R. (1958). On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90.

Blondel, V. D., Gajardo, A., Heymans, M., Senellart, P., and Dooren, P. V. (2004). A measure of similarity between graph vertices: Applications to synonym extraction and web searching. *SIAM REVIEW, Society for Industrial and Applied Mathematics*, 46(4):647–666.

Blondel, V. D., Guillaume, J., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008.

Bonner, R. (1964). On some clustering techniques. *IBM Journal of Research and Development*, 8:22–32.

Borg, I., Groenen, P. J. F., and Mair, P. (2013). *Applied Multidimensional Scaling*. Springer.

Boser, B. E., Guyon, I. M., and Vapnik, V. (1992). A training algorithm for optimal margin classifiers. In Haussler, D., editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA. ACM.

Brodnik, A. and Grgurovič. (2017). Solving all-pairs shortest path by single-source computations: Theory and practice. *Discrete Applied Mathematics*, 231.

Campbell, C. and Ying, Y. (2011). *Learning with support vector machines*, volume 10. Morgan & Claypool, first edition.

Cao, F., Liang, J., and Jiang, G. (2009). An initialization method for the k-means algorithm using neighborhood model. *Computers and Mathematics with Applications*, 58(3):474–483.

Chan, T. M. (2012). All-pairs shortest paths for unweighted undirected graphs in o (mn) time. *ACM Transactions on Algorithms (TALG)*, 8(4):1–17.

Chartrand, G., Lesniak, L., and Zhang, P. (2011). *Graphs and Diagraphs*. Chapman and Hall/CRC, US, fifth edition.

Chaudhari, G., Avadhanula, V., and Sarawagi, S. (2014). A few good predictions: selective node labeling in a social network. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, pages 353–362. Association for Computing Machinery.

Clauset, A., Newman, M. E. J., and Moore, C. (2004). Finding community structure in very large networks. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 70:6.

Cliff, A. D. and Ord, J. K. (1981). *Spatial Processes: models and applications*. Pion Limited, London.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms*. The MIT Press, London, third edition.

Costa, L. F., Oliveira Jr, O. N., Travieso, G., Rodrigues, F. A., Villa Boas, P. R., Antiqueira, L., Viana, M. P., and Correa Rocha, L. E. (2011). Analyzing and modeling real-world phenomena with complex networks: a survey of applications. *Advances in Physics*, 60(3):329–412.

Csárdi, G. and Nepusz, T. (2006). The igraph software package for complex network research. *InterJournal Complex Systems*.

Deng, N., Tian, Y., and Zhang, C. (2013). *Support vector machines: optimization based theory, algorithms, and extensions*, volume 29. CRC Press, Taylor & Francis Group, Boca Raton.

Dienes, P. (1931). *An Introduction to the Theory of Functions of a Complex Variable*. Clarendon Press, Oxford.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271.

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

Douglas, B. W. (2001). *Introduction to Graph Theory*. Prentice-Hall,Inc, US, second edition.

Erdős, P. and Rényi, A. (1959). On random graphs, I. *Publicationes Mathematicae (Debrecen)*, 6:290–297.

Erisoglu, M., Calis, N., and Sakallioglu, S. (2011). A new algorithm for initial cluster centers in k-means algorithm. *Pattern Recognition Letters*, 32(14):1701–1705.

Estrada, E. and Hatano, N. (2008). Communicability in complex networks. *Physical review*, E 77:1–12.

Everitt, B, S., Landau, S., Leese, M., and Stahl, D. (2011). *Cluster Analysis*. Wiley, UK, fifth edition.

Fazekas, I. and Pecsora, S. (2015). A generalization of the barabási-albert random tree? *Annales Mathematicae et Informaticae*, 44:71–85.

Floyd, R. W. (1962). Algorithm 97: Shortest path.

Ford, J. and Lester, R. (1956). Network flow theory. *RAND Corporation*, P-923:1–12.

Ford, L. R. and Fulkerson, D. R. (1956). Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404.

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

Girvan, M. and Newman, M. E. J. (2002). Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the United States of America*, 99(12):7821–7826.

Guillory, A. and Bilmes, J. (2009). Label selection on graphs. In Bengio, Y., Schuurmans, D., Lafferty, J. D., Williams, C. K. I., and Culotta, A., editors, *Advances in Neural Information Processing Systems 22*, pages 691–699.

Hagen, L. and Kahng, A. B. (1992). New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(9):1074–1085.

Han, J. and Kamber, M. (2006). *Data mining: concepts and techniques*. Morgan Kaufmann Publishers, San Francisco, second edition.

