Selected Problems in Data Driven and Traffic Related Networks

Author: Ashley James Farrugia

A thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy in Computer Science

Department of Computer Science

University of Liverpool

United Kingdom

November 17, 2016

Abstract

In our research we concentrate on networks. The study of networks have been extensively studied over the last few decades and it is still gaining popularity. In this thesis we study the challenge of gaining an understanding of networks when information about the network is unknown or limited in some way. Initially we consider the challenge of understanding from a vast amount of information what can be used to provide insight into the behaviour of the network, and for this we consider methods and techniques adopted from the social network analysis (SNA) community. Following this, we consider networks that have access to data that is limited in some way and demonstrate that statistical analysis methods can be used to overcome these challenges. Finally, we consider the challenge of having exposure to increasingly less information about the network, and we demonstrate this difficulty by considering a rendezvous problem, a known and establish problem but in this work we consider the search space to be a restricted network, in a distributed environment.

Acknowledgements

I would like to express my deepest gratitude to my supervisors Prof. Leszek Gąsieniec and Dr Russell Martin, whose knowledge, guidance, patience and encouragement has helped me greatly in both my PhD studies and in life. I would also like to acknowledge my examiners Prof. Prudence Wong and Dr Paul Sant for their invaluable input on helping with final preparations of this thesis.

I must acknowledge all of my co-authors for their contributions to the papers that we worked on together. I found these experiences both informative and enjoyable.

I would like to give very special thanks to my friends and family for all of your help and support, both throughout my studies and my entire life. Without their love, encouragement and support I would not have completed this thesis. Finally, in conclusion. I would like to thank the Centre for Global Eco-innovation (CGE) and AIMES Grid Services for providing such an opportunity.

"He who loves practice without theory is like the sailor who boards ship without a rudder and compass and never knows where he may be cast."

– Leonardo da Vinci

Contents

List of Figures

List of Tables

List of Algorithms

Dedicated to my parents, Pauline and Richard, my brothers Riccardo and Kieran, my sister Dionne, and my Uncle Michael and Auntie Julie.

1

Introduction

"We are drowning in information and starving for knowledge."

– *Rutherford D. Rogers*

1.1 Motivation and Problem Scope

The focus of this thesis is *Networks*, a topic that has been mainly in the domain of a branch of discrete mathematics known as *graph theory* [32]. The birth of graph theory was in 1735 when Swiss mathematician Leonhard Euler published a solution to the Königsberg bridge problem, which is described in the literature to be the challenge of finding a round trip that traverses each of the bridges of Königsberg exactly once. Following this discovery, graph theory has developed into a substantial body of knowledge that has provided answers to many practical questions, these include: what is the maximum flow per unit of time from source to sink in a network of pipes, how to colour the regions in a map using a minimum number of colours such that neighbouring regions have different colours, or how to fill n jobs by n people with maximum total utility [32]. The study of networks has generated interest beyond graph theory, and has been extensively studied in the last few decades with applications in Computer Science such as *communication* [119, 121, 126] where [119] and [126] study routing in packet-switched networks, and [121] study distributed networks, *document and resource sharing* such as World Wide Web (WWW) [28] and scientific citation, *knowledge representation* such as sematic networks [152], leading to advanced web technologies such as semantic web [29], data science, visualisation as well as application to other fields and disciplines including Computational Biology such as cataloguing molecules and their interactions of a living cell [22], Life Sciences such as Food Webs or Ecosystems, Physics, Chemistry, Economy, Environmental Studies and many others.

In the literature a network is described to be a ubiquitous object that comprises a set of items (or entities) known as *nodes* (or vertices) that interact with each other based on some common property known as an *edge*. The main attributes of a network include structure and organisation; structure and organisation of a network can be described by identifying inherent communities, statistical properties etc., communication means; how do nodes in the network communicate information with each other, capacity on nodes and edges; understanding the limitations of network entities such as identifying the maximum amount of information being transmitted through edges in the network at any one time, and many others that are described in [130]. These attributes can be dynamic and may vary with respect to time, creating what is known as *temporal networks*, [86], provides a good survey on these types of networks, or they may vary with respect to the user, creating a dynamic network that changes with respect to the intrinsic properties of the user such as height, weight, etc.

In the last decade a substantial movement in network analysis has been witnessed, with the focus shifting from the analysis of small and simple networks that comprise tens or in extreme cases hundreds of vertices towards the study of networks that comprise thousands or even millions of vertices with complex and irregular structures that evolve over time, this type of network is known as a *Complex Network*. This change of scale and complexity has forced a corresponding change in the approach of understanding networks. Previously, for *small* and simple networks it was a relatively trivial task of drawing the network on paper and answering questions about the network structure e.g., which vertex is the most influential., by examining the picture. However, as the scale of networks increased, it quickly becomes a non-trivial task; drawing a picture of large networks becomes an increasingly difficult and impractical task. This prompted research into investigating new methods, statistical methods - Chapter 4 is concerned with understanding network behaviour based on its statistical properties - of developing a similar understanding such as what does the network structure look like, of the larger scale networks [130]. We investigate this type of network in Chapter 3, where we consider a data repository to be a complex network comprising many data assets denoted as nodes in the network.

Most of the work in the field refers to the interplay between the attributes of systems modelled by networks in search for efficient solutions to combinatorial problems. The models for computing solutions can consider distributed or centralised settings, where centralised means solutions are processed based on a global data describing the network state, whilst in distributed models solutions are processed based on local data. In a distributed model each individual node only collects information about its neighbourhood, where its neighbourhood is defined to be the network entities that exist within some radius r of itself, and this information is considered when processing the solution. A common problem in computing distributed solutions is network symmetry, [95] gives intuition on the problem, it is a well known problem that suggests most functions cannot be computed in a distributed fashion using anonymous agents. This impossibility is a consequence of symmetries in the graph (or network). In order to overcome this restriction, symmetry is usually broken through means of randomisation or the use of labels [109].

Research into distributed algorithms in networks has produced many applicable solutions to real-world situations including routing [80, 92, 119, 121, 126], fraud detection [18, 46, 135], detecting terrorism groups and criminals [12, 70], disease detection [35, 146], and many more. For many of these applications domain knowledge is required in order to identify the best approach to the problem, for example, in [12], Alzahrani et al. consider a specific type of network to model the interactions between individuals and their respective affiliations captured in the Noordin terrorist group data. This information was then used to detect communities of terrorists. Alternatively, [92] investigate how the knowledge of network topology can impact the performance of algorithms, and prove that the least amount of knowledge leads to poorly performing algorithms. Similarly, in Chapter 5 we demonstrate the challenge of designing distributed algorithms for anonymous agents in a network where the amount of knowledge of the network becomes increasingly less.

As it is plainly evident from the research mentioned above this area of computing, although studied for many decades, can only continue to grow in the number of applications and its importance in solving the many challenges in modern systems and networks. This is why it is imperative that we grow our understanding of the data that we have available; describing the domain or topology of networks, so that we are able to identify and develop more intelligent and dynamic methods of solving problems in the networks of today. The main goal of this thesis is to develop methods of understanding network data, and demonstrating the need of this understanding in order to improve the speed and accuracy of solutions to problems in networks.

1.2 Contributions

The chapters of this thesis contain work related to the topic of the doctorate, and the material covered in this thesis has provided many contributions to the fields of *Big Data* and *Distributed Computing*, and these are:

1.2.1 Chapter 3

Chapter 3 is based on a workshop paper that was done by the author and co-authors Simon Thompson and Rob Claxton at BT Research, and was presented at Social Network Analysis in Applications (SNAA). This work is in press. The work in this chapter presents a novel application of Social Network Analysis (SNA) techniques in the Big Data domain, and therefore the main contributions are concerned with the Big Data community but it has potential to generate interest in the SNA community. In particular, the study is concerned with data lakes, that is, a single storage capacity for enterprise information that is aggregated from a plurality of applications into a single infrastructure. More specifically, it focusses on generating insights into the organisation of data assets in a data lake, and proposes a solution to the related challenge of providing automation and intelligence for managing and understanding a corpus of data accumulated in a data lake. In order to generate this understanding we use insights and understandings (including algorithms) developed in the SNA community, and propose several algorithms for validating our approach.

1.2.2 Chapter 4

Chapter 4 is based on work done by the author and Leszek Gąsieniec, which was part of an initiative Center for Global Eco-Innovation (CGE) that was designed to develop understanding on vehicular traffic. In this work we have identified a methodology, based on statistical analysis, to deliver insights into the patterns of traffic flow observed in an urban network. The methodology has produced many novel algorithms, most of which can be applied to time series data sets in order to clean, transform and generate understanding of its inherent patterns. Furthermore, this work also proposes a novel approach that can be used to model network traffic, and thus it has applications in a variety of fields and disciplines including Science and Engineering.

1.2.3 Chapter 5

Chapter 5 is based on the work in [10] joint work completed by the author and co-authors, Leszek Gąsieniec, Łukasz Kuszner and Eduardo Pacheco. The work focusses on rendezvous, a problem that is defined to be the rendezvous of two mobile entities A and B in the same time and point in space. In this work we consider the space to be a graph, where the manoeuvrability of the mobile agents (or robots) is restricted by the topological properties of the graph and the intrinsic characteristics of the robots preventing them from visiting certain edges in the graph. Furthermore, we consider three models of computation and study the feasibility of rendezvous, and if rendezvous is achievable we design relevant algorithms that discuss their efficiency.

Everything else is the authors work that was written for this PhD and supervised by Leszek Gąsieniec and Russell Martin.

1.3 Thesis Structure

The first part of the thesis (Chapter 2) will be concerned with providing the tools to obtain a practical understanding of network data. This part should be easily accessible for experts in the field as well as non-experts. The second part of the thesis (chapter 3) investigates the challenge of generating understanding from a vast amount of information what can be used to provide insight into the behaviour of a network. The third part of the thesis (Chapter 4) discusses the challenge of developing understanding of a networks' behaviour when the data available is limited in some way. The fourth part of the thesis (Chapter 5) considers a more theoretical study of network problems, in particular we discuss the challenge of having exposure to increasingly less information; than is considered in Chapters 3 and 4, about the network, and we demonstrate the difficulty of this challenge by considering the rendezvous problem - meeting of two mobile entities in the same point and time in space - in a distributed setting. Finally, (Chapter 6) provides conclusions of the work in each of the chapters, and provides insight into relevant future work.

2

Background

2.1 Statistics

2.1.1 Introduction to statistics

Statistics consists of a body of methods for collecting and analysing data [1], that is, statistics is considered to be a methodology for aggregating, analysing, interpreting and drawing conclusions from information about a specific object or process. It is quite common for any process that involves collecting, processing, interpreting and representing data belongs in the domain of statistics. It is a common misconception that statistics is just about tabulating numbers and providing graphical representations of these. Statistics is considered to be a science of obtaining information about uncertain phenomena or events from numerical and categorical data; considered to be qualitative data resulting from descriptions of objects or process e.g., blood type of a person, marital status, religious affirmation, size of vehicle engine etc. Furthermore, statistics is a very broad subject that has application in a multitude of fields. For example, to study the effectiveness of medical treatments, the reaction to new advertisements, or the attitudes of people within a constituency leading up to voting day. Statistics has been used in practice for many centuries covering agriculture, medicine, politics, economics, technology etc., answering questions about the usefulness, productivity, and quality of services provided.

Two key basic concepts of statistics are *Population* and *Sample*. Population can be characterised a collection of individuals or items in which forms a key component of investigation that an analyst or researcher is interested in for a given problem. In rare circumstances all measurements (or **units**) are obtained for all individuals (or items) in the population, but more often only a subset of individuals (or items) of the population are observed; such set constitutes a sample.

Definition 2.1.1. The population is the collection of all individuals or items under consideration in a statistical study [169]. More formally Johnson et al recognise it in [94] to be a set of measurements corresponding to the entire collection of units for which inferences are to be made.

Definition 2.1.2. The sample is part of the population from which information is obtained [169]. In more descriptive terms Johnson et al consider it to be in [94] a sample from statistical population is the set of measurements that are actually collected in the course of an investigation.

Samples can be used to infer information about the population. A population in statistics can be considered in one of two categories 1) Finite population, and 2) Hypothetical population. Finite population is one that can be listed e.g., quantity of students that are going to University after college, the quantity of products available in stock. In contrast Hypothetical population is an abstract concept that may consider something based on current results e.g., a vehicle manufacturer producing quality vehicles. If the manufacturer considered using the same equipment, materials, and processes on producing future vehicles, then the future vehicles would be considered to be in the hypothetical population. This means that a sample set of vehicles taken from the current production line can be used to make inference about the quality of vehicles that a produced by the manufacturer in the future. There are two main branches of statistics. The branch that involves the study to organise, summarise, display and describe data is known as *descriptive statistics*, and the branch concerned with drawing conclusions (or make inference) about a population based on information provided by a sample taken from that population is called *inferential statistics*. Descriptive statistics includes the construction of graphs, charts and tables, and the calculation of various descriptive measures such as averages, measures of variation, and percentiles [169]. Inferential statistics is concerned with probability theory and methods like point estimation; using sample data to compute a single value which is to serve as a best estimate of an unknown population characteristic, interval estimation; use sample data to calculate an interval of possible values of an unknown population characteristic, and hypothesis testing; hypothesis (basis for reasoning without assumption) that is testable on the basis of observing a process that is modelled using a set of random variables. The two branches of statistics are interrelated, given that it is sometimes necessary to use techniques from descriptive statistics to organise and summarise information from a given sample before inferential statistics can be applied to provide a more thorough analysis of the sample data.

Definition 2.1.3. Descriptive statistics consist of methods for organising and summarising information [169].

Definition 2.1.4. Inferential statistics consist of methods for drawing and measuring the reliability of conclusions about population based on information obtained from a sample of the population. [169].

In statistics it is common for features of the population under investigation e.g., blood type, material properties such as mass, permeability, flexibility, etc., to be summarised by numerical *parameters*, where these parameters are unknown and analysts use sample statistics to make inference about them. In [1] Agresti et al provide a distinction between both parameters and statistics, where a parameter is considered to be an unknown numerical summary of the population, and a statistic is a known numerical summary of the sample which can be used to make inference about parameters. This suggests inference about some unknown parameter is based on a statistic, and that known sample statistics are used in making inferences about an unknown population parameter. In research the main focus is on identifying parameters of the population, and not on the statistics calculated for a given sample set of items. An example, that will give the reader insight into differentiation between statistics and parameters is as follows:

- *Parameter*: The population p of young drivers that are involved in an accident within the first 6 months of obtaining a license.
- *Statistic*: The population $\hat{p}$ of young drivers that are involved in an accident within the first 6 months of obtaining a licence calculated from a sample of young drivers.

Data analysis (statistical processes) should have an objective of making inferences about a population from an analysis of information contained in sample data.

Descriptive and Inferential statistics play a key role in statistical analysis in that they provide a set of tools that allow analysts to organise, summarise and deduce reasoning from results presented in a given experiment in a way that they can be easily interpreted by non-technical users. In our work we use a combination of the methods to generate understanding of vehicular traffic networks based on data presented in the study, and to provide clear, concise, and interpretable conclusions on the phenomena that is vehicular traffic.

2.1.2 Variables

A characteristic, number or quantity that can be measured or counted that varies between objects is called a variable. Variables can be one of two types **Quantitative** or **Qualitative**. Quantitative variables are considered to be variables that yield numerical information (measurements) and examples of these about a human would be height, weight, number of siblings etc. Qualitative variables are considered to be variables that do not yield numerical information, and so provide categorical descriptions of objects, in the case of humans these could be sex, marital status, eye colour etc.

Definition 2.1.5. Quantitative variables can be classified as discrete or continuous. Discrete variables are the results of counting, e.g., quantity of students passing their driving test first time, number of accidents on a specific road in a city centre etc. Continuous variables are variables that can take any value between a set of real numbers, and not a value that is contained within a set of discrete values e.g., the response time to a question, it could be 1.5 seconds, or it could be 1.5111111 seconds.

Definition 2.1.6. Qualitative (Categorical) variables may be further described according to the scale on which they are defined. The scale of the variable gives a certain structure to variable and also implies some meaning of the variable. The category in which a qualitative variable falls may or may not have a logical order or rank. If a variable is considered to have a logical ordering then it is considered to be an *ordinal* variable. Therefore the categories associated with ordinal variables can be ranked higher or lower than one another, but do not necessarily establish a numerical difference between the categories. For example, an ordinal categorical variable would be academic grades (i.e., A, B, C, D, etc.,), clothing size (i.e., small, medium, large, extra-large etc.,) or strength of opinion (i.e., strongly agree, agree, disagree etc). Otherwise, the variable category is considered to be nominal, that is, the categories are merely names, and bare no relevance on the structure of the variable. For example, a nominal categorical variable would be sex, eye colour, religion etc.

Variables are a fundamental concept in statistics that are used to provide a description of a person of object using its attributes e.g., height, weight etc. This attribute can be considered to be a characteristic of the object that may change between objects, and therefore is considered to be a variable. In the context of our work, we consider the object to be a specific thoroughfare (or road) in an urban traffic network, and consider the corresponding attribute (characteristic) to be the traversal time along that road, where the observed values for the variables are considered to be numerical, and therefore this variable type is quantitative.

2.1.3 Organising and describing data

In statistics an analyst will observe the values of variables to which they are interested in e.g., height, weight, eye colour, of a given subject (or object) this is called *data*, each individual component of the data is called an *observation*, and the collection of observed values for one or more subjects (or objects) is considered to be a *data set*. It is common for a categorical data to be code by assigning numerical values to different categories e.g., Sex would be coded by 1 and 2 to denote a person being male or female., but the categorical data remains to be nominal as the values 1 and 2 do not share any properties of numbers that can be used in arithmetic operations. This is an important observation, and a key component of statistical analysis that highlights that one should always check whether the mathematical treatment of statistical data is really legitimate.

Data is typically presented as a matrix, where all the values of a particular variable are organised into a single column; forming a complete column in the matrix, and observations (measurements) collected for a number of variables form the rows in the matrix. For example, consider an experiment in which the analyst is interested in k variables and the experiment is conducted on n subjects, then the matrix M would be denoted

$$M_{kn} = \begin{pmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1k} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2k} \\ & \vdots & & \ddots & \\ x_{n1} & x_{n2} & x_{n3} & \dots & x_{nk} \end{pmatrix} \tag{2.1}$$

where for all $i \in [1, n]$ and $j \in [1, k], x_{ij}$ is the value of the j^{th} variable collected from the i^{th} observation.

The quantity of values that fall into a particular category of a qualitative variable is known as the *frequency* of that category, and a table that comprises all possible classes and their respective frequencies is considered to be the frequency distribution for that particular experiment. Frequencies provide insight into the likelihood that a specific variable is likely to take on a given or fall within a specific category. An interesting characteristic of a given variable is the *relative frequency* for a given category, that is, the percentage of values that are observed to be contained within a specific category, this can be identified using a simple division $\frac{F}{M}$, where F is the frequency and M is the total number of values observed to be contained within the category. A table that comprises a list of all possible categories and their respective relative frequencies is considered to be the *relative frequency distribution.* In the process of understanding data, *relative frequencies* provide fundamental insight into the pattern of the data. It is common for categorical (or qualitative) data to be represented using **pie or bar charts**, where a pie chart is a circle divided into the categories and each portion of the circle is shaped in proportion to the relative frequency of that specific category, and bar charts represents the categories on a horizontal axis, and the frequencies are represented by vertical bars with height proportional to the frequency.

Similarly quantitative variables can also be represented by a frequency distribution. In the case that the discrete variable is limited to a small quantity of distinct values, then the data can be summarised using qualitative method in a frequency table. However, the rows would be the distinct numerical measurement, and the columns would be their respective frequency and cumulative frequencies, where the cumulative frequency is the total frequency of values leading up to and including that particular numerical measurement. In the contrasting case; the variable can take on a multitude of values or is continuous, then the data would have to be grouped using domain knowledge such that categories can be formed, and the frequencies of each category can be computed. This can be achieved by choosing intervals of equal length that cover the range between the minimal and maximal observed values without overlapping, this is known as *class intervals*, and the endpoints of each are called *class limits.* These intervals are used to compute the class relative frequency by counting the quantity of observed values that fall within the class (class frequency), and dividing it by the total number of observations in the data. *As a rule of thumb it is recommended that one should group values into 5 to 15 intervals.* It is common for quantitative data to be represented graphically either as a **histogram**, where a histogram is like a horizontal bar chart. However, the bars touch each other, and it is formed from grouped data, displaying frequencies or relative frequencies of each class interval.

Organising and describing data are two crucial components in statistical analysis as they allow analysts to determine a structure of the data; the structure of the data can help in the process of identifying an appropriate method of handling the data. Thus provide the foundations for analysts to identify key characteristics e.g., patterns, motifs etc., of the objects being analysed.

2.1.4 Frequency Distributions

The distribution of a data set is a table, graph, or formula that provides the values of the observations and how often they occur [169]. That is, a distribution provides insight into the frequency of unique values that a given variable takes on throughout a random experiment. In the previous section distributions were considered to be represented by frequency and relative frequency distributions through bar-charts, pie-charts and histograms. In this section we discuss *frequency distributions* and how they apply to both population and samples from that population, these are known as population distribution and sample distribution respectively. A common analogy for sample distribution is to compare it to a blurry photograph, that is, as the sample size increases, the clarity of the photograph is gradually improved, and the category intervals move closer to the true population relative frequency, until the population distribution is reached and a clear photograph can be recognised. A key characteristic of a distribution for a quantitative variable is its shape, and a common method of identifying the shape of a distribution is to use a smooth curve that approximates the overall shape. An advantage of using smooth curves to identify distribution shapes is that we do not need to worry about minor differences in shape [169]. This approach allows analysts and statisticians to concentrate on the overall patterns of a given distribution, and due to the relatively small number of shapes identified through this method [1], this allows the classification of such distributions. Further techniques that are commonly used to distinguish the shape of distribution are modality (see Definition 2.1.7), symmetry (see Definition 2.1.8), and skewness (see Definition 2.1.9).

Definition 2.1.7. Modality is a simple concept that considered the number of peaks that are observed within a given distribution. A distribution is unimodal if it has one peak; bimodal if it has two peaks; and multimodal if it has three or more peaks [169].

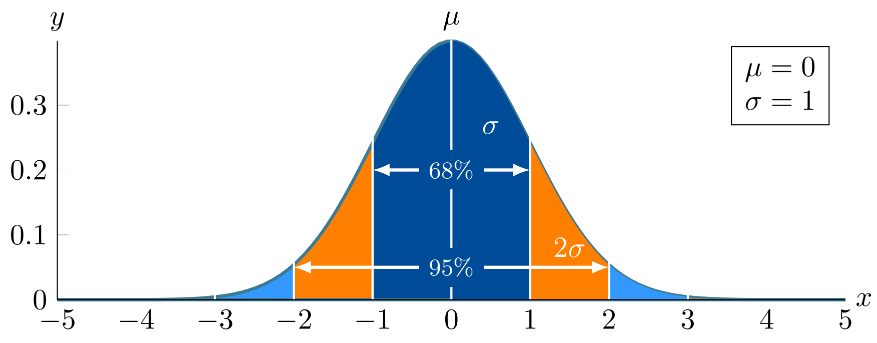
Definition 2.1.8. Symmetry in statistics is a concept that derived from the geometrical meaning of symmetry, where an object is considered to be symmetric if it is invariant to geometrical transformation such as reflection.

A symmetrical distribution is considered to be the natural variation of many variables, and is a distribution of values in which the values occur at regular frequencies, where the majority of values are clustered symmetrically near the mean (discussed in 2.1.5). A symmetrical distribution can be uni-modal, bi-modal, or multi-modal, and when depicted on a graph can represent a single bell-shaped curve, two bell-shaped curves side-by-side, or multiple bell-shaped curves side-by-side respectively. In any case, if a line

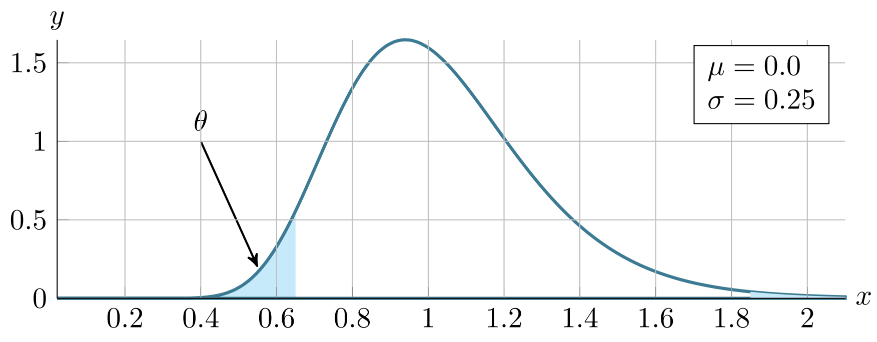
[1]the shape does not have to be exact to be assigned to a specific class, the key is that it is an approximation.

was drawn down the middle of the graph, the two sides will mirror each other. However, it is not always the case that bimodal distribution is symmetrical, and that some are asymmetrical. An asymmetrical distribution is a distribution of values in which values occur at sporadic frequencies, suggesting the distribution of values has some discrepancy. The discrepancy observed is due to the data being skewed, as opposed to being normal. An asymmetrical distribution is a distribution of values in which values occur at sporadic frequencies, suggesting the distribution of values has some discrepancy. The discrepancy observed is due to the data being skewed, as opposed to being normal. Therefore we need to include another measure that can further classify distributions that a non-symmetrical given that this class of distributions can cover a variety of distribution shapes. This measure is known as Skewness, for details see Definition 2.1.9.

Definition 2.1.9. Skewness is a measure that can further classify distributions that a non-symmetrical given that this class of distributions can cover a variety of distribution shapes. The skewness of the distribution can be right-skewed or left-skewed. A key characteristic of skewed data sets is that one tail of the distribution is longer than the other, and generally if a curve representing the distribution has one tail longer than the other, the mean is always towards the longer tail.



(a) Standard Normal Distribution



(b) Log-normal distribution

2.1.5 Measures of center

Measures of center are descriptive measures that indicate the location of a given distributions central point or the most typically observed value for a variable in a set of measurements. It is common for these to be recognised as *averages*. There are three measures of central tendency **mean** (see Definition 2.1.10), **median** (see Definition 2.1.11), and **mode** (see Definition 2.1.12). A key observation is that only the mean and median apply to the quantitative data, however, the mode applies to both quantitative and qualitative data.

Definition 2.1.10. Mean is the most commonly used measure of central tendency in quantitative data, and is frequently referred to as the average. The mean of a data set is the sum of the observations divided by the number of observations [169]. The sample mean is mathematically denoted

$$\bar{x} = \frac{x_1 + x_2 + \cdots + x_n}{N} = \frac{\sum_{i=1}^{n} x_i}{N} = \frac{1}{N}\sum_{i=1}^{n} x_i \tag{2.2}$$

where x_i is the i^{th} observation of a variable v in the data set for a sample size n.

Definition 2.1.11. Median is another commonly used measure of central tendency for quantitative data. The median of a data set is the number that divides the bottom 50% of the data from the top 50% [169]. Thus, the values in the first half are less than or equal to the median, and the values in the second half are greater than or equal to the median value. In order to identify the median value, one must arrange the values into a list of increasing order, and then determine the middle value in the ordered list. If the quantity of values is odd, then the median is exactly the middle value, otherwise the median is the number halfway between the two middle values. In both cases if we denote n to be the number of observations, then the median is defined to be at position $\frac{n+1}{2}$ in the ordered list [169].

Definition 2.1.12. Mode is a measure that can be applied to both quantitative and qualitative data because it has no arithmetical properties other than being the most frequently observed value for a data set. That is, the sample mode is the value of the variable which occurs with the highest frequency in the data set. In the case that no value occurs more than once, then there is no mode identified for the data set.

The mode should always be used as the measure of central tendency when the underlying data set is qualitative. In the case the distribution is symmetrical then the mean is the best measure of center. However, in the case of a asymmetrical distribution, which is skewed to the left or right, the median is generally the best choice of measuring the center. The mean is sensitive to extreme (very large or very small) observations, whereas the median is not [169]. This is related to the fact that the mean can be highly influenced by an observation that falls far from the rest of the data, this is known as an **outlier**. A resistant measure of central tendency is one that is not sensitive to outlying values. The median is considered to be a resistant measure, due to the 50% breakdown value property; that is 50% of the values have to be changed to infinity before the value of the median changes. However, the mean is not a resistant measure, as the breakdown value for this is 1, that is, only one value can change the result. Finally it can be assumed that the sample mode, mean and median of a variable have corresponding population measures of central tendency. Given this assumption, we can use the sample mean, median or mode to provide an estimate of the values for the corresponding unknown population values.

2.1.6 Measures of variation [17, 94, 169]

An important aspect of descriptive analysis for a given data set is measuring the variability. Identifying the measure of center is not a granular enough method for detecting properties that can easily separate distinguish a distribution from another. A pair of data sets can have exactly the same measure of center e.g., Mean, for their respective distributions but can be completely different in shape. Therefore a measure of variability is introduced to associate to each data set a set of numbers that quantitatively measures how the data is scattered away from its center, or clustered towards it. There are three main measures of variation 1) *Range* (see Definition 2.1.13), 2) Interquartile Range (IQR) (see Definition 2.1.14) and 3) Variance and Standard Deviation (see Definition 2.1.15).

Definition 2.1.13. The range is the simplest measure of variation, and it is obtained by computing the difference between the largest observed value of a given variable and the smallest one for a random experiment. The range of a data set is the number R defined by the formula

$$R = x_{max} - x_{min} \tag{2.3}$$

where x_{max} and x_{min} are the maximum and minimum values respectively. The range is a measure of variability because it denotes the size of the interval over which the data is distributed. However, in using the range a great deal of information about the data is ignored, this is because only the largest and smallest values of the data are considered, and the rest is disregarded.

Definition 2.1.14. The Interquartile range (IQR) is a measure of variation using quartiles for a given data set. A quartile of the data set is obtained by numerically ordering the data set s.t., $x_1 \leq x_2 \leq \cdots \leq x_n$, and dividing the observed values into quarters (or 4 equal parts. A given data set has 3 quartiles Q_1, Q_2 and Q_3, where the first quartile, Q_1 divides the bottom 25% of the values from the top 75%, the second quartile Q_2 is the median, which is the quantity that divides the bottom 50% of values by the top 50%, and the third quartile Q_3, is the quantity that divides the bottom 75% of values by the top 25%. The quartiles are mathematically defined:

1. Q_1 is defined to be a position $\frac{n+1}{4}$
2. Q_2 is defined to be a position $\frac{n+1}{2}$
3. Q_3 is defined to be a position $\frac{3(n+1)}{4}$

The IQR is generally the preferred measure of variation when the median is used as the measure of central tendency (typically in cases when the distribution is asymmetric or skewed). The IQR is the difference between the first and third quartiles for a given data set and is denoted,

$$IQR = \left(\frac{3(n+1)}{4} - \frac{n+1}{4} \right) = (Q_3 - Q1) \tag{2.4}$$

The IQR of a data set can be observed to be the range of the middle 50% of the observed values, that is, it represents the length of the interval covered by the center half of the observations values for a data set. This measure is commonly used in outlier detection due to the fact that the measure of variability is not disturbed if a small quantity of values are outliers (extremely low or high).

Definition 2.1.15. The variance and standard deviation are two of the most common methods of measuring variability, and are considered to be more elaborate that the range by being highly dependent on the properties of the data i.e., sample or population data. In the case that the data set comprises only a sample of values we compute the sample variance. The sample variance is a positive number that represents the average of the squared deviations from the mean for the variable in question. For a variable x, the sample variance is denoted by s_x^2 is,

$$s_x^2 = \frac{\sum (x - \bar{x})^2}{n-1} \tag{2.5}$$

and the respective sample standard deviation is the square root of the sample variance denoted $\sqrt{s_x^2}$, or s_x. The standard deviation is a measure to quantify the measure of dispersion of a set of observed values. In particular, it determines how spread out the data points are by quantifying the amount a value deviates from the mean value. A key observation for the standard deviation is that the more variations that is observed in the distribution of values, the larger this quantity. For example a standard deviation of 0 suggests that the data points tend to be close to the mean (expected value) of the data, whilst a high standard deviation indicates that the data points are widely spread out. This suggests the standard deviation satisfies the criteria of measures of variability. However, the standard deviation can heavily be influenced by outliers, given that it uses the mean, and the mean has a breakdown value of 1. That is, only one value in a data set has to be set to infinity for the mean to change its value to infinity. In the complementary case that the data set comprises the whole population of data, then the variance and respective standard deviation are denoted,

$$\sigma^2 = \frac{\sum(x-\bar{x})^2}{n}, \text{ and } \sigma = \sqrt{\frac{\sum(x-\bar{x})^2}{n}} \tag{2.6}$$

where the denominator is the full size of the data, and not the reduced one as is the case for the sample variance and standard deviation. The reason for this reduced number of values is based on the *Bessel's correction* to produce an unbiased estimator of the population variance and respective standard deviation.

2.2 Networks

Networks are universal objects that arise in a variety of fields and disciplines as disparate as *Sociology*, *Physics*, and *Biology*. A network is an interacting system that comprises entities (referred to as nodes) that interact with each other based on some common property to form edges. Systems generally take the form of networks in a variety of fields, examples of this are World Wide Web (WWW), staff rostering, transportation etc. The study of networks or Network Theory dates back to 1735 when Euler produced a solution of the

Königsberg bridge problem, work that is often cited as the first proof of the theory in networks, and during this period Graph Theory (see Section 2.2.1 for details) was developed.

2.2.1 Introduction to Graph Theory [130, 153]

The study of networks dates back to the 17th century when famous mathematician Leonard Euler became interested in a problem *Königsberg Bridge Problem.* The city of Königsberg was built on the banks of the river Pregel and on two islands that lie in midstream, and seven bridges connected the land masses. According to the literature a common brain teaser at this time was to construct a single path that crosses all seven bridges exactly once. In 1736 Euler proved the impossibility of such a path existing using graphs - a mathematical object comprising points, also known as vertices or nodes, and lines, referred to as edges or links. A graph is defined in Definition 2.2.1. It is a mechanism that enables the abstraction of all unnecessary details from the original problem such as land mass sizes but retains the connectivity between points. Euler modelled the Königsberg problem using a graph that comprised four vertices - each representing the four land masses - and seven edges - each joining the land masses - following in the pattern of the Königsberg bridges represented in Figure 2.1. This allowed Euler to consider the Königsberg problem as a mathematical problem that involved finding the Eulerian path or *tour* (see Table 2.1) on the graph - precisely the closed path that traverses each edge exactly once. Euler proved that there does not exist such a path, observing that, since any path must enter and leave every vertex it passes through, except the first and last, there can be at most two vertices in the graph with an odd number of incident edges. In graph theory the property that describes a vertex by the number of edges that are attached is known as degree. Therefore there are at most two vertices in the network that have odd degree. Since all four vertices in the Königsberg graph have odd degree, the bridge problem has no solution. Eulers' proof is considered to be the first theorem in a now highly developed field of discrete mathematics known as graph theory, which has developed to become the principle language for describing the properties of networks [82, 168]. A key benefit of using graph theory is that one can abstract away details of problems to describe the topological features with clarity that would otherwise be impossible if the details were retained. As consequence graph theory has spread well beyond pure mathematics to applications in engineering [4], operations research [127] computer science [114] and sociology - ethnographic studies [57, 148, 168].

Definition 2.2.1. A *graph* G consists of a collection of *vertices* (see Table 2.1) denoted by $G(V)$ and a collection of *edges* (see Table 2.1) denoted by $G(E)$, for which we write $G = (V, E)$. Each edge e $\in E$ is said to join two vertices, which are called its *end points.* If e joins $u, v \in V$, we denote this relationship by $e = \langle u, v \rangle$, and we say that u and v are adjacent to each other or we say e is incident to vertices u and v. A common method of representing graphs is via an *adjacency matrix.* A single cell in the adjacency matrix A is $A[i, j]$ which denotes the the number of edges joining vertices v_i and v_j, and the sum of values for a specific row i is equal to the degree of vertex v_i.

A graph that does not have any *loops* or *multiple edges* - a graph that does not contain any vertices that have edges linking back to itself or that does not contain any pairs of vertices that are joined by multiple edges - is known as *simple.* Furthermore, a simple graph can be formed of vertices that are adjacent to each other, this type of graph is known as a *complete graph*, a complete graph comprising n vertices is denoted by K_n. Moreover, a graph can be considered *directed* or *undirected.* A directed graph (or digraph) is a graph in which the edges have orientations, and an undirected graph is a graph in which edges have no orientation. However, an undirected graph can be transformed into a directed graph by associating a direction with each edge, and this is commonly known as an *orientation* [153]. An important concept of graphs is the *connectivity* between each vertex v and any other vertex w. If any v can be reached from any other vertex w we can denote this relationship through a chain of adjacent vertices. This relationship can also be considered to be an alternating sequence $[v_0, e_0, v_1, e_1 \cdots, v_k - 1, e_k, v_k]$ of vertices and edges that connect v_0 and v_k. This sequence is known as a (v_0, v_k)-*walk.* A walk is considered *closed* when $v_0 = v_k$. A walk is considered to be a *trail* when all of the edges are distinct, and a *path* is a trail with distinct edges. In a directed graph a *directed-walk* would follow a specific orientation such as $\langle\overrightarrow{v_i, v_{i+1}}\rangle$. Following this, the notion of a path is used to define a graph as being *connected*, when there exists a path between each pair of distinct vertices. Furthermore, based on this property a graph could consist of a collection of components, where each component is a subgraph. Table 2.1 gives a glossary of terms in graph theory and Table 2.2 gives a glossary of terms in network analysis both of which are used throughout this Thesis.

Weighted Graphs

A common extension of basic graphs is to assign weights to edges (or arcs), this type of graph is known as a *weighted graph.* Formally in [153] the authors defined a weighted graph to be a graph for which each edge e has an associated real-valued number $w(e)$ called its weight. Commonly weighted graphs are used in the literature to identify subgraphs denoted $H \subseteq G$, with

Vertex: A vertex in a graph represents a fundamental unit of a network, in computer science this is referred to as a *node*.

Edge: An edge in a graph represents a connection between two vertices, the connection can denote physical links, physical interactions, intangible connections or logical connections.

Tour : A tour of a graph G is a closed walk that traverses each edges e in G. An *Euler tour* is a tour in which all edges are traversed exactly once.

Path: A path is a sequence of distinct edges which connects a sequence of distinct vertices. In a directed graph a path has a constraint, that the edges are all directed in the same direction.

Subgraph: A subgraph of G is a graph H that consists of a subset of vertices and edges in G, such that the end points of the edges in H are also contained in H.

Component: A component or *connected component* of G is a subgraph of G that is connected and not contained in a connected subgraph of G with more vertices of edges.

Table 2.1: Glossary of terms in Graph Theory [130, 153]

Vertex Degree: The degree is a property of a specific vertex and it represents the number of edges that are connected to it and is denoted by $\delta(v)$ - loops are counted twice. A directed graph has both an in-degree, denoted $\delta_{in}(v)$ - quantity of edges that direct towards the vertex, and out-degree, denoted $\delta_{out}(v)$ - quantity of edges that direct away from the vertex, for each vertex $v \in V$.

Eccentricity: The eccentricity of a vertex (or node) u gives the maximum distance between u and any other vertex $v \in V$ in the graph (or network).

Diameter: The diameter of a graph G is the maximal distance between any two vertices u and v in the network. Therefore the diameter of the graph is the largest eccentricity value in the graph (or network). Formally this is defined to be the maximal shortest path between any two vertices: $diam(G) = max\{d(u,v)|u,v \in V(G)\}$.

Radius: The radius of a graph G is the minimal distance between any two vertices in the network, or the minimum eccentricity observed. Formally this is defined to be the minimal shortest path between any two vertices: $rad(G) = min\{d(u,v)|u,v \in V(G)\}$.

Betweenness Centrality: Defines the centrality of a vertex u by considering the fraction of shortest paths that cross u. The more such paths, the more important u is to be considered. Formally, the betweenness centrality of a vertex u is denoted by:

$$g(u) = \sum_{s \neq u \neq t} \frac{\sigma_{st}(u)}{\sigma_{st}} \tag{2.7}$$

where σ_{st} is the total number of shortest paths linking nodes s and t, and $\sigma_{st}(u)$ is the number of those paths that contain node u.