Hand, D. J. (1982). *Kernel discriminant analysis*, volume 2. Research Studies, Chichester.

Härdle, W. (1990). *Smoothing Techniques With Implementation in S*. Springer-Verlag, New York.

Haykin, S. (1999). *Neural networks: a comprehensive foundation*. Prentice Hall, Upper Saddle River, NJ, second edition.

Hssina, B., Merbouha, A., Ezzikourl, H., and Erritali, M. (2014). A comparative study of decision tree id3 and c4.5. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 4(2).

Huang, X. and Lai, W. (2006). Clustering graphs for visualization via node similarities. *Journal of Visual Languages and Computing*, 17(3):225–253.

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

Jeh, G. and Widom, J. (2002). Simrank: A measure of structural-context similarity. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 538–543. ACM.

Jiang, M., Chen, Y., and Chen, L. (2015). Link prediction in networks with nodes attributes by similarity propagation. *Computing Research Repository (CoRR)*, abs/1502.04380.

Johnson, D. B. (1977). Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM (JACM)*, 24(1):1–13.

Jungnickel, D. (2013). *Graphs, networks and algorithms*, volume 5. Springer, Heidelberg, fourth edition.

Kannan, R., Vempala, S., and Vetta, A. (2004). On clusterings: Good, bad and spectral. *Journal of the ACM (JACM)*, 51(3):497–515.

Katsavounidis, I., Kuo, C. C. J., and Zhang, Z. (1994). A new initialization technique for generalized Lloyd iteration. *IEEE Signal Processing Letters*, 1(10):144–146.

Kernighan, B. W. and Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307.

Kim, M. and Leskovec, J. (2011).

Kolaczyk, E. and Csárdi, G. (2014). *Statistical Analysis of Network Data with R*. Springer, London.

Kovac, A. and Smith, A. D. A. C. (2011). Nonparametric regression on a graph. *Journal of Computational and Graphical Statistics*, 20(2):432–447.

Kulkarni, S. R., Lugosi, G., and Venkatesh, S. S. (1998). Learning pattern classification-a survey. *IEEE Transactions on Information Theory*, 44(6):2178–2206.

Kurose, J. F. and Ross, K. W. (2017). *Computer networking: a top-down approach*. Pearson, Boston, Mass, 7th edition.

Larose, D. T. and Larose, C. D. (2014). *DISCOVERING KNOWLEDGE IN DATA: An Introduction to Data Mining*. John Wiley and Sons, Inc, second edition.

Leskovec, J., Adamic, L. A., and Huberman, B. A. (2007). The dynamics of viral marketing. *ACM Transactions on the Web*, 1(1).

Leskovec, J., Huttenlocher, D., and Kleinberg, J. (2010). Predicting positive and negative links in online social networks. In *Proceedings of the 19th international conference on world wide web*, pages 641–650. ACM.

Leskovec, J., Kleinberg, J., and Faloutsos, C. (2005). Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on knowledge discovery in data mining*, KDD '05, pages 177–187. ACM.

Leskovec, J. and Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection.

Leskovec, J., Lang, K. J., Dasgupta, A., and Mahoney, M. W. (2008). Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th international conference on world wide web.*, pages 695–704. ACM.

Li, C., Zhang, S., Zhang, H., Pang, L., Lam, K., Hui, C., and Zhang, S. (2012). Using the k-nearest neighbor algorithm for the classification of lymph node metastasis in gastric cancer. *Computational and Mathematical Methods in Medicine*, 2012:876545:1–876545:11.

Liben-Nowell, D. and Kleinberg, J. (2007). The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7):1019–1031.

Lu, L. and Zhou, T. (2011). Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170.

Lucas, E. (1959). Récréations mathématiques.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 2, pages 281–297. University of California Press.

Macskassy, S. A. and Provost, F. (2007). Classification in networked data: A toolkit and a univariate case study. *Journal of Machine Learning Research*, 8:935–983.

Madkour, A., Aref, G., Rehman, F., Abdur Rahman, M., and Basalamah, S. (2017). A survey of shortest-path algorithms. *Computing Research Repository (CoRR)*, abs/1705.02044.

Maimon, O. and Rokach, L. (2010). *Data Mining and Knowledge Discovery Handbook*. Springer, second edition.

McAuley, J. and Leskovec, J. (2014). Discovering social circles in ego networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(1):1–28.

Michie, D., Spiegelhalter, D. J., and Taylor, C. C. (1994). *Machine learning, neural and statistical classification*. Prentice Hall, Englewood Cliffs, N.J.