Table 2.2: Glossary of terms in Network Analysis [130, 153]

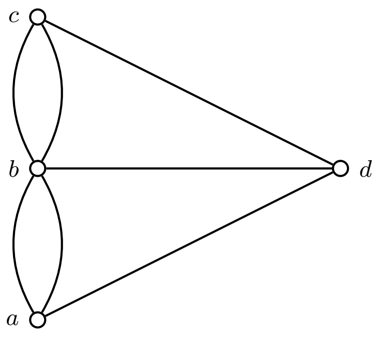


Figure 2.1: Eulerian Graph $G = (V, E)$, where V denotes a set of vertices that represent the land masses for Königsberg and E denotes the set of edges that represent the bridges connecting the land masses.

a maximal (or minimal) weight in order to determine the distance between two vertices. This type of graph allowed mathematicians such as Edsger Dijkstra in [62] to investigate problems such as the shortest path problem - finding the shortest path(s) from a source vertex v to a destination vertex u, where the shortest path is determined by finding path that yields minimal cost of the combined distance denoted $d(v_i, v_{i+1})$, for each segment of the (u, v)-*path*. Dijkstra's algorithm is considered to form the core of many routing algorithms [153].

Trees

Another important concept in graph theory is that of *trees*. A tree denoted by T is a special type of graph that does not have a cycle, known as an *acyclic graph*, that is, between any two vertices u and v in G, messages can only travel through a unique path. A tree can also be seen to be a simple graph. Just as in nature, a set of trees is called a *forest*, therefore a forest is a graph in which every connected component is a tree [33]. Trees have many applications in computer science, mathematics etc. For example, Dijkstra's algorithm - a method of solving the shorest path problem, the solution is essentially a tree $T(u)$ that is rooted at vertex u, and each different vertex considered along the (u, v)-*path* is a different rooted tree $T(v_i)$. In general shortest-path problems have been consideed to many networks - that are typically modelled in the literature as trees, such examples are communication networks [119, 121, 126] and vehicular traffic networks [80], where the requirement of both examples is to locate the path the yields minimal transportation cost between two given endpoints A and B. A specific type of tree is known as a spanning tree, a spanning tree is a subgraph of a connected graph G containing all of $G's$ vertices [153]. It is not difficult to observe that any connected graph G has a least one spanning tree (see [153] for more details). Spanning trees occur in many real-world scenarios, such an example is:

A railway company wants to expand into a 20-city area where presently they have no lines. They thoroughly examine the relevant data, and observe that for each of the $\binom{20}{2} = 190$ *pairs of cities the exact amount they would have to spend to build a direct link between those two cities. The goal of the rail road is to build a connected network - every city is reachable from every other city - but they do not want any redundant lines.* [33].

In order to solve this problem, the author considers a graph theoretical approach whereby they consider a weighted graph G, where the weights of the edges are determined by the cost of building that segment of the railway between two cities. Then the solution to the problem can be reduced to locating the spanning tree T that yields the least cost.

Similar to a graph, trees have tours. An Euler tour of a graph is a path that traverses each edge exactly once. In the context of a tree, we say that each edge is bi-directional, that is, the tree is viewed as a directed graph that contains two arcs (denoting a single edge) to denote each direction of flow between two subtrees - a connected subgraph of a tree [144]. Therefore the

Euler tour is a path along the tree that begins and ends at the root node u, and therefore traverses every edge twice - once to enter each subtree and once to leave it. So for a tree of size n - containing n vertices - the cost of an euler tour would be $T(n) = 2n$ [58].

2.2.2 Introduction to Networks

Networks have been extensively studied, and are generally classified into 4 groups *social networks*, *information networks*, *biological networks* and *technical networks*. Table 2.3 provides descriptions of the groups and their respective examples. Research into networks has changed in recent years, shifting focus away from the analysis of single small graphs (comprising tens of vertices) and the properties of individual vertices or edges within such graphs towards consideration of large-scale statistical properties of graphs that typically comprise millions or even billions of vertices [130]. This shift of focus created a movement in network research, a movement that was motivated by the change of scale and complexity of networks, forcing a corresponding change in the approach of understanding networks. This type of network is commonly referred to as a *Complex network*. A network is typically considered to be complex when they are so large that it becomes impossible to understand or predict the overall behaviour by looking into the behaviour of the individual entities or links. A further characteristic of a complex network is that the underlying system cannot be described by a single rule, and the behaviours of system entities are interdependent, that is, they depend on the behaviour of other components. Complex networks are apparent almost everywhere and if we model any real-world situation as a network we can discover novel insights that can help characterise behaviour such as detecting similarity amongst disparate systems, for example, observing that the

Group	**Description**	**Examples**
Social Network	A set of people (or animals) or groups of people (nodes) with associations between them (edges).	Friendship networks, Academic collaborations, Business relationships, and Animal behaviour
Information Network	Networks of information flow. Pieces of information are stored at the nodes. The edges between nodes signify connections between information sources.	Academic citation networks and World Wide Web
Biological Network	Networks of biological systems, where the nodes are biological entities, and the edges are connections between them.	Metabolic Networks, Neural Networks, and Food Webs
Technological Network	Man-made, physical networks, designed to transport people, resources or commodities.	Transportation networks (Roads, rail, airlines), The Internet, and Electricity Grid

Table 2.3: Types of networks [83]

organisation of the human brain resembles that of social communities.It is common for technological and information systems to be described in terms of complex networks. The internet is an example of a complex network, which is formally defined to be a topology of a large quantity of diverse and interacting entities, where in the context of the internet an entity is a computer, each linked through dense interconnections combing both organisation and randomness [6, 129].

In preliminary network theory, networks were depicted by drawing graphs on paper, and questions were answered, for example, which vertex is the most influential in terms of connectivity in the network, about the structure of the network by visually inspecting these graphs. For networks of size tens or even in worse cases hundreds of vertices this would be considered a trivial task. However, for networks of size millions or even billions (not uncommon) of vertices answering such questions would provide little meaning on the structure. This is because no single vertex, when removed from the network, will have substantial impact on the connectivity [130]. Furthermore, it would be an increasingly difficult task to provide a meaningful representation of such networks even with sophisticated graph rendering tools such as Gephi [23]. This challenge forced researchers to change their approach of understanding simple networks towards developing novel methods that can quantify larger networks in a similar way [130]. The primary focus of this work aims to do three things. First, identify statistical properties or characteristics such as length of paths, degree distribution etc., that can be used to characterise the structure and behaviour of the system, and suggest appropriate ways to measure these. Second, create models of networks to help generate understanding and meaning of the properties identified. Third, predict the network behaviour on the basis of the structural properties identified and the local rules that govern the individual vertices. The scientific community have begun investigation into the first two by drawing on ideas from physics and mathematics. However, studies on the effects of structure on system behaviour are still in the infancy stages. It will be interesting to see what developments appear in this area. Researchers have extensively reviewed the first two by drawing on ideas from a broad variety of disciplines such as physics, sociology and mathematics. However, studies on the effects of structure on system behaviour are still in the infancy stages. It will be interesting to see what developments appear in this area. A common method of identifying structure in networks is through community detection.

Definition 2.2.2. Community detection is a method that is designed to identify community structure in a network, that is, the division of network nodes into groups within which the nodes connect more to each other than any other nodes in the network, for example, in a social network a community can be defined as people that are related to each other, or they share similar interests. The ability to find and analyse such groups can provide valuable help in understanding and visualising the structure of large networks [131].

Community detection is important method of identifying structure and organisation in a network, it provides insight into groups of vertices that share common properties or play similar roles in the network, and allows for their classification. Several algorithms have been proposed in the literature to find communities in a network such algorithms can be classified into types: divisive algorithms are "top down" methods that detect inter-community connections and remove them from the network in order to isolate communities, agglomerative methods are based on hierarchical clustering methods derived from computing science and sociology [130], that is, they are "bottom up" methods that recursively merge nodes or communities of nodes based on some similarity property, and optimisations methods are based on optimising an objected function, typically this function is modularity; see definition 2.2.3 for details. In this thesis we focus on three different community detection algorithms, each of which are concerned with optimising an objective function (modularity). First, [31] propose a heuristic method called *Multi-level* that is a combination of the second and third class of algorithms. The algorithm is based on an agglomerative (bottom up) approach, where initially each node is assigned to its own community, so there are as many communities as there exists nodes in the network. Then, for each node i we consider each of its neighbours j and evaluate the gain in modularity that would take place by moving i from c_i into the community of j denoted c_j. Node i is only moved if the modularity score is increased (in case of a tie we use a breaking rule). This approach repeatedly moves nodes between communities until no improvement can be achieved with respect to the overall modularity score of the network. Second, [40] propose a greedy algorithm called *Fast-Greedy* that is concerned with finding the local optimum modularity score at each step with a hope of finding the global optimum modularity score on conclusion. The algorithm is based on an agglomerative approach similar to [40], however, instead of moving nodes, communities are merged based on the condition that an increase of the overall modularity score is observed. This approach repeated until no improvement can be observed., and Third, [136] propose a method called *Walk trap* that is considered to reside in both the second and third classes of algorithms for community detection. The algorithm uses random walks, see appendix C for details, to measure the similarity between nodes and communities in the network. This approach is based on the idea that with high probability random walks in the network tend to get trapped into densely connected parts corresponding to communities [136]. This provides a measurement of structural similarity between vertices and communities in the network. This measurement is then applied to an agglomerative algorithm, where vertices or communities (the two are the same in the first step of the algorithm) are merged based on the structural similarity between them. Finally, the quality of partitions of the network are evaluated using modularity.

Definition 2.2.3. The modularity of a partition is a scalar value between -1 and 1 that measures the density of links inside communities as compared to the links between communities. In the case of weighted networks the modularity for a partition C is defined to be

$$Q(C) = \frac{1}{2m}\sum_{i,j}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j) \tag{2.8}$$

where c_i is the community to which node i is assigned, A_{ij} is the weight of the edge between nodes i and j, k_i is the degree of node i, m is the total number of edges in the network, and the function $\delta(u, v)$ is 1 if $u = v$ and 0 otherwise.

However, modularity optimisation algorithms are subject a constraint or limitation, known as resolution limit. The resolution limit was first identified by Fortunato and Barthelemy in [73] where they showed that communities with edge numbers $\leq O(\sqrt{m})$ may not be detected through community detection algorithms.

In a less algorithmic and more statistical approach one can characterise the structure of a large scale network based on the its degree distribution. Recall that the degree of a node is the number of edges that are that are connected to it. The degree distribution denotes the fraction of nodes in the network that have each degree value k, equivalently the degree distribution denotes the probability distribution that a vertex chosen at uniformly at random has degree of value k. The degree distribution is then used to discover a further property of a network, and this property is based on the likelihood that the degree distribution is drawn from a power-law distribution. For networks with high likelihood of satisfying this property, referred to as *scale-free* networks, it is suggested that most of the nodes have small degree (or few connections) and fewer nodes have degree that greatly exceeds the average for the entire network. Scale-free networks were originally considered in Price's network of citations between scientific publications [53], where Price explained the occurrence of power laws in citation networks which was based on an interesting observation that the number of links to research papers (i.e. citations) had a heavily tailed distribution. More recently, it was observed that through the extraction of a subset of the World Wide Web (WWW) that some nodes, referred to as hubs, had a larger degree than others in the network, and therefore the degree distribution of this network can be drawn from the power-law distribution, suggesting that the subset of the WWW is scale-free. Other work includes studies into social and biological networks showing that they follow the same pattern of heavily tailed distributions, and therefore can be considered to be scale-free. A property (or characteristic) of scale-free networks is the node hierarchy, that is, small hubs are followed by other nodes of degree that is even smaller. This property means that these networks have a certain level of robustness or fault tolerance, this is due to the vast majority of nodes having small degree, and therefore if faults occur at random the likelihood that a hub would be affected would be negligible. This property isn't without its drawbacks, for example, if a significant event occurred on a hub node, the network could either be split into isolated graphs, or be left paralysed.

2.3 Big Data

We are in the "Big Data" era, where many enterprises are continuously aggregating large volumes of data that records customers interactions, product sales, performance metrics e.g., Key Performance Indicators (KPI's), diagnostic intelligence e.g., telemetry data on a network topology, and many more. The ability to perform scalable and efficient processing on these data sets has become a major ingredient for the success of large enterprises [21]. Fortunately Apache Hadoop [19] provides a cost effective method of storing and processing these large data sets. Data files are uploaded into Hadoop through one of many interfaces such as Hue, Hadoop Distributed File System (HDFS) CLI etc., and the file is stored in a distributed file system known as HDFS. In order to process the stored data in order to gain insight into a particular problem or subject area such as diagnostics on a network topology, one has to use parallel processing frameworks such as MapReduce [54]. MapReduce is a programming model and associated implementation for processing and generating large data sets on commodity hardware. However, the framework requires users to create custom MapReduce tasks that implement their own processing logic by specifying customised map() and reduce() function. In order to create such tasks one must be familiar with programming languages such as Java, Python, R or Ruby, and the MapReduce framework. Hadoop handles MapReduce jobs by evenly splitting the targeted data set into blocks of size 128MB (default value), before scheduling each block to one of the processing nodes that are available. Once this processing has completed, Hadoop launches a set of reduce tasks in order to produce the final results of the function [21].

In order to enable a user to gain access and perform some operation on the data, Hadoop relies on tools such as Apache Hive and Impala [41].

Apache Hive and Impala are tools that provide access to data stored within Hadoop through an SQL-like syntax. Apache Hive is a tool for Hadoop that enables users to perform processing tasks over large data sets without being restricted by having to gain an understanding of the underlying complexities such as MapReduce of Hadoop. Apache Hive [20, 156] was the first SQL-like language that was built on top of Hadoop to exploit the advantages of Hadoop such as scalability, resilience and high availability over other similar systems. A Hive query is achieved through the Hive SQL syntax known as HiveQL. A HiveQL statement that is submitted to Hive is parsed, compiled and optimised in order to yield a query execution plan - directed acyclic graph (DAG) of MapReduce tasks which are executed through the MapReduce framework [21]. In Hadoop it is common for all files contained in a single folder in HDFS to be represented by a Hive table. The data enclosed in the table is then interpreted from these files at query time based on the definition of the fields, which are stored in Hives' metadata.

Cloudera Impala [41] is another SQL-like language that provides scalable processing capabilities over data stored in Hadoop. In comparison to Hive, Impala is framework independent, it provides its own long running daemons that are attached to each node of the Hadoop cluster. The main daemon process comprises the query planner, coordinator, and execution engine [21] that is used to coordinate and manage a queries lifecycle. Impala is considered to be a highly efficient method of processing data in Hadoop, and this is due to its efficient Input/Output (I/O) layer that reads data stored in HDFS. However, it isn't without its limitations such as requiring the work set of a query to fit within the aggregate physical memory of the cluster. This imposes restrictions on the type of datasets that it is able to process [21].

3

Towards Social Network Analytics for Understanding and Managing Enterprise Data Lakes

3.1 Introduction

The work in this chapter is concerned with *Data Lakes*. A Data lake is a single data storage capacity (or data store) for enterprise information that is aggregated from a plurality of applications into a single infrastructure. The data lake concept was motivated (or driven) by the economical utilisation of commodity storage, the incredible throughput of cost effective data processing systems and the recognition that significant value can be obtained by co-locating data from a multitude of applications. Data lakes are anticipated to be used by multiple teams of data scientists in order to create insights into business operations and propose novel applications on inferences that would have been otherwise impossible or impractical to develop. The work in this chapter presents a response to the challenge of providing automation and intelligence for managing and understanding a corpus of data assets accumulated in a Data lake. In particular, this work involves using insights and understandings (including algorithms) developed in the Social Network Analysis (SNA) community to provide understanding into the structure and organisation of data assets contained in an enterprise data lake. Furthermore, this work has highlighted a number of key questions that can be used to direct future research in this space.

The study is structured as follows. First an outline of the motivation 3.2 for investigating this area detailing specific use cases 3.2.1 some of which are addressed in this work. In section 3.3 we provide a basic introduction to complex systems and networks, similar to that in chapter 2, and how they are related to our problem. Section 3.4 discusses the data used in our analysis. Section 3.5 is concerned with a review of the functionality and technical approaches taken in the development of a tool for creating insights into the emergent relationships between data assets in a data lake, which is a defined use case in 3.2.1. In particular the tool is develops the capability to: discover connections between data assets, manage discovered links, and perform evaluation and management of the resilience of the data or network. Section 3.6 is concerned with performing experiments that provide insights into the structural and organisation properties of the data store. Finally, section 3.7 provides conclusions of our work and our perspective on the unmet technical challenges, whilst outlining future research opportunities that is motivated by the need to manage data lakes of the type considered.

3.2 Motivation and Background

In our investigation we consider a data lake for an organisation that operates a large and diverse range of infrastructure for hosting, querying and analysing data. The infrastructure comprises in excess of 2,700 registered relational database instances utilising over 3.7 Petabytes of storage; a variety of mainframe databases, and two production Hadoop [171] clusters, each deployed with 1.2 Petabytes of raw storage. The Hadoop facilities are provide to the organisation on a self-service basis through a multi-tenancy platform that is known as Hadoop-as-a-Service (HaaS).

In many organisations a traditional database estate is managed as a federated data architecture [84], that is, a collection of well understood and highly controlled data domains linked via an overarching enterprise data model. The adoption of a single 'Big Data' infrastructure challenges this approach. The autonomy of the domains means that creating and maintaining a single overview of the asset is a new and labour intensive requirement. Conversely the adoption of the federated pattern has been driven by business dynamics and funding challenges of the enterprise and a return to a centralized single domain is impractical. We have found this clash of requirements (autonomous management of data origination and information integration) especially problematic when we need to rapidly bring data assets together in Hadoop to answer novel questions - a key attraction of a big data asset.

In common with Facebook [157] we have found that simply discovering data is a non-trivial issue. Likewise, understanding data lineage and the emergent relationships amongst assets that are landed in a data lake are problems that only get worse with scale. We note that these difficulties are not specific to Hadoop. We can see the evidence of this in the discrepancies between published schema for relational database assets versus the live data, and the widespread reliance on 'finding an expert' to help discover relevant data for a given task and understanding its (undocumented) idiosyncrasies. These observations and experiences have led us to explore alternative approaches to understanding and managing large scale data assets, in particular approaches that don't depend on top-down centralised curation.

3.2.1 Use Cases

A key goal of our work is to address a series of use cases for understanding and managing Data lakes, these include:

- Data Ownership - understanding data asset ownership and usage
- Data Science - discover relationships to identify relevant or related data assets, and help locate core data.
- Information Security - identify and understand the emergent risks present in assembled data and the consequences of adding new data or access rights.
- Organisational design - understand, identify and connect the social network of people around data assets.
- Application Ownership - exploit data gravity to build business cases by understanding what is already in the Data Lake.

There are two general areas of application that our work is focussed on. First, data discovery and utilisation, this is concerned with extracting value from data assets. In particular, Section 3.5 discuses a tool that we have developed for the purpose of data discovery, and Section 3.6 provides a series of experiments that provide insight into the organisation of data assets in a data lake. Second, protecting corporate assets, this takes the perspective that data is a corporate asset that must be protected and it must be understood that any threat is a potential accident waiting to happen. In particular, Section 3.5 discuses the capability of using the proposed tool for managing the emergent risks present in a data lake. The following is a descriptive list that provides insight into relevant problems that can be addressed in this research area. However, in this work we only focus on a few of these, namely, *finding relevant data*, *organisational design*, and *risk management*.

1. Data discovery and Utilisation

   Data Lakes provide organisations with an asset that could be leveraged to create value. To get at the value assumed to be available, Data Scientists must be able to identify and use the data within them (see class 2 of these use-cases for issues with this perspective). Drilling down we identify two specific open issues.

   (a) Finding relevant data - the data lake we examine in this work has 3000 data tables. In our experience data owners generally fail in their obligation to properly document the data in the asset (again see class 2). Discovery of assets that can be used in an investigation is a starting point for a Data Scientist; finding a data table is just the beginning of a series of conversations that eventually reveal the usefulness of the information that it contains. The tools we present are reviewed, below, from the perspective of how useful they are in pursuing this detective work.

   (b) Promoting and generating value - residual and unknown value in data sets is of deep concern to data owners. Data which has shareholder value may be destroyed inadvertently, or conversely data which has low value may be kept expensively, riskily and unnecessarily. Our tool enables the discovery of relationships amongst data assets that can be used to expose this residual value.

(c) Organisational Design and Expert Detection - the pattern of use of Data Lakes may reveal issues or features in the structure of the organisations that host them. Additionally discovering key users who link spheres of knowledge and ensuring that they are not inadvertently lost from the organisation is of interest [34].

(d) Data Documentation - The use of data has changed radically in the last few years, as has its impact on the enterprise. Documentation of data assets was required to support the applications that those assets hydrated, but now it is required to support ad-hoc investigations into unseen and unimagined business investigations. We are not aware of a body of work that provides scientific information to inform the construction of appropriate data documentation in the modern context. The techniques and tools presented in this work provide one initial approach to tackling this issue; what metrics can be extracted from Data Lakes using known techniques? We attempt to provide information on how useful these are at the moment.

(e) Application Ownership - Co-location of data enables rapid exploratory analysis (LAN bandwidth, security processes, and data export and staging limitations impose substantial barriers in terms of time and effort). Some applications generate key data resources frequently reused by analysts, identifying these allows architects to identify what should be ported to where to deliver agility and fidelity (because in some applications older data introduces greater uncertainty), this is particularly important for legacy systems.

2. Risk Management and Security

Whereas data may be of great value to an organisation it is increasingly also a potential liability. There have been a number of high profile data breaches in recent years where personal and corporate data have been stolen and published, some of which include [26, 42, 47]. These demonstrate that professional and well-funded organizations can be compromised and substantially damaged at any time. Risk management in a Data Lake has two aspects; removing dangerous data and detecting potential future attacks.

(a) Removing dangerous data - One of the reasons for developing Data Lakes is that they contain diverse and extensive data assets, some of which contain data that enable unanticipated inferences. This feature allows the Data Scientist to create business value, but it exposes Data Owners to the risk that unexpected forbidden insights into corporate finances or customer privacy may be made and used maliciously. Can Social Network Analysis techniques reveal risky tables that link domains in a potentially dangerous way?

(b) Potential future risks - extending 2a, can we use Social Network Analysis to determine areas of the Data Lake that are relatively risk free in the face of future expansion and innovation or conversely are high risk and therefore requiring careful review and policing to prevent the possibility of risky novel links being introduced?

3.3 Complex Systems and Networks

It is common for technological and information systems to be described in terms of complex networks. A complex network typically has a topology of a large quantity of diverse and interacting entities (vertices), each linked through dense interconnections combing both organisation and randomness [6, 129]. Complex networks can be characterised by the fact that the underlying system cannot be described by a single rule, and the behaviours of system components are interdependent - they depend on the behaviour of other components. In our work we consider a Data Lake to be a complex system that is formed of many data assets; each described by database tables that may or may not have some interdependency relationship. In our representation the entities, e.g., the data assets, are denoted by vertices in the network, and the relationship between entities is denoted with an edge. A network approach to understanding a system is a natural method that is commonly used across the literature [148]. In networks the only relevant information with respect to detecting relationships between system entities is connectivity - the number of entities that are connected to a single entity of the system; referred to as neighbours in networks. The relationships between system entities can represent physical links, physical interactions, intangible connections, logical connections etc. In this work we consider the relationship between system entities to be logical - data assets are interconnected by their intrinsic properties e.g., column names. In the remaining sections of this work we denote the network by an abstract graph $G = (V, E)$, where V is the set of data assets, E represents the set of logical connections between pairs of data assets, and a logical connection between two assets suggests that they have one or more columns in common.

3.4 Data

The data we are examining comes from one of our early Research Hadoop clusters and comprises information about over 3000 Hive [156] tables. These were used to store data on infrastructure, telemetry and telecommunications network faults. The Metadata for the Hive tables was roughly 20 Megabytes of records that were extracted from the Hive Metastore. The following procedure was used to extract and prepare the data.

1. Execute an SQL query on the Hive Metastore to extract - for each Hive table - a list of columns and associated data types. The query involves a join across several tables in the Hive Metastore schema including: TBLS, SDS, COLUMNS_V2, and DBS.

2. The output is a multi-dimensional list of records saved to a text file, where a single record represents a specific column for a Hive table. A single record is represented by a tuple: {NAME, TBL_NAME, COLUMN_NAME, TYPE_NAME, TBL_TYPE}, where TBL_NAME is a the same for a given table.

3. The resulting file was analysed using Pig [132], a scripting language for Hadoop that allows developers to perform complex data transformations without having to know the MapReduce framework, and thus an alternative to Hive QL or Impala, in order to identify the logical relationships between Hive tables. In the context of this work the relationship between a pair of tables is determined by the quantity of columns that they have in common. The output of this step is a text file comprising a list of records; each representing an edge, where a single record (or edge) is a tuple {source, target, commonColumns, relativeCommonColumns}, the items contained in the tuple (or record)

Property	**Description**
source	the source Hive table
target	the target Hive table
commonColumns	quantity of columns in common between the source and target tables
relativeCommonColumns	proportion of columns in common between the source and target tables to the total number of columns between the two tables

Table 3.1: Description of the properties for a single edge in the graph

Property	**Value**
Nodes (Tables)	3,052
Edges (Column Matchings	551, 466
Connect Components	26
Diameter	7
Clustering Coefficient	0.814

Table 3.2: Global properties of the Hive graph

are described in Table 3.1.

The output of step (3) in the procedure was used to create a graph $G = (V, E)$ of relationships, where each $v \in V$ denotes a Hive table and each edge $e \in E$ between any two Hive tables suggests that they have at least one column in common. These relationships could be considered to be directional such as one-to-many or many-to-one but for purposes of this work we consider the graph is undirected. We give a brief summary of the global properties of the relationship graph in Table 3.2. The properties chosen are commonly considered in the literature as discussed in [129, 130] for generating understanding of the structure of a complex network. An immediate observation of the graph is that the degree distribution can be drawn from the fitted power law distribution with p-value 0.02, where p-value defined to be the probability of a null hypothesis, that is, it has 0.02% chance of not being a power-law distribution, suggesting that with high likelihood the degree distribution of the network is drawn from the power-law distribution. Recall in Section 2.2, a power-law degree is a commonly used to method of categorising a network as being scale-free. This property indicates that very few vertices in the high degree, large amount of vertices have low degree in the network. This observation combined with previous knowledge that edges represent column matchings suggests that lineage relationships might exist between these hub vertices (Hive tables) and the rest of the network. This prompted investigation into the structure behind the network, and in particular the communities that exist (see below, section 3.6). Although our Data Lake is implemented using Hive on Hadoop and our tool (see below, section 3.5) uses the Hive Metadata as the basis of its inference and functionality, these structures are derived from the tabular pseudo relational nature of the data design, something that is common to many modern and legacy data stores, and hence the approach can be generalised.

3.5 Data Lake Introspection Tool

The Data Lake Introspection (DLI) tool provides an environment for discovering relationships that appear amongst Hive tables stored in Hadoop. DLI uses a combination of graph analytics and visualisation mechanisms to facilitate this analysis. DLI is a web-based application that was created using R [74], Shiny [143], igraph [49], and visNetwork [155] to present users with a graphical representation of the Hive graph and facilitates exploration of the relationships amongst the assets contained within a Hadoop cluster. The tools were chosen based on the familiarity of them with the authors.

DLI provides an overview of the Hive graph through simple graph statistics, these statistics include: (1) quantity of tables and edges, (2) diameter of the graph is the longest shortest path between any pair of vertices (or Tables), or in this context the longest chain of similarity observed between a pair of Hive tables, and (3) number of connected components gives indication of the number vertices that are connected to each other by paths but are not connected to any other vertex in the graph, that is, this quantity signifies the number of partitions (or communities) that are inherent in the graph based on the quantity of columns in common. The statistics are supplemented by a chart depicting the distribution of the connect component sizes, which might provide insight into the size of the communities that are formed. This information can enable a user to gain an appreciation of the graphs' scale and its structure e.g. is there a giant component of tables or are there many disconnected islands. Furthermore, DLI provides a tabulated list of Hive tables, where each table is described using a three properties. First, betweenness centrality will provide insights into the influence that a specific Hive table has within the network. Second, degree will provide knowledge of how many other Hive tables are similar to itself. Third, component the Hive table resides within can provide insight into its locality in the network. This list is searchable and can be ordered by any of the table properties, thus enabling quick searches for tables of interest. At the point a user selects a particular table in the list DLI produces an interactive graph of its ego-network, that is, a subgraph $G' \subset G$ that only contains the selected Hive table and Hive tables that are directly connected to. This filtering ability provides emphasis on connected tables whose connectivity pattern differs from that of the selected Hive table. The goal here is to highlight tables that provide the user with novel information and are not simply duplicates of the table they have started from. Highlighting a table in the ego-network furnishes the user with information on its schema and other relevant Metadata. This information provides invaluable insights, that would have required a series of conversations with applications and data owners, for a data scientist of the data sets available in the data lake.

In the case that a pair of tables are selected, the graph will be drawn depicting, if it exists, the shortest path between the tables, i.e. the join relationships that allow the Data Scientist to connect information in the first table with that in the second, and thus to assemble a target schema. In our data, the longest such path comprises six joins. Finding this would be a formidable task for a human, but is trivial using the tool.

Whilst the Data Scientist will appreciate the facility to easily discover linkages amongst tables, those tasked with managing the emergent risks present in assembled data might be troubled at the ready exposure of such information. However, DLI provides a facility to identify the minimal list of edges (the minimum-cut [79]) whose removal will disconnect a pair of tables (i.e. prevent them from being joined). Using this functionality the user can simulate the removal of edges and inspect the impact on the data graph. By attributing business value to specific joins in the graphs (as an edge weight) the minimum-cut would effectively find the minimum cost set of joins that need to be 'removed' to prevent joining of a pair of tables. For example, by applying a risk score of the data assets to the graph as edge weights, one can protect sensitive data assets by using this intelligence to monitor the connecting Hive tables, and enforce restrictions on user abilities such as permissions for these tables.

Collectively, these mechanisms support data discovery tasks and can give a measure of the emergent risks present in assembled data and the means to manage the addition of new data or access rights to end users with a view to preventing unintentional outcomes, such as the exposure of sensitive data assets.

3.6 Experiments

In Section 3.5 we described DLI and the information that it provides to support the interpretation and management of the Hive Graph. In this section we present and discuss a number of key observations that have been revealed throughout our study of the Hive Metadata Network.

3.6.1 Data refinement

The goal of this work was to combine business meaning and network analysis techniques in order to gain insight into the structural and organisational properties of data assets for relationship discovery. The business meaning aspect of our work involved discussions with Data architects to identify a set of columns they recognised as being fundamental to joining assets contained within the Data Lake. The set of columns identified were used to refine the original graph, and the new graph characteristics are detailed in Table 3.3. The graph was refined by removing Hive tables from the graph that do not contain any column in the set of identified columns, and this graph is considered for all experiments in this work. Although the quantity of vertices and edges has been reduced by using business columns, we observe that the entire refined graph is contained within a single connected component.

Property	**Value**
Nodes (Tabes)	921
Edges (Column Matchings	24, 405
Connect Components	1
Diameter	6
Clustering Coefficient	0.3390

Table 3.3: Refined Graph Properties

3.6.2 Network Communities

Community detection [72] is a method of identifying structure or organisation in a network. It is an important concept in complex networks as it provides insight into groups of vertices that share common properties or play similar roles in the network, and allows for their classification. In this work we use the community detection algorithms mentioned in Section 2.2.2 to discover the organisation of Hive tables in the graph. In particular, we consider a set of experiments to discover and investigate the communities in the graph. This section of work is structured as follows. In section 3.6.2 we conduct an experimental study to detect communities using the community detection algorithms, and evaluate the results based on the modularity score (see Definition 2.2.3). In section 3.6.2 we investigate the detected communities through a chosen detection mechanism, which is based on the modularity score, to identify properties for characterisation of the detected communities.

Detecting communities

This work involved conducting an experimental study on the refined network using R and igraph to provide a clear understanding of the natural communities that can be formed. In particular we performed four experiments, each using a different method of identifying communities, these include: a) partition by database name - a naive but logical approach that assumes the pre-determined structure of the tables will provide a good partition of the network, b) partition using the Fast-Greedy algorithm [40], c) partition using the Multi-level algorithm [31], d) partition using the Walktrap algorithm [136], and present the number of communities identified with their associated modularity scores in Table 3.4. These methods were chosen based on their common use in the literature, and implementations of them existing in

Method	**Number of communities**	**Modularity Score**
Database Name Partition (1)	31	0.0099612
Fast-Greedy (2)	8	0.590688
Multi-Level (3)	9	0.589229
Walktrap (4)	11	0.5887023

Table 3.4: Community detection - resulting properties of community detection on the network.

R. For the experiments b, c, and d we consider functions in the R igraph library which are implementations of the respective algorithms, where the input parameters of the functions were (1) input graph is the refined graph that was represented using an R data frame, and (2) edge-weights provide the quantity of columns that are matched between a pair of Hive table, and the output of the functions is a *communities* object that comprises many properties (see [49] for details on the properties of the communities object), but in this work we are only interested in two, namely, *modularity*, which is a numeric value that denotes the modularity score of the partition in the input graph, and *membership*, which is a numeric vector that gives the new membership of the vertices in the input graph, both of which are dependent on the chosen algorithm. An immediate observation based on the results in Table 3.4 is that applying a community detection algorithm improves the modularity score by a factor of around 60, suggesting that partitioning our network based on the database name is not an effective method of detecting meaningful communities. A further interesting observation is that more detected communities does not necessarily suggest a better modularity score.

Investigating Communities

Here we perform analysis on the communities detected in order to investigate the structural properties of the network. This work involves three progressive experiments 1) examining the stability of the detected communities, 2) perform community decomposition into communities that are described by the database name, and 3) investigation into the modularity of the network, that provides insight into the structure of the network and its associated characteristics.

Community Stability - in this experiment we test whether the structure we observe is consistent across a range of community finding algorithms and not simply an artefact of a particular algorithm. We expect to see low variation in community membership if the communities themselves have practical significance and real-world utility in a business application. The results of this analysis are given in Table 3.5, wherein direct mapping denotes the average pairwise equality score between similar communities for two community detection algorithms, that is, the average similarity between the set of vertices that are identified to be a community for each of the community detection algorithms. The similarity between a set (or community) is computed using the Jacard Index [164] denoted by

$$J(C_a, C_b) = \frac{C_a \cap C_b}{C_a \cup C_b} \tag{3.1}$$

,where C_a and C_b are the set of vertices identified to be communities using two community detection algorithm a and b respectively. In order to compute the average similarity (or stability score) between two community detection algorithms we propose Algorithm 1, which is motivated by the Jacard Index and its ability to compute a similarity between two sets. As expected, the membership of communities shows a high degree of consistency giving us confidence that they are not simply artefacts of any given algorithm.

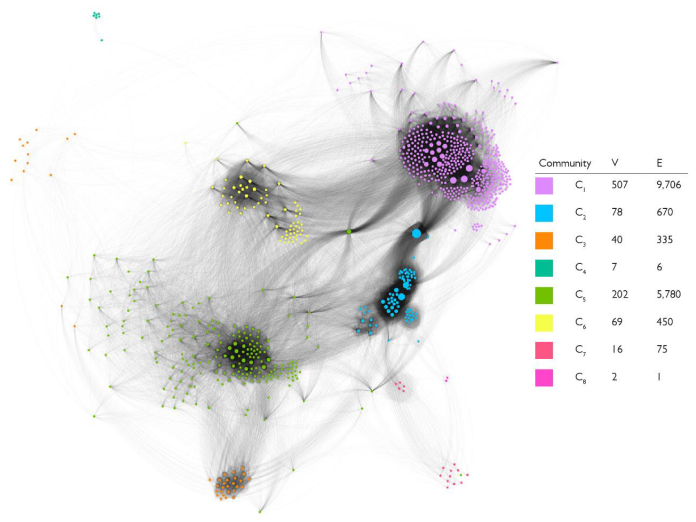
Comparison	**% Direct mapping**
Fast-Greedy vs Multi-level	80
Fast-Greedy vs Walktrap	66
Multi-level vs Walktrap	63

Table 3.5: Community Stability - mapping score between communities detected for different detection algorithms computed using Algorithm 1.

Algorithm 1 Community Stability

Input: C_1: partition of the network based on a community detection result; C_2: another partition of the network based on a community detection result.
Output: *result*: community stability score

```
1: $counter \leftarrow 0$
2: $scores \leftarrow []$
3: for each $c_i$ in $C_1$ do
4:     for each $c_j$ in $C_2$ do
5:         if $c_i \cap c_j \neq \emptyset$ then
6:             $scores[counter] \leftarrow \frac{|c_i \cap c_j|}{|c_i \cup c_j|}$    ▷ compute stability between $c_i$ and $c_j$
7:             $counter \leftarrow counter + 1$
8:         end if
9:     end for
10: end for
11: $result \leftarrow 0$
12: for each $score$ in $scores$ do    ▷ compute the overall stability score
13:     $result \leftarrow result + score$
14: end for
15: $result \leftarrow \frac{result}{counter}$
16: return $result$    ▷ stability score between $C_1$ and $C_2$
```



Community		V	E
🟪	C_1	507	9,706
🟦	C_2	78	670
🟧	C_3	40	335
🟩	C_4	7	6
🟩	C_5	202	5,780
🟨	C_6	69	450
🟥	C_7	16	75
🟪	C_8	2	1

Figure 3.1: Communities Detected - representation of the detected communities in the Hive Metadata Graph using graph visualisation tool Gephi [23]

Community Breakdown - this work is a natural extension of community detection, which provides insight the similarity between the business expectation of data asset communities such as schemas and databases, and the actual communities that were detected in the previous section. This insight would suggest that schemas and database names do not reflect the data they describe, and thus suggesting that federated architectures are not ideal for database systems of the future. Community breakdown is achieved by a graph reduction algorithm to each of the detected communities that were identified through Fast-Greedy, where Fast-Greedy was chosen because its yields the best modularity score. Figure 3.1 provides a representation of the detected communities through Fast-Greedy using Gephi, a graph visualisation tool. The procedure considered for graph reduction is described by two successive algorithms. Algorithm 2 called graphReduction, is used to reduce the graph into communities that are described by their database name membership, and the convergence of this algorithm is based on the output of

Algorithm 3 called validateComponents. The procedure is described as follows. Line (1) computes the connected components of the input graph, Line (2) identifies the vertex or edge in the graph that has the largest value of a defined property denoted as *prop*, where the property can be betweenness centrality, vertex degree or edge weights. Line (3) removes the identified vertex or edge, and line (4) recomputes the connected components of the graph with the modification applied in line (3). Line (5) uses Algorithm 3 to check for convergence, where convergence is achieved when communities (or connected components) are decomposed into sub-communities that are described by their database name membership. The algorithm is detailed as follows. Line (1) initialises a vector denoted as $\vec{C}$ that will contain the number of unique database names in each connected component of C_0. Line (2) assigns the membership to each $v \in G$ based on the membership of $v \in C_0$. Line (3) initialises an integer value that will represent the number of connected components in G. Lines (4) to (9) is a *For* loop that determines the quantity of unique database names that are contained in each connected component denoted by c_i. More specifically, line (5) identifies the vertices in G that are contained in c_i, line (6) identifies the number of unique database names that are contained in component c_i, and line (7) adds this quantity to the vector $\vec{C}$. Line (8) increments the number of connected components counter. Line (9) computes the mean number of unique database names per connected component in G denoted as $\bar{C}$. Finally, lines (11) to (14) checks if each connected component is described by a single database name, and returns a boolean value signifying this. Thus, Algorithm 3 checks that the communities in G are decomposed into sub-communities c_i that described by their database name membership. The goal of this work was to identify some quantity that can provide a measure of similarity between the expected partition, according to database name membership, and the real partition. The graph reduction procedure was applied to each of the measures and the results are presented in Table 3.6. Based on the results presented in Table 3.6 an obvious observation is that community 7 comprises Hive tables that are already associated with the database name membership. A further observation is that vertex metrics, namely degree and betweenness centrality, are not the most effective property to use in our graph reduction procedure, given that is takes on average 14% and 19% vertex removal using betweenness centrality and degree respectively, and edge weights are highly effective yielding an average of 9% edge removal. This suggests that removing edges between pairs of Hive tables that have a high column matching can yield database partitioning of the community more efficiently - as the edge-weight represents the column matching between Hive tables. Furthermore, this helps identify the bridges (cut-edges) that exist within the individual communities - edges that when removed increase the number of connected components.