Moore, E. F. (1959). The shortest path through a maze. *Proceedings of the International Symposium on the Theory of Switching*, pages 285–292.

Moran, P. A. P. (1950). Notes on continuous stochastic phenomena. *Biometrika*, 37(1/2):17–23.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press, Cambridge, MA.

Nadaraya, E. A. (1964). On estimating regression. *Theory of Probability and Its Applications*, 9(1):141–142.

Nannicini, G. and Liberti, L. (2008). Shortest paths on dynamic graphs. *International Transactions in Operational Research*, pages 551–563.

Newman, M. E. J. (2003). The structure and function of complex networks. *SIAM REVIEW, Society for Industrial and Applied Mathematics*, 45(2):167–256.

Newman, M. E. J. (2006). Modularity and community structure in networks. *PNAS*, 103(23):8577–8582.

Newman, M. E. J. (2010). *Networks: an introduction*. Oxford University Press, Oxford.

Nikolić, M. (2012). Measuring similarity of graph nodes by neighbor matching. *Intelligent Data Analysis*, 16(6):865–878.

Ping, S., Liu, D., Yang, B., Zhu, Y., Chen, H., and Wang, Z. (2018). Batch mode active learning for node classification in assortative and disassortative networks. *IEEE Access*, 6:4750–4758.

Price, D. D. S. (1965). Networks of scientific papers. *Science*, 149(3683):510–515.

Rattigan, M. J., Maier, M., and Jensen, D. (2007). Graph clustering with network structure indices. In *Proceedings of the 24th International Conference on Machine Learning*, pages 783–790, New York, NY, USA. ACM.

Rawashdeh, A. and Ralescu, A. L. (2015). Similarity measure for social networks- a brief survey. In *proceedings of modern AI and cognitive science conference (MAICS)*, volume 1353, pages 153–159.

Rencher, A. C. and Christensen, W. F. (2012). *Methods of multivariate analysis*. Wiley, Hoboken, New Jersey, third edition.

Richard, E., Argyriou, A., Evgeniou, T., and Vayatis, N. (2012). A regularization approach for prediction of edges and node features in dynamic graphs. *CoRR*, abs/1203.5438.

Schaeffer, S. E. (2007). Graph clustering. *Computer Science Review*, 1(1):27–64.

Seber, G. A. F. (1984). *Multivariate observations*. Wiley, New York;Chichester.

Seidel, R. (1995). On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403.

Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905.

Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis*. Chapman and Hall Ltd, London.

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

Song, S. and Zhao, J. (2014). Survey of graph clustering algorithms using amazon reviews.

Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, 36(2):111–147.

Suaris, P. R. and Kedem, G. (1988). An algorithm for quadrisection and its application to standard cell placement. *IEEE Transactions on Circuits and Systems*, 35(3):294–303.

Tan, P., Steinbach, M., and Kumar, V. (2006). *Introduction to Data Mining*. Pearson Education, Inc, second edition.

Team, R. C. (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Terrell, G. R. and Scott, D. W. (1992). Variable kernel density estimation. *The Annals of Statistics*, 20(3):1236–1265.

Tou, J. T. and Gonzalez, R. C. (1977). *Pattern recognition principles*, volume 57 of *Appl. Math. Comput.* Addison-Wesley.

Udaya Kumar Reddy, K. R. (2016). A survey of the all-pairs shortest paths problem and its variants in graphs. *Acta Universitatis Sapientiae, Informatica*, 8(1):16–40.

Warnekar, C. S. and Krishna, G. (1979). A heuristic clustering algorithm using union of overlapping pattern-cells. *Pattern Recognition Society*, 11(2):85–93.

Warshall, S. (1962). A theorem on boolean matrices. *Journal of the ACM (JACM)*, 9(1):11–12.

Wasserman, S. and Faust, K. (1994). *Social Network Analysis*. Cambridge University Press, Cambridge.

Watson, G. S. (1964). Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics*, series A 26:359–372.

Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of small-world networks. *Nature*, 393(6684):440–442.

Wu, X., Kumar, V., Q, J. R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G. J., Ng, A., Liu, B., Yu, P. S., Zhou, Z., Steinbach, M., Hand, D. J., and Steinberg, D. (2008). Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37.

Xu, R. (2005). Survey of clustering algorithms. *IEEE Transactions On Neural Networks*, 16(3):645–678.

Yang, Y., Pei, J., and Al-Barakati, A. (2017). Measuring in-network node similarity based on neighborhoods: a unified parametric approach. *Knowledge and Information Systems*, 53(1):43–70.

Zwick, U. (2001). Exact and approximate distances in graphs -a survey. pages 33–48.