Algorithm 2 graphReduction

```
Input: $G$: Hive graph, $prop$: property that is used to reduce the graph
Output: $G'$: A reduced Hive graph
1: Compute connected components of $G$
2: Identify the vertex $v$ or edge $e$ that has largest value of $prop$
3: Remove the vertex or edge
4: Recompute connected components of $G$
5: Repeat until convergence    ▷ convergence is determined by Algorithm 3
```

Modularity - this work is designed to identify the properties of the network with respect the modularity observed. In particular we investigate if the network is limited by the modularity resolution problem, detailed in Section 2.2.2, which is a limitation of modularity optimisation where in some

Algorithm 3 validateComponents

Input: G: Hive graph, C_0: components in G identified in line (4) of Algorithm 2
Input: Converged: Boolean to represent satisfaction of the convergence criteria

```
1: $\vec{C} \leftarrow []$    ▷ initialise the vector
2: Assign membership to vertices $v \in G$    ▷ membership is based on the membership of vertices in $C_0$
3: $counter \leftarrow 0$
4: for each $c_i \in C_0$ do
5:     $G' \subset G$, where $G'(V) \in c_i$    ▷ Identify vertices in the graph that are contained in component $c_i$
6:     $unique = |\{v_1, ..., v_k\}|$, where $v \in G'(V)$, and $k = |G'(V)|$    ▷ Identify the quantity of unique database names in $c_i$
7:     $\vec{C}[counter] \leftarrow unique$
8:     $counter \leftarrow counter + 1$
9: end for
10: Compute the mean of $\vec{C}$, denoted as $\bar{C}$
11: if $\bar{C} > 1$ then    ▷ check for convergence
12:     return false
13: else
14:     return true
15: end if
```

Method	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
Betweenness Centrality	25	47	8	6	14	18	0	19
Vertex Degree	23	20	7	6	14	11	0	21
Edge Weight	5	5	3	12	33	3	0	4

Table 3.6: Community Breakdown Results - Percentage of edges/vertices removed to yield sub-communities based on database name membership

Community Id	**Communities detected**	**Community Modularity Score**	**Network Modularity Score**
1	4	0.17	0.5491945
2	3	0.11	0.5785004
3	4	0.18	0.57850044
4	2	0.29	0.5906086
5	1	6.4E-17	0.5906875
6	2	0.27	0.5795237
7	1	0	0.5906875
8	4	0.27	0.5408847

Table 3.7: Community Resolution Results

cases it fails to identify modules that are of smaller sizes. This involves applying a community detection algorithm to each of the communities detected using the first pass of the algorithm in search of observing a better overall modularity score, when applying the new membership, and we present the results in Table 3.7. It is immediately obvious through the results that we do not observe an improvement in the original modularity score 0.590688, and therefore for this network Fast Greedy is not affected by the modularity resolution problem. Furthermore we try to maximise the modularity score using a technique proposed in [129], where Newman et al. propose a method of increasing the modularity score that is based on a basic graph partitioning method. The method we use is motivated by [99], and the procedure is described as follows. First we search for the vertex (Hive table) that, when moved into another community will yield the largest increase of the modularity score when the new membership is applied. The order of vertex movement does not matter in this procedure. The process is repeatedly, with the constraint that each vertex is moved at most once. Therefore the this stage of the process is linear to the size of the graph $O(n)$. Once all vertices have been moved we identify the state (if any exist) that has the greatest

Community Detection	**Community Modularity (before)**	**Community Modularity (after)**
Fast-Greedy	0.590688	0.590688
Multi-level	0.589229	0.590688
Walktrap	0.588702	0.59055

Table 3.8: Modularity Maximisation

increase in modularity, and rerun the process with this new membership until no further improvements can be experienced (convergence). The goal of this work is to verify that the network is not affected by modularity resolution, that is, the resulting communities are not just a consequence of being in the local maxima of the optimisation technique involve. The results of this experiment are presented in Table . The results suggest that although we observe an increase in modularity for the Walktrap algorithm, we cannot achieve a better decomposition of the network than through Fast-Greedy using this method.

3.7 Conclusion

In this work we report on a tool that we have developed to create understanding of real-world complex networks. In particular we have identified a methodology, based on social network analysis, to deliver insights into the relationships between data assets assembled in a data lake. By inferring potential joins between database tables and creating a graph of those tables, we are able to address a variety of practical use cases towards data governance and data science. The analysis presented here is based on a single data lake, one of our early Hadoop research clusters. Nevertheless we have established a number of interesting results about the structure of its Hive Metadata graph. Of particular note is the observation that in our graph the natural community structure does not follow that initially imposed by end users through the creation of logical databases. Instead, there is a richer, emergent structure which perhaps was not appreciated when the data was first assembled or which evolved opportunistically over time. Investigation into the stability of detected communities through community detection algorithms showed a high degree of consistency based on Algorithm 1. Further work on the stability in community detection can include a systematic analysis of cluster similarity algorithms in the literature such as RAND index [164] - this work is beyond the scope this paper. Work on the modularity of detected communities showed that the communities in the network identified through community detection algorithms provide the most meaningful partition of the network with respect to modularity. The graph reduction experiments showed that through careful removal of edges (9% on average) based on the weights (or columns in common), we could decompose the detected communities into sub-communities that are described by their database name. This suggests that the set of communities derived by naive partitioning according to database name are not too dissimilar to those detected by algorithms such as Fast-Greedy after all. This difference could be seen as one measure of the gap between the 'designed' and 'emergent' structure in a data lake. A further observation is that our Hive metadata graph followed a power-law degree distribution, therefore suggesting it to be scale-free. This implies that the content of a small number of Hive tables is influential in the structure of the network. In turn, this means that lineage relationships might be apparent between a small set of Hive tables and the rest of the population in the cluster. In future work we hope to make the following enhancements to our tool: 1) improve the accuracy with which we can automatically infer joins between tables; 2) construct a bi-partite network based on the interactions of end-users with data assets. Improved join inference might come from applying efficient indexing techniques to the content of database tables or by harvesting real-world interactions through monitoring of query logs to extract the joins that are actively being made by end users, similar to work in [157]. The same real-world interactions will also help to address the construction of a network that captures the relationships between users and the data they make use of. Through this we hope to develop a rich picture of how people and tasks are organised around data, or conversely how the organisation of data influences the structure of organisation.

In the next chapter we consider a statistical analysis approach to understanding networks. More specifically, we consider a data-driven approach, and through statistical analysis we demonstrate the challenge of understanding real-world networks when access to data that describes the network is limited in some way.

4

Vehicular Traffic Network Analysis

4.1 Introduction

Traffic management has become a major burden on local authorities; spending large quantities of money on monitoring and understanding traffic jams, and consequently with the rapid development of cities' infrastructure, traffic demand increases, generating significant economic loss - increasing travel times, fuel consumption and aggravating pollution. Traffic monitoring and detection techniques are generally classified into two groups. First, *road-side sensors* are a widely adopted method for managing urban traffic networks. This method includes a group of road-side sensors that together form an induction loop [115] at specific locations in the network, where the locations are generally points that have influence on traffic flow in the network. Induction loops are a fundamental component in Intelligent Transportation Systems (ITS), they provide ITS with high-quality real-time traffic information about the network they describe. This information is used to detection and mitigate the impact of traffic congestion in an urban environment. Second, Global Positioning systems (GPS) based traffic monitoring is a method that collects and aggregates information about a specific vehicle as it routes across the network. In this method every vehicle is equipped with a GPS that acts as a sensor for the network, detecting vehicle speed, location and the direction of travel. This information is posted to a central processing centre, where useful information can be identified about the network, such information includes traffic status, alternative routes etc [115]. Traffic flow descriptions on a *highway* were originally described by Greenshields 82 years ago in [78], where the author discussed the first documented techniques of measuring traffic flow, density and speed using photographs. In the same study, Greenshield postulated what may seem to be an obvious linear relationship between speed and traffic density denoted by $Flow = density{\cdot}speed$, suggesting that a vehicles speed reduces linearly as the density of vehicles increases.

More recently, research into traffic has evolved towards considering novel methods of understanding traffic. [162], for example, propose a data-driven *artifical neural network (ANN)*, the book by Yegnanarayana et al. [173] is a good survey of Artificial Neural Networks), to predict traffic flow on a specific road link using traffic information aggregated from inductive loops (similar to SCOOT). [104, 165] consider an algorithmic approach to analysing traffic in the network. In [165], Wang et al. consider a methodology that includes detecting sensitive regions in the network to discover structural issues. Sensitive regions are considered to be key junctions or *Hub* nodes in the network at which when they become attacked by traffic incidents or events such as signal failure, traffic accident, road works etc., cause paralysis of the network. This approach is based on analysing aggregated Global

Positioning System (GPS) sensing information; describing the travel time on a specific road link, to detect traffic jams on a specific road link over a specific period of time. Thus discovering spatial temporal traffic jams in the network. This information was used to build traffic propagation graphs. The graphs was then analysed for common patterns (or motifs) in view to identify the archetype (typical behaviour) of traffic in the network. Finally, they reviewed the archetype to discover key junctions or *Hub* nodes in the network that are sensitive to attack, and validated this approach with historical events on the network. In this approach traffic jams were detected by comparing the current travel time against road-specific free flow values; based on work in [104], and for observation periods on specific roads at which travel times were beyond some threshold above these free flow values they were considered to be experiencing traffic jams. The observed traffic jams were then used in a refined version of the STOTree Algorithm presented in [110] to generate a series of propagation graphs, where each graph denoted the spatio-temporal causal interactions (or traffic propagation) between roads in the network for a specific date in the observation period.

Looking more towards probabilistic modelling and how it can be applied to predict future behaviour in the network [39, 87]. The work done in [87] considers a *Bayesian network (BN)*, [145] gives a basic introduction into Bayesian Networks, to predict traffic jams on main arterial routes in a metropolitan area. The Bayesian Network was trained on various pieces of information that described the network over a 15 month period, the information includes: (1) past and present traffic jam information, (2) accident reports, (3) weather reports, (4) holiday information e.g., Thanksgiving, Christmas etc., and (5) event information, for a specific time of day and day of the week. The trained BN was then used to compute the probabilities for traffic jam occurrences at specific locations in the metropolitan network on a specific day of week and time of day. In [39], Cheng et al. develop an accurate method of modelling traffic flows, a method that incorporates the full extent of what they call behavioural heterogeneity. Behavioural heterogeneity is the difference in behaviours observed for drivers on the road when selecting a route across the network, where route selection is based on subjective representations of space and estimations of prospective travel times. The approach taken in this work is to use a *Markov Chain*, see chapter D for details, to model and predict behaviour. Cheng et al. consider this approach to capture all relevant behaviour for an improved statistical description of the full heterogeneity of choice amongst a population of individual drivers on the network. Other work on traffic analysis [167] includes methods of interacting with experts in the field in order to gain domain knowledge to develop hierarachical classifiers (see [7] for details) in order to model traffic jams for traffic management control.

The focus of this work is to investigate vehicular traffic in an urban network, a topic that has been heavily studied in the literature [87, 104, 115, 145, 162, 165–167]. However, in this study we consider a GPS-based approach to answer a series of questions that are designed to develop insights into the behaviour of traffic in an urban network. The questions considered are:

1. How can self-similarity be applied to a traffic network in order to develop insights into the inherent communities?
2. What influential factors can be observed to have an impact on the behaviour of traffic in the network?
3. How can probabilistic modelling be used to model behaviour in a traffic network?

The structure of this chapter is described in Figure 4.1, where the roadmap outlines the individual flows of exploration on the GPS-based traffic data, and experiments are conducted using *Python* [142] and its associated packages including *NumPy* [161], *Matplotlib* [89], *SciPy* [96], *Scikit-learn* [133] and *Pandas* [120]. The programme of work is structured as follows. First in Section 4.2 we provide an introduction to the i-MOVE project and its motivation, in Section 4.3 we introduce the i-Move traffic data - describing the processes used in order to extract, wrangle and clean the GPS-based data. This section also describes the characteristics of the traffic data through statistical analysis methods such as those discussed in Chapter 2. Section 4.4 is dedicated to the analysis of the i-MOVE data and this section is split into three sub-sections 4.4.1, 4.4.2, and 4.4.3, each denoting a flow in the roadmap, describing the processes of detecting self-similarity, investigating local changes in the network, and modelling the traffic for a specific road segment. Finally section 4.5 contains a conclusion of this work and discusses potential future work in this area.

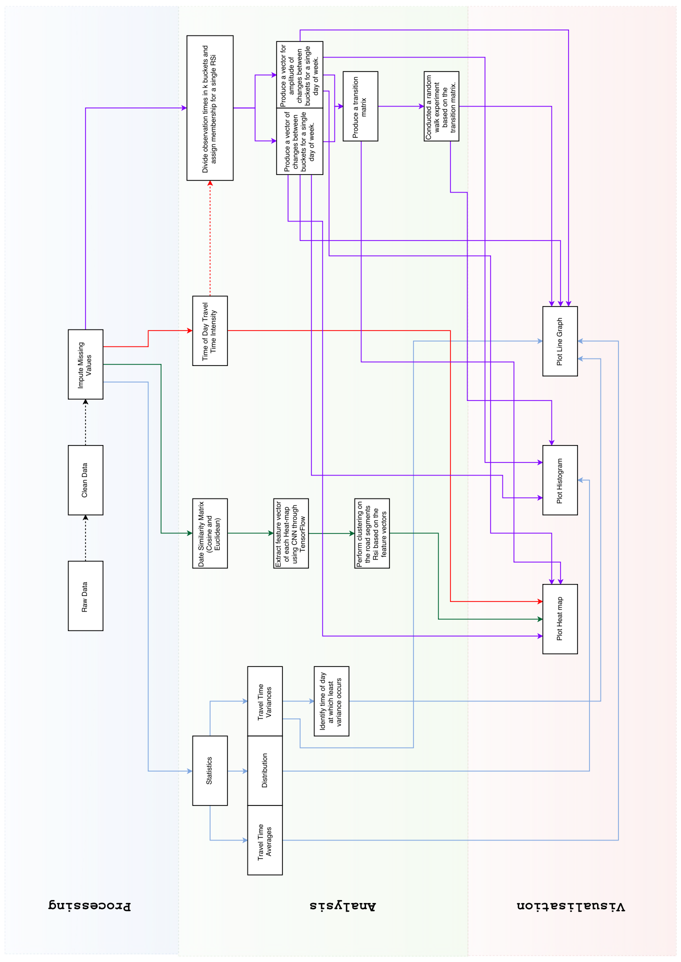

Processing
Analysis
Visualisation
Raw Data
Clean Data
Impute Missing Values
Divide observation times in k buckets and assign membership for a single RSi
Produce a vector for amplitude of changes between buckets for a single day of week.
Produce a vector of changes between buckets for a single day of week.
Produce a transition matrix
Conducted a random walk experiment based on the transition matrix.
Time of Day Travel Time Intensity
Date Similarity Matrix (Cosine and Euclidean)
Extract feature vector of each Heat-map using CNN through TensorFlow
Perform clustering on the road segments Rsi based on the feature vectors
Statistics
Travel Time Variances
Distribution
Travel Time Averages
Identify time of day at which least variance occurs
Plot Line Graph
Plot Histogram
Plot Heat map

Figure 4.1: Chapter Roadmap

4.2 Introducing i-MOVE

The i-MOVE project was part of a group of projects that were funded by Innovate UK [160] to create a digital ecosystem for the Internet of Things (IoT) [100]. The projects focused on specific areas related to IoT describing real-world objects. For example, observing the movement of goods or people throughout the UK using different modes of transportation. i-MOVE focussed on identifying a range of data assets that describe vehicular and railway networks in Liverpool. The motivation behind the project was to identify the impact that the fast development of Liverpool is having on the end users. The city is increasing in scale which is caused by surging populations, expansions to the port etc. Consequently this is having negative impact on congestion in the transportation network. In order to mitigate these issues, city councils must rethink their strategies to design a more sustainable solution of reducing congestion such as bottlenecks in transportation corridors, whilst increasing energy efficiency in urban environments [98]. The goal of the project was to design an interoperable platform (or *hub*) that can expose a multitude of data assets that describe traffic, incidents [91, 150], weather [122] etc., to a developer community. Exposing the data assets to a developer community means that experiments or applications can be created that will provide insight into the impact that traffic congestion has on the transportation corridors. The interoperability requirement of the platform was a requirement for each of the projects funded by Innovate UK, and as part of this requirement an IoT platform interoperability standard was developed, *HyperCat* [125]. HyperCat provided a universal protocol for interoperability amongst platforms, that is, it allowed platforms to expose the data assets they contained with all other platforms.

In this study we concentrate our interests on one particular data feed (or source) - a feed provided by INRIX [91] - that in our opinion was the most complete and reliable data set that provides intelligence on the network. In the next section of this work we discuss the data set in detail.

4.3 i-MOVE Traffic Data

Traffic can be observed using two distinct methods 1) Trajectory data (TD), and 2) Floating-car data (FCD). Trajectory data is where traffic is observed directly by cameras on top of tall buildings, or mounted on airplanes. The information is tracked, and trajectories are extracted i.e. positions of each vehicle α over time δ. In [158], the authors suggest that if all of the vehicles on a given road segment (and time span) are captured using this method, the result data is called trajectory data. Floating-car data uses probe vehicles which *float* in the traffic flow, such vehicles collect geo-referenced data via Global Positioning System (GPS) receivers which are then *map-matched* to the physical road-map, and the speed is a derived quantity determined from the spacing between two GPS points for a road segment. In this work we consider floating-car data that is extracted from a small quantity of vehicles on a specific road segment. In particular, we consider data aggregated by **INRIX**; a data provider for vehicular traffic systems, where the data describes the traffic flow for a multitude of road segments in and around Liverpool City Centre. The data is aggregated using GPS based systems, and Automatic Number Plate Recognition (ANPR) vehicle recognition systems, and is *map-matched* to the network blueprint in order to identify the average travel time for a vehicle on a specific road segment at a specific time δ. The raw data supplied by INRIX was aggregated for the i-MOVE project (mentioned above).

timestamp	**code**	**speed**	**travel time minutes**
2013-10-14 11:22:36	C10P12345	13	0.037
2013-10-14 11:22:36	C10-12345	13	0.017
2013-10-14 11:22:36	C10+48845	13	1.003
2013-10-14 11:22:36	C10-48845	13	2.201
2013-10-14 11:22:36	C10P48945	13	5.000
2013-10-14 11:22:36	C10N48945	13	3.001

Table 4.1: Floating-car XML link object elements

4.3.1 Data Preparation

In its raw form the floating-car data was provided as a sequence of records, where each record describes a pair of reference points. Specifically, a single record is represented by an XML object that comprises four attributes 1) *timestamp*; date and time of the observation, 2) *speed*; estimated space mean speed for the roadway segment in miles per hour (MPH), 3) *travel time minutes*; current estimate time it takes to traverse between the two reference points in the network, and 4) *code*; the unique identification for a specific pair of reference points, Table 4.1 provides examples and Section 4.3.1 provides explanations. In an online setting it can be observed that a procedure to identify records for a specific pair of reference points that describes a link (Lnk_i) would not be simple. In particular, it would have to include some clustering mechanism that will identify a group of records, and classify them as belonging to the same group if their code attribute is the same. Furthermore, the procedure would have to include some component for code-location translations, specifically it would have to perform location translations using a Traffic Message Channel (TMC) translation table, in order to provide feedback on the spatial relevance of the data.

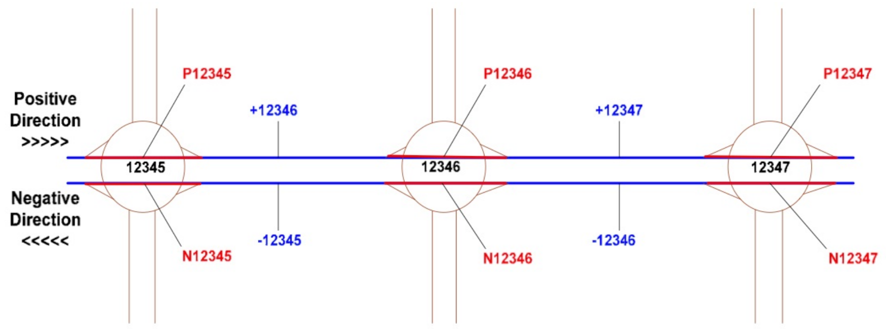


Figure 4.2: Relationship between RDS-TMC codes and GIS Road Links

Data Wrangling

In our work we create a data wrangling process for transforming the raw (crude) floating-car data into a clean and usable format. This is part of a known process in *data warehousing* called Extract Transform Load (ETL) [163]. The transformation phase in the process is performed on a specific record using the *code* attribute - a specific code is used to determine location information (e.g., longitude, latitude, road name etc.) from the TMC table. In its raw form, the code attribute was provided as two different types: 1) ***C10P12345***; representing large junctions in the network e.g., roundabouts, and 2) ***C10+12345***; representing road segments. Figure 4.2 provides an illustration of the two types of code forms. The code is segmented into sections, the first 3 characters can be discarded as they are always the same, the next character denotes the direction in which the sensor is measuring travel times, and the last 5 characters represent the identity of the road segment (used to identify a road link in our new dataset). The measuring direction can be referenced using either letters or symbols to denote the direction e.g., P and $+$ represent positive, and N and - represent negative. Intuitively if a sensor s_i is measuring in a positive direction the next reference point would be s_{i+1} and similarly if the sensor is measuring in a negative direction the next reference point would be s_{i-1}. Therefore, for a sensor detecting travel times in a position direction from reference point *12345* that code would be denoted as either *C10P12345* or *C10+12345.* This understanding can be applied to the road segment identification procedure (detailed below), in order to determine the unique reference points that a single record describes.

Transformation Procedure

In this section we discuss the transformation procedure to extract meaningful information from the input data to present a simple and usable data set. The transformation procedure of a single record is detailed below:

1. Extract the code attribute of the record and split the code at position 3, in order to derive an array of sub-strings

2. Discard the first item of the array as per detailed in the data wrangling process, this information is consistent and does not provide any additional information.

3. Using the second item of the array, extract the first character, this is the direction of measurement denoted d_i, and the remaining part of the substring is the initial reference point id that the record describes and this is denoted by p_s.

4. Using the reference point id p_s and the direction of measurement d_i, the end reference point can be derived and this is denoted by p_e. If $d_i = P$ or $d_i = +$ then $p_e = (p_s + 1)$, or if $d_i = N$ or $d_i = -$ then $p_e = (p_s - 1)$.

The outputs of the procedure p_s and p_e are used to perform location translations with the TMC table in order to identify location information e.g., geographical coordinates, road name, and crossing name, for a specific record. In an object-oriented environment the new information is used to create

Element	Description
from	Object that describes the geographical location of the first reference point in a record using World Geodetic System 1984 (WGS84).
to	Object that describes the geographical location of the second reference point in a record point using World Geodetic System 1984 (WGS84).
crossing	Textual description of the road name that both reference points intersect.
time	Date and time the observation was recorded represented in ISO 8601 format.
value	Decimal value that represents the travel time in minutes
unit	Unit of measurement that was used to record the travel time.

Table 4.2: Floating-car records in raw form

a *Lnk* object, where the object contains information about geographical locations of the start and end points, time of measurement, and the measurement value. At the point the transformation procedure has completed the list of records is now a list of Link objects. Subsequently, the Link objects are grouped together such that reference points following each other consecutively (in a positive or negative direction) are assigned to the same group, where the group can be considered to describe the same road or section of the network. Furthermore, the grouped objects (denoting the records) are sorted temporally, that is, they satisfy the following condition $Lnk_i^{t_i} < Lnk_i^{t_{i+1}} < \cdots < Lnk_i^{t_{m-1}} < Lnk_i^{t_m}$, where t_i is time that the measurement value was observed, and m is the last observed time in the observation period. In order to allow the data to be accessible for a range of applications, the data is converted into a series of Extensible Markup Language (XML) files, where a single XML file describes the observations extracted across the network of links for single time of day. Table 4.2 gives a tabular representation of the output data.

4.3.2 Data Cleaning

A common practice in any data mining task is to clean the data. Cleaning the data is a method of pre-processing the input data to detect, remove and correct inconsistencies in order to improve the data quality. Data quality issues commonly arise in data collections, and are generally caused by data entry mistakes, missing values or invalid data. Data correctness is required in data mining tasks in order to avoid wrong conclusions being made on the object or system being examined. Data cleansing is an important component of ETL. Data is typically extracted from the information source such as sensors, actuators etc., cleaned using a method such as that mentioned below before being loaded into a data warehouse such as Hadoop Distributed File System (HDFS) , Oracle data store etc, for further analysis. According to [138] data cleansing techniques should satisfy several requirements, these include but are not limited to: 1) ability to detect and remove all major errors and inconsistencies in a data source, 2) be supported by existing tools to limit manual inspection and programming effort, and 3) scalable and reusable for a multitude of data assets that may be different in size or content.

Prior to the data cleaning process we must ingest the transformed data, and this is achieved through parsing each of the previously created XML files using Python [137] built-in XML handling libraries. The data records now stored as XML objects in a list are then grouped by the link (or pair of reference points) that they describe. This creates a matrix of observations M, where each cell in the matrix $M = [i, j]$ is an observation made on a specific road segment indexed by i, and the observation denotes travel time measurements extracted on a specific date at a specific time of day. Following this, a single row $M[i, :]$ is sub-divided into m lists of objects - each describing the observations made for a specific date in the observation period, where the number of lists equates to the number of dates in the observation period which in this instance is 85; an explanation of this quantity can be found in Section 4.3.3, and each list L is sorted temporally.

Missing Entries

Description: Missing entries are common in data extraction procedures where data collection methods are imperfect. For example, user surveys are often unable to collect responses to all questions. In such cases three common techniques are used to handle missing entries (1) eliminate records that contain missing values (2) missing values maybe be imputed, or (3) design analytical methods to work with missing values - often the most commonly used in data mining tasks as most methods in the literature work robustly with missing values [3]. It is obvious that all three methods have issues: (1) is not practical in the case that most of the records contain missing values, (2) could add bias inherent in the imputation process or imputing based on the wrong feature of the data, and (3) would add extra complexity to the design of analytical processes. In this work we consider an intuitive process that is based on option (2) - a simple but effective method of cleaning the original data. In particular the method identifies missing entries and imputes them using an established statistical technique. In this method for a single list L describing a specific date d_i and link Lnk_i, we identify what we call *legal observations*.

Definition 4.3.1. A legal observation is an observation that is measured within some threshold ϵ from the minute marker. In the case that an observation is beyond ϵ seconds of the closest minute in a positive or negative direction, we consider it to be a missing value; in order maintain consistency across the links in the network and dates in the observation period, and we denote this by 0.

This approach was motivated by [3], where the Aggarwal specifies that it is often convenient to have time series values that are equally spaced and synchronised across different behavioural attributes for data processing. It was method that is designed to create a synchronised time series (see Definition 4.3.2 for details), that is equally spaced; observed every 5 minutes, and contains no missing entries. Therefore a single list L is described by a vector of 288 elements.

Definition 4.3.2. A time series denoted as $S = X(t), t \in T$ is a set of statistics that represent an observation of a stochastic process, where T is the set of chronological time points (or indexes) at which measurements are made, and $X(t)$ is the observation made at time t. The observations $S = X(1), X(2), \ldots, X(n)$ are typically represented as a sequence of real numbers R, where $X(1)$ denotes the value taken by the series at the first time point, $X(2)$ denotes the value for the second time point, $X(3)$ denotes the value for the third time point, and so on. It is common for the observations to be extracted at consistent and regular time intervals across the specified period of observation e.g., hour, day, week, year etc. A subsequence $S_{(i,k)} = X(t_i), X(t_{i+1}), \ldots, X(t_k)$ of time series S is a shorter time series of length k starting from position i.

Method: The method used to parse legal observations and handle missing entries is described by two successive algorithms. Algorithm 4, called parse-Legal, is used to parse out legal entries and create a synchronised set of time series'. The procedure starts between lines (2) and (4) where the variables are initialised, and the first time point is identified. Lines (5) to (19) is a *For* loop that is the main part of the algorithm. Lines (6) and (7) compute the time delta (or difference) in minutes and seconds respectively between the previous and current times of day. Lines (10) and (14) are then used to identify legal observations in the data. In the case that a legal observation is identified, a tuple is added to the output vector denoted by $(time, value)$. At the point the procedure terminates at line (20), the output vector *legal* is returned. Algorithm 5, called zeroPad, is a successive step in the method that inserts place-holders in the vector returned from Algorithm 4 at points in the time series where observations are missing. The goal of this algorithm is to ensure that the time series' are synchronised regardless of the quantity of missing values. The procedure starts between lines (2) and (3) where the variables are initialised. Line (4) checks if a potential missing value exists at the start of the day. In the case that the missing entry criteria is satisfied a single 0 is added to the output vector. Lines (7) to (18) is a *For* loop that is the main part of the algorithm. At line (8) the current time point is identified, which is then used in line (9) to calculate time elapsed since the last time point. In line (10) the number of intervals; an interval is 5 minutes in length, elapsed since the last time point is computed, line (11) checks if this quantity is more than 0, and if this is the case the *For* loop between lines (12) and (14) add that quantity of 0 to the vector. Following this, line (17) adds the current time points' observation to the output vector. In line (22) a check is performed to see if missing values occur at the end of the day, and in the case that the missing entry criteria is satisfied a single 0 is added to the end of the vector. Finally, in line (24) the output vector is returned *complete*. The output of second algorithm creates a consistent and equally spaced time series for a single date and link in the observation period.

Algorithm 4 Parse Legal

Input: L: a list of objects describing observations for a specific date d_i; ϵ: threshold to identify legal observations

Output: *legal*: a vector comprising legal observations for link Lnk_i and date d_i

```
1: procedure parseLegal($L$, $\epsilon$)
2:     $legal \leftarrow []$
3:     $prev \leftarrow L[0].time$   ▷ Set prev to the first extraction time in seconds
4:     $legal[1] \leftarrow prev$
5:     for $t_i$ in $\{1, \ldots, Length(L)\}$ do
6:         $current \leftarrow L[t_i].time$ ▷ Set current to the current extraction time in seconds
7:         $\Delta T_m \leftarrow \frac{(current - prev)}{60}$   ▷ Time delta in minutes
8:         $\Delta T_s \leftarrow (current - prev) \bmod 60$   ▷ Time delta in seconds
9:         if $(\Delta T_m \bmod 5) = 4$ then  ▷ Approaching the time from the left
10:            if $(60 - \Delta T_s) \leq \epsilon$ then
11:                $legal[t_i] \leftarrow (current, L[t_i])$
12:            end if
13:        else if $(\Delta T_m \bmod 5) = 0$ then ▷ Approaching the time from the left
14:            if $\Delta T_s \leq \epsilon$ then
15:                $legal[t_i] \leftarrow (current, L[t_i])$
16:            end if
17:        end if
18:        $prev \leftarrow current$
19:    end for
20:    return $legal$
21: end procedure
```

Algorithm 5 zeroPad: function to zero pad the missing values for a specific date

Input: *legal*: a vector comprising legal observations for link Lnk_i and date d_i
Output: *complete*: a vector comprising observations and place holders for missing values for Lnk_i and date d_i

```
1: procedure zeroPad($legal$, $\epsilon$)
2:     $complete \leftarrow [1, 2, \ldots, 288]$
3:     $prev \leftarrow legal[0]$    ▷ Set prev to the first legal extraction time in seconds
4:     if $\frac{(prev/60)}{5} > 0$ then    ▷ Check for missing observations at start of the day
5:         $complete[0] \leftarrow 0$
6:     end if
7:     for $t_i$ in $\{1, \ldots, Length(legal)\}$ do
8:         $current \leftarrow legal[t_i][0]$
9:         $\Delta T_m \leftarrow ((current - prev)/60)$
10:        $window \leftarrow (\frac{\Delta T_m}{5}) - 1$
11:        if $\#window > 0$ then    ▷ Add missing place-holders for the time window
12:            for $w_i$ in $\{t_i, \ldots, t_i + window\}$ do
13:                $complete[w_i] \leftarrow 0$
14:            end for
15:        end if
16:        $complete[w_i] \leftarrow legal[t_i][1]$    ▷ Add legal observation
17:        $prev \leftarrow current$
18:    end for
19:    $last \leftarrow \frac{legal[Length(legal)][0]}{60}$
20:    if $(60 - last) > 5$ then    ▷ Check for missing value at the end of the day
21:        $complete[Length(legal)] \leftarrow 0$
22:    end if
23:    return $complete$
24: end procedure
```

Imputation

Description: It is common for a time series to contain missing values [3], where the missing values may be caused by factors of randomness, limitations of measuring equipment etc. Recall in the previous step the data was pre-processed to create consistent and equal size vectors for each of the dates and links in the network. In this section we consider a method of the handling missing data, that is, missing observations that were identified in Section 4.3.2. In [3], it is stated that there exists three classes of techniques for handling missing entries (1) eliminate missing entries, (2) impute or estimate missing entries, and (3) design the analytical method to work with missing values. The first method is unrealistic as this would mean inconsistent time series' are created, the second method seems the most plausible because it would create consistent time series', and would not require major changes in future processing functions, whereas the third option would. Based on this, we consider the second option to be the most appropriate, and design a method to impute the missing values with the arithmetical mean of the vector that contains similar observations in the data, that is, observations on the same link and time of day. The procedure for imputing the missing values is described next.

Procedure: The procedure that concludes the data cleaning process and imputes the missing values is presented in Algorithm 6. The *For* loop located between lines (1) and (12) is the main part of the algorithm. Line (2) is used to check for missing values, if no missing value exists move onto the next iteration of the for loop. In the case that a missing value place-holder exists lines (5) to (8) are used to build a vector of observations that are similar to the current observation at index t_i. Once the vector has been populated line (9) computes the arithmetical mean of the vector in order to estimate the missing observation using

$$f(t_i, D) = \frac{1}{N} \sum_{d_j \in D}^{|D|} d_i[t_i] \tag{4.1}$$

,where t_i is the current time point in the day, D is the vector of similar observations created between lines (5) and (8), and d_j is the vector of observations for a specific date j. Line (10) uses this computed value e_j to replace the place-holder in the vector. Finally the procedure concludes at line (13) where the modified vector is returned.

Algorithm 6 imputeMissing: function to impute the missing values for a specific date

Input: *complete*: a vector comprising observations and place holders for missing values for Lnk_i and date d_i; X: a list of vectors comprising the time series for each date in the observation period
Output: *complete*: a modified vector comprising observations and imputed missing values for Lnk_i and date d_i

```
1:  for $t_i$ in $\{1, \ldots, Length(complete)\}$ do
2:      if $complete[t_i] > 0$ then
3:          Continue;
4:      else if $complete[t_i] = 0$ then          ▷ build the vector of similar observations
5:          $D \leftarrow []$
6:          for $d_i$ in $\{1, \ldots, Length(X)\}$ do
7:              $D[d_i] \leftarrow X[d_i][t_i]$
8:          end for
9:          $e_i \leftarrow f(t_i, D)$     ▷ compute the arithmetical mean using Equation 4.1
10:         $complete[t_i] \leftarrow e_i$
11:     end if
12: end for
13: return complete
```

Conclusion

In conclusion of this section of work we have identified a method for cleaning the floating-car data. In particular, the cleaning procedure includes four successive operations: 1) parsing the floating-car data to extract observations that satisfy the criteria of being legal, 2) identifying missing observations for a specific date, 3) zero-padded the output vector to include place-holders that signify missing observations, and 4) performed imputation on the output vector to replace missing observation place-holders with substituted values. The data cleaning procedure was performed on each date in the observation period for all road segments in the network denoted Lnk_i, and this produced a list of vectors denoted $L_{Lnk_i} = [\vec{v_1}, \vec{v_2}, \cdots, \vec{v_{85}}]$, where each vector $\vec{v_i}$ is a time series of observations for a particular date in the period of investigation.

4.3.3 Data characteristics

In this section of work the cleaned floating-car data is described to provide the reader with insight into the characteristics and limitations of the traffic data. Furthermore, this section describes the statistical properties of the data that can be used in further analysis in order to characterise road segments in Liverpool based on their traffic flow profiles.

Description of input data

The data represents observations that are captured across the network by many randomly distributed floating cars, and a single datum denotes the travel time for a single floating car to traverse between two reference points (or road segment) in the network. Recall in Section 4.3.2 that the data was transformed and cleaned into a more usable and consistent data set. This create a set of time series' for a single road segment Lnk_i that is of length equal to the quantity of dates in the observation period, or in this work 85, which is the number of dates between 29th November 2013 and 21st February 2016. In this set a single time series describes a date d_i which is represented by a vector of size 288, where each element x is a real number denoting a measurement of travel time between two reference points (or along a road segment) at a time point, or 5 minute interval, in a 24 hour period.

Data Statistics

Statistics are used to provide useful and accessible information about a specific object [169]. In this work we use statistics to develop insights into the behaviour (or patterns of behaviour) of traffic in the network using the floating-car data. A common method of understanding the behaviour of a random variable is to compute its frequency distribution. A frequency distribution provides insight into the likelihood that a random variable will take on a given value, and thus provides insight into the pattern of the data over an specified observation period. Recall from Section 2.1 that the type of method of statistical analysis is dependent on the type of variable being examined. Based on this, and the fact that the floating car data is numerically valued, we must consider a method that is effective for analysing quantitative data. However, the floating car data can include a multitude of values that measure the observed travel times on a specific road segment Lnk_i. In order to compute the frequency distribution of such data, the observations must be grouped. A method of grouping observations is to consider class intervals, that is, intervals of equal size that cover the range between the minimal and maximal observed values without overlapping. The intervals are then used to compute the frequency of observed values, thus creating a frequency distribution, and this is represented by a histogram.

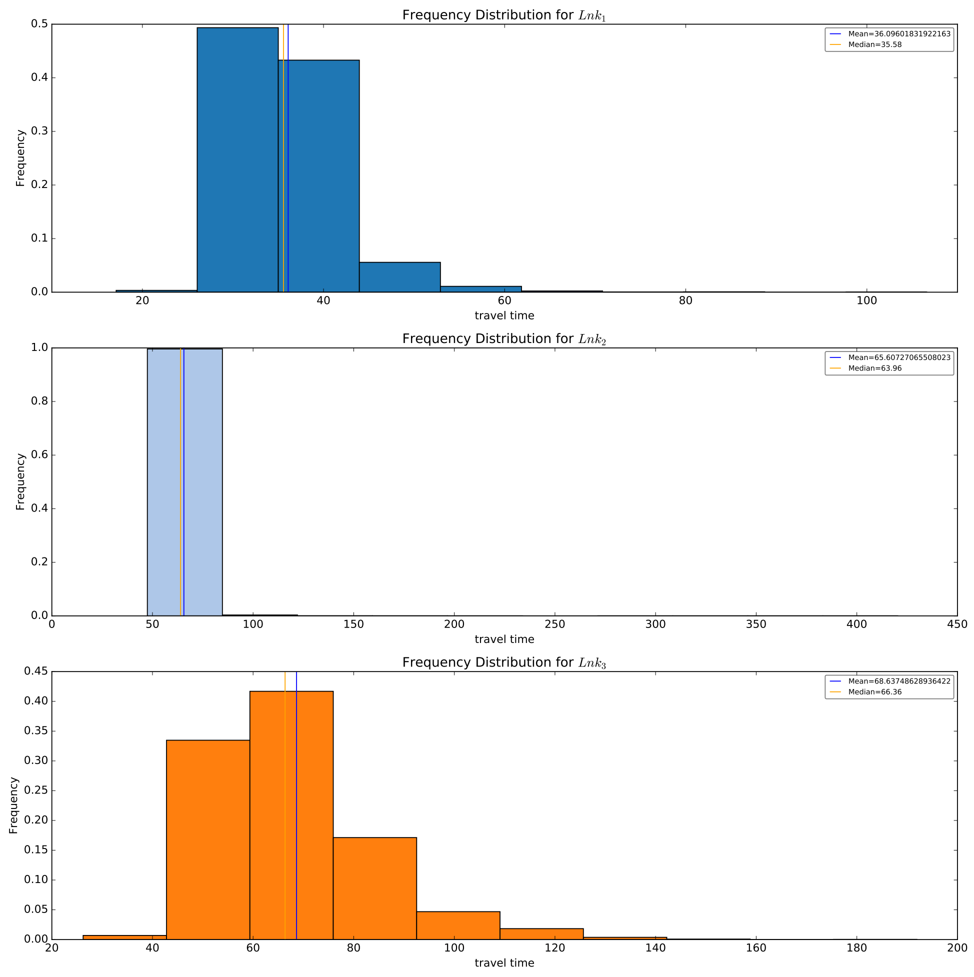


Figure 4.3: Travel Time Distribution

In this work we compute the frequency distributions of observations for three sample road segments Lnk_1, Lnk_2 and Lnk_3, and represent these as histograms in Figure 4.3. The resulting histograms can be categorised as normally distributed, that is, the distribution is centered about its mean, meaning that the majority of observations reside in the middle of the distribution and a very few observations lie in the tails of the distribution [76, 94]. This result aligns with the result presented in [30], where Berry et al. demonstrate that travel times of vehicles are normally distributed. The distribution only provides a summary statistic of the traffic data, it gives a global insight into the typical behaviour of a specific road segment in the traffic network. In order to gain further understanding of the traffic behaviour one must investigate other statistics or measures such as averages. The average (or mean) is a measure of central tendency, that is, it is a descriptive measure that indicates the central point in a given distribution of values, where the central point is the typically observed value for a random variable. In this work we compute the arithmetical mean for each of the sample road segments and each time point (or interval in the day), and represent this in Figure 4.4. The mean is computed for specific time point t_i and road segment Lnk_i using,

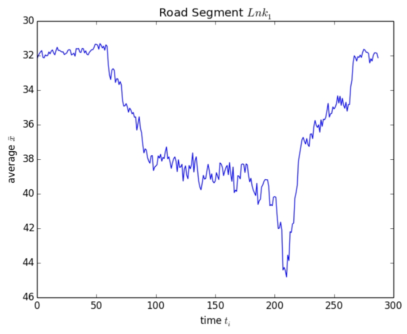
$$\bar{X_{t_i}} = \frac{1}{|D|} \sum_{d_j \in D}^{|D|} d_i[t_i] \tag{4.2}$$

where D is the set of dates in the observation period, and d_j is a vector of observations for a specific date. Reviewing the plots in Figure 4.4, and in particular analysing the shapes of the curves suggests (as expected) that each road has a different traffic flow behaviour. This expectation is based on the fact that the sample road segments are located at different points in the network that are not directly connected. However, similarities does arise between the behaviours, and in particular these similarities are troughs, dips in the curve, that appear at 8:30am and 5pm. This similarity aligns with the expectations that these points in the day are rush hour periods.

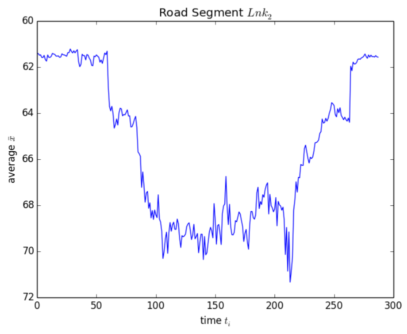
In the literature on descriptive analysis for a data set it is suggested that identifying the measure of central tendency is not enough to distinguish between two distributions (see Chapter 1 for more details). In statistics a measure of variability, determining how scattered a distribution is away from its center (mean or average) or clustered towards it, is used to provide insight into the structure of data, and thus be used to differentiate two distributions. In this work we compute the variance for each of the sample road segments and each time point (or interval in the day), and represent this in Figure 4.5. The variance will be used to determine the variability amongst travel times in order to discover how regular (or irregular) traffic flow is on specific road segments in the network. For periods in the day at which the least variance is observed, it would be suggested that the traffic flow is consistent. In the complementary case, periods at which the highest variance is observed, it would be suggested that traffic flow is irregular, sporadic, or volatile, meaning that traffic volumes frequently fluctuate. The variance is computed for specific time point t_i and road segment Lnk_i using

$$\sigma_{t_i} = \sum_{d_j \in D}^{|D|} (d_i[t_i] - \bar{X_{t_i}})^2 \tag{4.3}$$

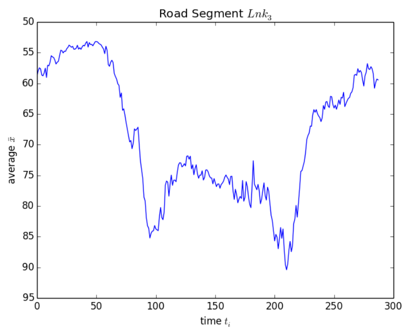
,where D is the set of dates in the observation period, d_j is a vector of observations for a specific date, and $\bar{X_{t_i}}$ is computed using Equation 4.2. Reviewing the plots in Figure 4.5, and in particular the curves, it is apparent that troughs appear at rush hour periods in the day, similar to those for the averages plots. A further observation in the plots is a large trough that appears at 5pm (or position 200 in the plots), this would suggest that this period in the day is the most difficult to estimate for each of same road segments. It would be difficult to identify this characteristic of the travel time data based on the averages alone. For example, the curve that denotes the averages for the road segment identified by Lnk_2 a trough is apparent at roughly position 200, but the significance of the trough is not obvious. However, using the variances we can identify how significant the trough is.



(a)

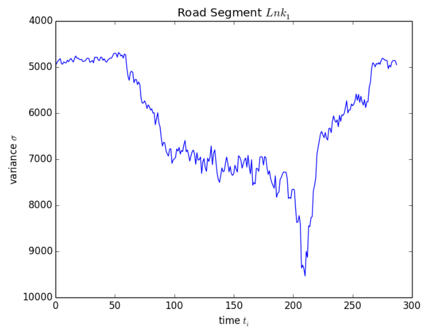


(b)

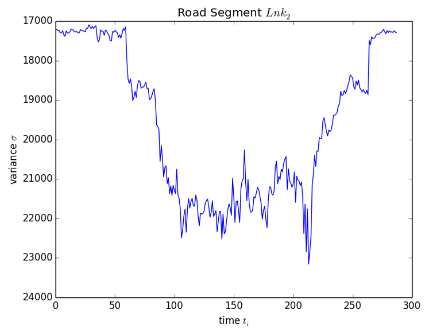


(c)

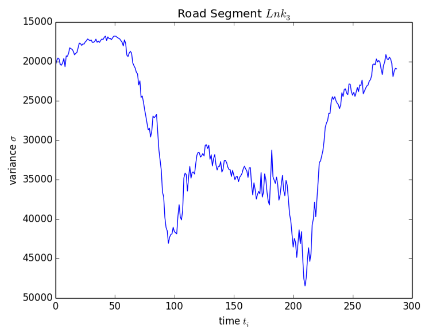
Figure 4.4: Plots of travel time intensity averages for three road segments, where the x axis denotes the time of day and the y axis denotes the average travel time observed



(a)



(b)



(c)

Figure 4.5: Plots of travel time intensity averages for three road segments, where the x axis denotes the time of day and the y axis denotes the variance of the observed travel times

4.4 Analysis

The primary focus of this chapter is to develop understanding of *urban traffic* networks. Recall in Section 2.2, and the work in Chapter 3 that a method of developing understanding of networks is through community detection, that is, discovering a division of the network within which the nodes connect more to each other than any other nodes in the network. In this study we consider a different approach, an approach that is based on the original method of understanding networks; examining the properties and characteristics of the individual network entities using the data that describes them. In particular, we consider a data-driven approach, and propose a series of experiments that are designed to answer the questions presented in Section 4.1. We consider four experiments in this study, each of which are conducted using each of the sample road segments mentioned in 4.3.3. First, in Section 4.4.1, we examine the self-similarity of each road segment and based on this we perform clustering to classify road segments. Second in Section 4.4.2 we examine the intensity of observed travel times for a specific road segment in order to gain appreciation of the typical (or regular) traffic flow at a specific time of day, and the influence that time of day has on the experienced travel time. Third in Section 4.4.3, we model and simulate the traffic flow by applying the floating-car data to a Markov Chain. The structure of this work is centered around the experiments, and the structure of each experiment is as follows. First, *description* provides details of the experiment and relevant background. Second, *procedure* provides a description of the procedure used to conduct the experiment. Third, *results* details the results of the experiment. Four, *key findings*, provides a summary of the experiment and the results obtained.

In order to yield meaningful comparisons between pairs of time series' both must be normalised [139]. This transforms the data into a consistent scale, that is, each of the time series' denoting the travel time observations for a specific date d_i and road segment Lnk_i appear in the same scale. The scale we transform the data into is determined by the inherent statistical properties of the data, which is based on the Z-score (see Definition 4.4.1), determining how far away from the expected (mean denoted μ) value the datum x is.

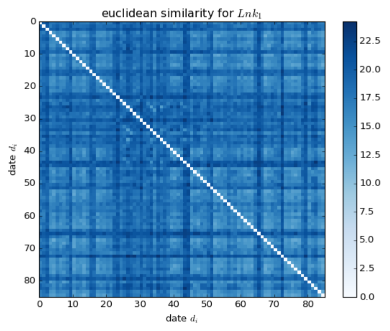
Definition 4.4.1. **Z-score** or commonly regarded as Z-number, denoted z. For any value x, z is equal to $(x - \mu)/\sigma$, where μ and σ are the population mean, and the population standard deviation respectively. The Z-score gives an indication of how far x is away from μ in units of standard deviation, so z is negative when $x < \mu$ and z is positive when $x > \mu$.

4.4.1 Self-similarity

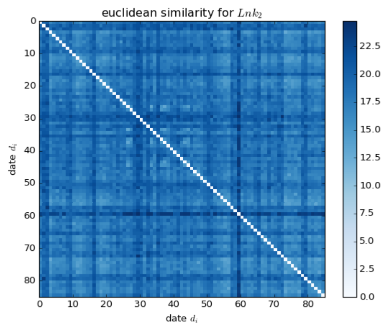
Description: The focus of this experiment is to investigate Question 1, that is, we investigate self-similarity in the traffic network with a view to discover inherent communities of road segments. The work in this section is motivated by similar studies [27, 48, 106] that are concerned with discovering self-similarity in networks, considering World Wide Web (WWW) and Ethernet Local Area Network (LAN). In [106], Leland et al. demonstrate that Ethernet LAN traffic is statistically self-similar, and that none of the commonly used methods of traffic modelling such as Poisson are able to capture this behaviour, and [27] considers that using such methods presents unrealistic results. In order to investigate self-similarity in the traffic network, we compute pair-wise comparisons between all pairs of vectors denoted by $\vec{v_i}$ and $\vec{v_j}$, where a single vector $\vec{v_i}$ comprises a series of observations for a specific date d_i in the observation period, meaning the size of the vector is

288, or equal to the number of 5 minute intervals in a 24 hour period, and the pair-wise comparison is computed using two distance (or similarity) measures, *Euclidean Distance* and *Cosine Similarity*, see Appendix A for details, where the chosen measures are common methods of discovering similarity in data [37, 93, 139]. The results of the comparisons are represented in a matrix of similarity denoted $S := (s_{i,j})_{n \times n}$, where n denotes the quantity of dates in the observation data, and a single cell $s_{i,j}$ is a real number that represents the similarity between a pair of dates i and j. The self-similarity is performed for a sample set of road segments in network, and the resulting matrices are represented by heat maps in Figures 4.6 and 4.7.

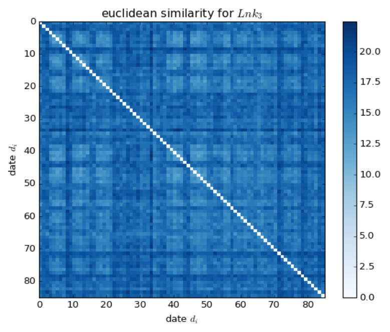
Results: Firstly, it is immediate to observe that the patterns in the heat maps are consistent across the two measures for each of the road segments used in our analysis. This consistency means that the value denoting the pair-wise similarity between pairs of dates resides at a similar position in each of a scales for the two different distance measures. This observation suggests that it does not matter which measure we choose in our analysis, it will always yield the same pattern in the resulting heat map. A further observation on the heat maps is that self-similarity does not follow any defined pattern, that is, self-similar regions denoted by patches of light (or dark) colours, dependent on the measure, are randomly distributed. However, what does appear to be consistent across the Heat maps is that the dates at which high similarity is observed are identified to be weekends.



(a)

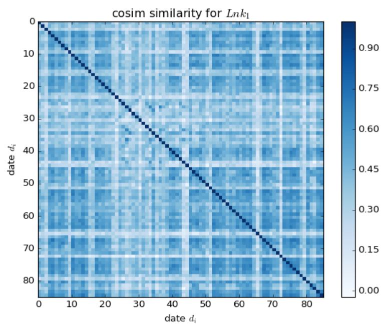


(b)

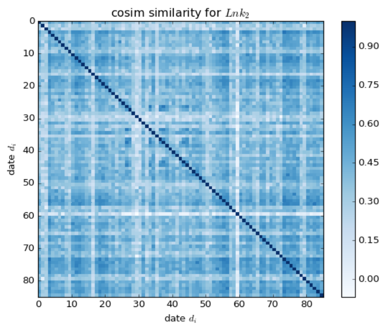


(c)

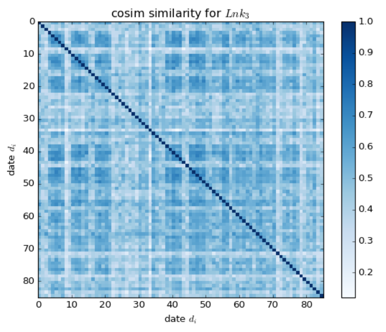
Figure 4.6: Euclidean similarity matrices for three road segments



(a)



(b)



(c)

Figure 4.7: Cosine similarity matrices for three road segments

4.4.2 Travel time intensity

Description: The focus of this experiment is to investigate Question 2, that is, we investigate if a temporal relationship exists between the travel time and the time of day at which is was observed. The work in this section is motivated by [87], where Horvitz et al. consider a Bayesian Network that was trained on information describing the network in order to predict traffic jam occurrences on specific points in the network, where predictions are based on the day of the week and time of day. In order to gain an appreciation of this relationship, we compute the frequency distribution of travel times observed for each day of the week and time of day across the 12 week observation period; a period that is more than 3 times of that convered in [165], and produce a matrix of frequency distributions for a single road segment Lnk_i denoted as $M := (m_{i,j})_{n \times m}$, where n is the number of unique travel times, m is the number of 5 minutes intervals in a 24 hour period; 288. A single cell in the matrix denoted as $m_{i,j}$ represents the number of times travel time i is experienced at time of day j across the observation period. The following procedure is used to produce this matrix for each of the sample road segments, and the resulting matrices are represented as heat maps in Figure 4.8.

Procedure: Algorithm 7 was used to compute the matrix of frequency distributions for a specific road segment Lnk_i. The algorithm is described as follows. Lines (1) to (9) initialise the set of travel times denoted as A and populates this set with observations on Lnk_i. Line(10) initialises the frequency distribution matrix denoted as M. Lines (12) to (18) is a *For* loop that forms the main part of the algorithm. Line (12) identifies the next time point in the day denoted as t_i, line (13) identifies the next date in the observation period denoted as d_i, line (14) identifies the index of the current observation denoted as x, line (15) identifies the index of x_i in A, and line (16) increments the corresponding cell value in M, that is, m_{t_i,x_i}. Finally the procedure terminates at line (17) where the frequency distribution matrix is returned.

Algorithm 7 frequencyDistribution: function to compute the frequency distribution matrix for a specific road segment in the network

Input: D: a list of vectors of observations, where each vector is a the observations for a date d_i
Output: M: a matrix of frequencies distributions

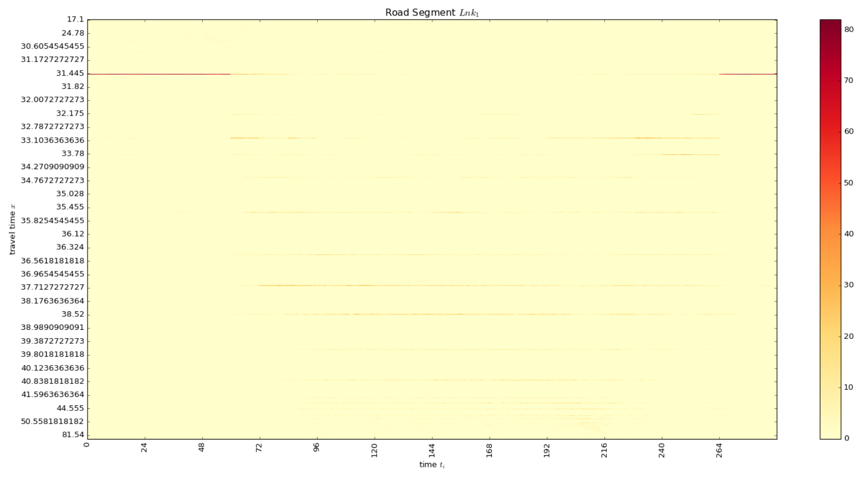
```
1:  $A \leftarrow \emptyset$                                   ▷ initialise the set of travel times
2:  for $t_i$ in $\{1, \dots, 288\}$ do
3:      for $d_i$ in $D$ do
4:          $x \leftarrow d_i[t_i]$
5:          if $x \notin A$ then ▷ check if the travel time is not already in the set
6:              $x \cup A$
7:          end if
8:      end for
9:  end for
10: $M \leftarrow m_{1,1} = 0, m_{1,2} = 0, \dots m_{m,n} = 0$      ▷ initialise the $m$ by $n$ matrix
11: $\vec{dist_i} \leftarrow []$
12: for $t_i$ in $\{1, \dots, 288\}$ do
13:     for $d_i$ in $D$ do
14:         $obs \leftarrow d_i[t_i]$
15:         $x_i \leftarrow A.index(obs)$          ▷ get the index of the observation in $A$
16:         $m_{t_i,x_i} \leftarrow m_{t_i,x_i} + 1$               ▷ update the matrix cell value
17:     end for
18: end for
19: return $M$
```

Results: In review of the Heat maps in Figure 4.8, it is immediate to notice that for each of the sample road segments the frequency that a specific travel time x is observed fluctuates as the time of day progresses. This can be noticed by observing that the colour gets closer to red as we approach periods of the day at which travel times were observed frequently, suggesting that these are periods of the day at which traffic is stable. In the complementary case, we notice that colour of the cells in the Heat map change between yellow and white for periods of the day at which travel times were observed infrequently, suggesting that these are periods of the day at which traffic is sporadic (or irregular). However, it is difficult to identify patterns in the heat maps due to the domination of stable traffic. In order to manage this issues, we rescale the heat maps such that the colour spectrum only represents travel times with frequencies that are below a specified threshold θ, and cell with a value that is beyond θ is coloured black. In our work, we consider $\theta = 5$, this is the corresponding value of the colour that dominates the heat maps, and therefore if we consider this as a threshold value for rescaling the heat maps we can gain exposure to the patterns that occur between infrequently observed travel times. The rescaled heat maps are presented in Figure 4.9. Reviewing the rescaled Heat maps shows that patterns do exist, patterns that were obscured by the stable traffic, and these patterns suggest that at position $t_i = 216$, the traffic appears to be the most irregular. This observation aligns with the troughs observed in both the intensity averages plots in Figure 4.4 and the intensity variances plots 4.4, where the troughs are more apparent in the variances plots.

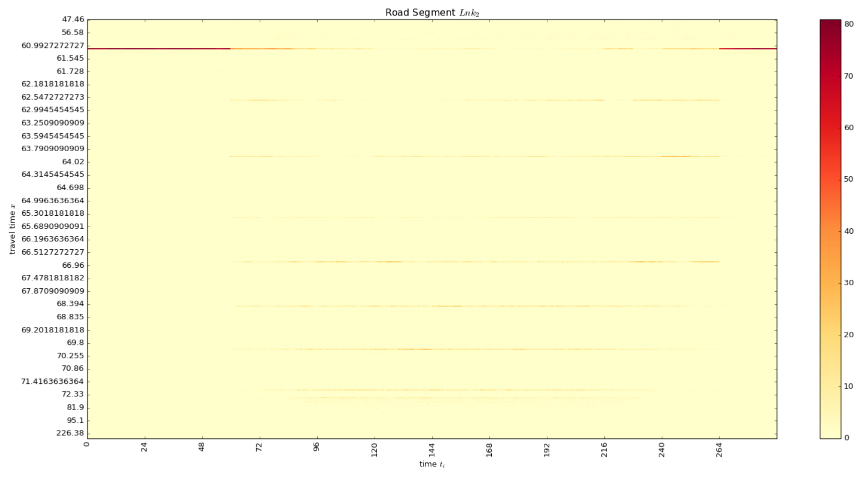
In order to further our understanding of the behaviour on road segments in the network, we investigate the influence that the day of the week has on the regularity (or irregularity) of traffic flow. In this analysis we produce filtered frequency matrices using Algorithm 7 but we filter the dates on line (13) such that they are only weekdays or weekends not both, and we produce a new set of Heat maps with respect to the weekdays or weekends observations. These Heat maps are presented in Figure 4.10 and 4.10. On review of the Heat maps for day of the week we observe that the weekdays appear to be the most similar to the unfiltered Heat maps for each of the road segments. This suggests that the weekdays had more influence on the regularity of traffic flow. However this could be related to the fact that there are more dates encompassing the weekday category.

Key findings

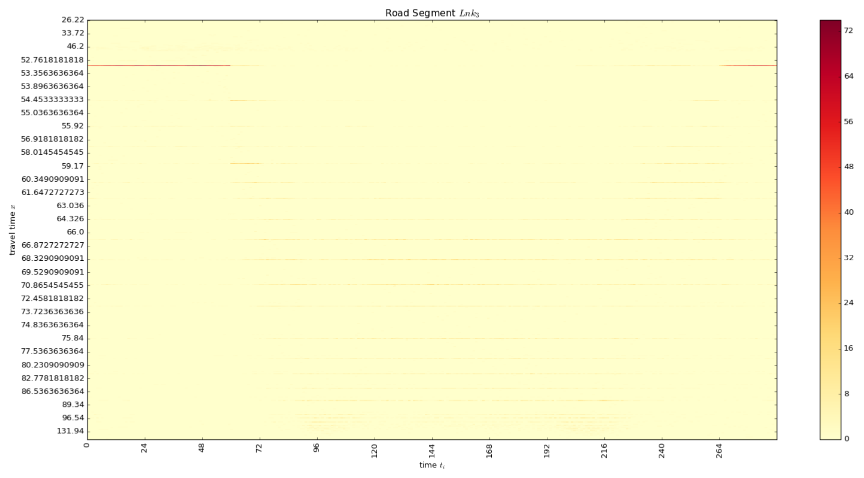
In this work we have identified a method of discovering and visualising the traffic flow behaviour for a sample set of roads in the network. In particular, we have developed a method that uses frequency distributions of observed travel times to visualise through heat maps the behaviour of traffic flow for a specific road segment. Upon investigating these the heat maps we observed that temporal influence does exist meaning the time of day at which an observation was made does have influence on what was observed. Furthermore, it was noticed that for each of the sample road segments traffic flow was mostly stable, and through further investigation patterns of irregularity occur on road segments that align with the troughs observed in the plots of averages and variances. Moreover, to further our understanding of traffic flow influences, we decided to separate observations into weekdays and weekends and produce heat maps for each. The outcome of this suggested that weekdays have more influence on the regularity of traffic flow, and this observation is based on the fact that the darker cell intensity is significantly more on weekday heat maps for each of the sample road segments. An obvious reason for this would be that more dates are considered in the weekday heat maps than the weekend heat maps.



(a)

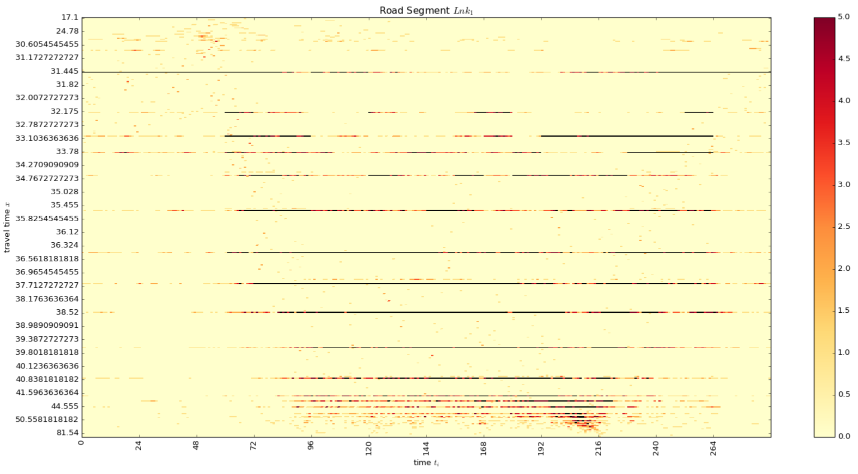


(b)

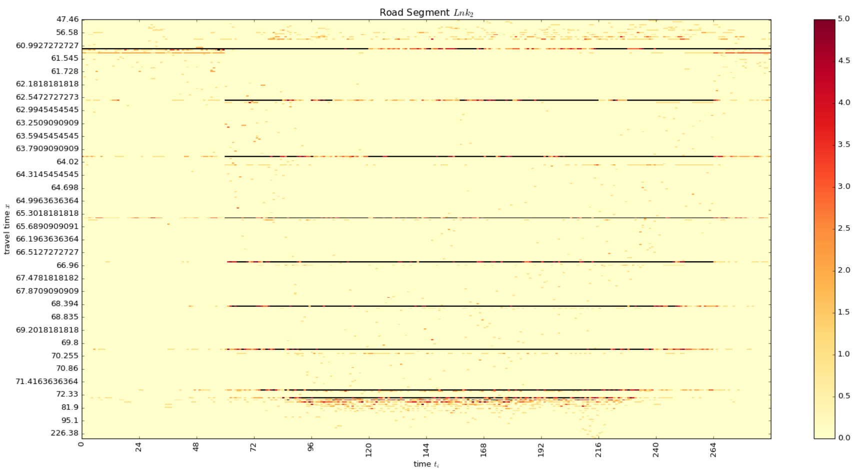


(c)

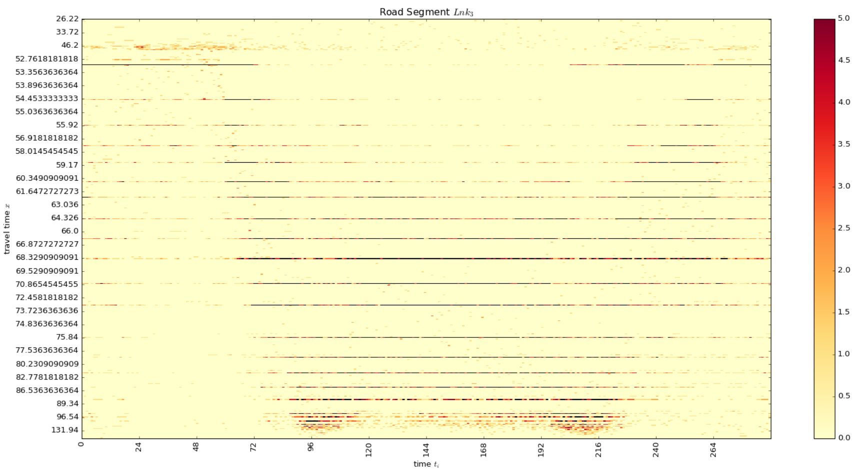
Figure 4.8: Travel time intensity Heat maps (unscaled), where light colours denote low frequency and dark colours denote high frequency.



(a)

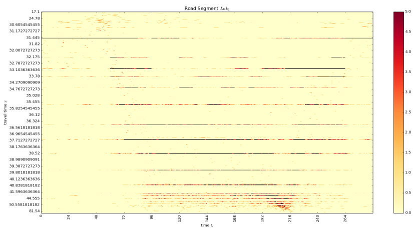


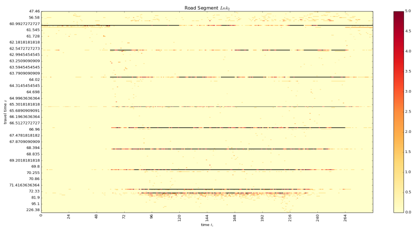
(b)



(c)

Figure 4.9: Travel time intensity Heat maps (scaled), where light colours denote low frequency and dark colours denote high frequency.


(a)


(b)

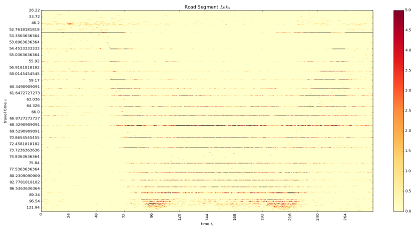
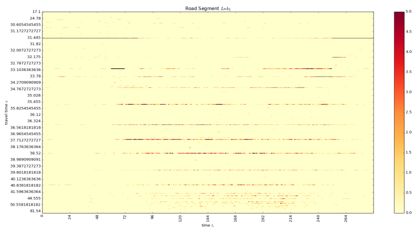
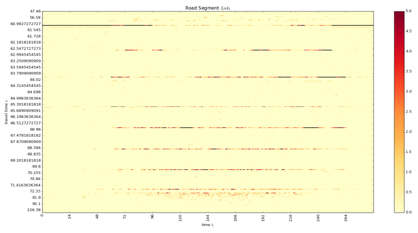

(c)

Figure 4.10: Travel time intensity Heat maps (scaled) for *weekdays*, where light colours denote low frequency and dark colours denote high frequency.


(a)


(b)

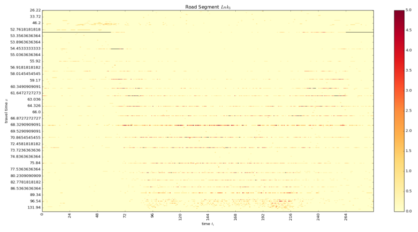

(c)

Figure 4.11: Travel time intensity Heat maps (scaled) for *weekends*, where light colours denote low frequency and dark colours denote high frequency.

4.4.3 Traffic flow modelling

The focus of this experiment is to investigate question 3, that is, we consider probabilistic modelling techniques to simulate traffic flow in the network. More specifically, this work is motivated by similar work [118] that considers stochastic processes for predicting traffic flow in a vehicular traffic network. In [118], Manley et al. consider a stochastic process known as a Markov Chain, see appendix D for details, to model traffic flow for route selection across London based on taxi-cab trajectory data. Recall that [87] motivated the work in section 4.4.2, the work in this section uses this understanding to develop a method of providing a time-dependent predictive model of traffic flow on a specific road segment Lnk_i.

In order to build an accurate predictive-model we must first consider a method of reducing the dimensionality of the data in order to make our analysis tractable. Dimensionality reduction in this case means reducing the number of possible observed travel times to a set of discrete values, meaning it is a technique that is used to reduce the data domain to a small discrete space, where each datum is described by some local property. A method of achieving this reduction is known as *limit grouping*, which is a technique that is derived from statistics, and in particular the analysis of quantitative data, the unfamiliar reader should consult [169] for details. Limit grouping is a method that is typically applied to whole numbers to classify data based on the limits of a class, where each class consists of a range of values, the smallest is the *lower limit* in the class, and the largest is the *upper limit* in the class. It is common for this technique to be considered as a method of binning data, that is, a method of allocating data into k bins, where k determines the range of values that reside within a single bin denoted b_i. Recall in Section 2.1 it was suggested that quantitative data should be grouped into intervals between 5 and 10, so in this work we consider the middle of this range, 10.

In our analysis we consider this technique to reduce the number of uniquely observed travel times to a defined quantity and use this refined data set in our investigation of discovering how to simulate network traffic behaviour using probabilistic modelling. The structure of this work is defined as follows. Section 4.4.3 is motivated by the work in [87] and is concerned with transforming the refined data into a series of matrices that denote the transitional probabilities between the identified bins for a specific time of day and day of week. Section 4.4.3 is motivated by the work in [118] but focusses on using the generated transition matrices to model the behaviour of traffic on a specific road segment in the network.

Transition Matrices

Description: This experiment is motivated by work in [118], where Cheng et al. propose a method that is based on Markov Chains for selecting routes in a road network based on the current location of the taxi-cab and a preprocessed set of route choice behaviours for 700,000 taxi-cab journeys in London. The preprocessing step involves generating a transition matrix to denote the transitional probability between all previously visited locations and the locations they moved to in the network. However, instead of generating transition matrices to denote transitions between locations in the network, we consider transition matrices to denote transitions between the bins in the refined data set (denoting travel times). Therefore, we build a probabilistic model that represents the behaviour of the traffic flow for a specific road segment in the network. Furthermore, based on the results presented in 4.4.2, we consider the transition matrices to be *temporal*, that is, they are dependent on the time of day and day of week at which the original observations were made. This means that a single road segment Lnk_i and day of week d_i is described by a series of transition matrices denoted $T_{d_i} = M_t^1, M_t^2, \ldots, M_t^{288}$, where each matrix denotes the transitions between observed travel times x at specific times of day t_i, and a single cell in a matrix denoted $M_t^1[i, j]$ represents the probability that travel time i transitions to travel time j between times of day t_i and t_{i+1}. The following procedure is used to produce a series of matrices for a specific Lnk_i and d_i.

Procedure: Algorithm 8 is used to produce the transition matrices for a specific road segment, it is described as follows. At the start of the algorithm on line (1) the output list is initialised. Between lines (2) and (9) is a *For* loop that forms the main part of the algorithm. Line (2) loops throughout each time of day t_i, line (3) initialises the transition matrix for a specific time of day s.t. each cell denoted $m_{i,j}$ of the matrix is 0. Line (4) to (6) is a nested *For* loop that identifies the transition frequencies between consecutive times of day t_i and t_{i+1}. In line (7) the created transition matrix for a specific time of day t_i is added to the list of matrices for a specific road segment Lnk_i. Finally, in line (9) the list of transition matrices denoted T is returned.

Traffic Flow Simulation

Description: This experiment is a natural progression of section 4.4.3, that focuses on using the generated transition matrices to model the behaviour of the traffic flow for a specific road segment and day of the week. In order to model the behaviour on the road segment we propose an approach that is based on a Markov chain, similar to that mentioned in [118], and a novel state selection algorithm that is based on *prefix-sums*, an unfamiliar reader should consult [45] for details. The following procedure is used to

Algorithm 8 transitionMatrices: function to compute a series of transition matrices for a road segment on a specific day of the week

Input: L: a list comprising vectors of observations for a specific road segment Lnk_i and a specific day of the week day_i
Output: L_{trans}: a list of transition matrices for a specific road segment Lnk_i and a specific day of the week day_i

```
1: $T_{d_i} = M_t^1, M_t^2, \ldots, M_t^{288}$    ▷ initialise the list of transition matrices
2: for $t_i$ in $\{1, \ldots, 288\}$ do
3:     $M_t^{t_i} := (m_{i,j})_{n \times n}$    ▷ initialise matrix of size $n$ by $n$, where each cell $m_{i,j} := 0$
4:     for each $\vec{v_i}$ in $L$ do    ▷ identify the transition frequencies
5:         $M_t^{t_i}[\vec{v_i}[t_i], \vec{v_i}[t_{i+1}]] \leftarrow M_t^{t_i}[\vec{v_i}[t_i], \vec{v_i}[t_{i+1}]] + 1$
6:     end for
7:     $T^{t_i} \leftarrow M_t^{t_i}$    ▷ set index $t_i$ of the output array to transition matrix
8: end for
9: return $T$
```

run the Markov process for each day of the week on a specific road segment Lnk_i, and the current state at each stage in the process is recorded. This produces a list that has length that is equal to the number of iterations that the process is run for. In order to compare the accuracy of the method for modelling traffic flow on each day of the week, we compute the frequency distribution of this list and produce a histogram, and compare the resulting histogram with a histogram that denotes the frequency distribution of the refined data. The histograms for both the synthetic and refined data are presented in Figures 4.12, 4.13, 4.12 and 4.13, where the synthetic data is that produced using the methods described below. The results of this experiment are following the procedure description.

Procedure: The traffic flow experiment is conducted using a combination of two algorithms. Algorithm 9 performs the Markov Chain experiment that is dependent on Algorithm 10, where Algorithm 10 is a function that is based on *prefix-sums* [45] that identifies the next state of the experiment (or travel time) based on the input transition vector for the current state b_i and time of day t_i. The procedure of conducting a Markov Chain experiment is described as follows. The procedure is initialised by passing in the input parameters into Algorithm 9, the input parameters are: T the list of transition matrices for a specific road segment, and m the number of expected steps of the experiment. Once initialised the algorithm begins. At steps (1) and (2) the algorithm initialises the the Markov Chain vector and current bin index respectively. Between steps (3) and (10) the algorithm loops throughout each time step of the experiment. Whilst looping throughout each time step, step (4) extracts the relevant transition vector associated with the current time index and current travel time. Step (5) uses Algorithm 10 to select the next travel time using the transition vector identified in step (4). The procedure of identifying the next travel time using Algorithm 10 is described as. Steps (1) to (3) check if the vector passed in contains any probability values that are greater than 0, if not then the algorithm terminates returning -1. In the case the vector passes the check step (4) extracts a random integer between 1 and $\max(v)$ from the normal distribution. Step (5) initialises the next state denoted as b_j to 0. Steps (6) to (9) are a *while* loop that iteratively subtracts the consecutive value of b_j in the transition vector from the random integer. At the point the place-holder for the random integer goes below 0 the loop stops, and the previous b_j is returned. The idea behind the procedure is that frequently observed travel times are more likely to be select as the next travel time as opposed to those that are infrequently observed. On conclusion of algorithm 10 the result is returned to Algorithm 9. Step (6) to (8) of this algorithm checks the result is valid, in the case that it is the current travel time is changed to this result. Step (9) assigns the travel time to the result vector $\vec{r}$. This procedure is repeated for all time steps in m, before terminating and returning the result vector in step (11).

Algorithm 9 markovChain: function to perform a Markov Chain process using the transition matrices for a specific road segment Lnk_i.

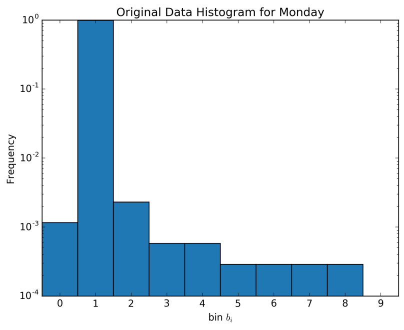
Input: T: a list of transition matrices for a specific road segment Lnk_i and a specific day of the week d_i; m: an integer value denoting the number of time steps in the Markov Chain
Output: *result*: a vector of trajectory for the Markov Chain experiment

```
1:  $result \leftarrow []$                          ▷ initialise the Markov Chain trajectory vector
2:  $current \leftarrow 0$                          ▷ initialise the current bin index
3:  for $t_i$ in $\{1, \dots, m\}$ do               ▷ loop throughout each time step of the experiment
4:      $\vec{t_i} \leftarrow L^{ti}_{trans}$        ▷ extract the appropriate matrix
5:      $tmp \leftarrow prefixSumSelection(\vec{t_i})$   ▷ get next bin based on the transition matrix using Algorithm 10
6:      if $tmp \neq -1$ then
7:          $current \leftarrow tmp$
8:      end if
9:      $result_{t_i} \leftarrow current$
10: end for
11: return $result$
```

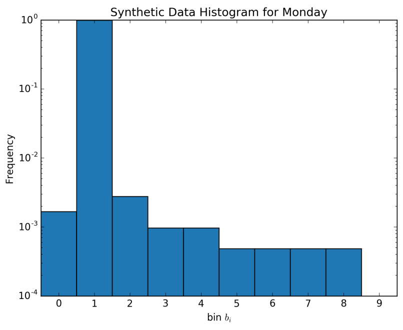
Results: Through the examination of the histograms presented in Figures 4.12, 4.13, 4.12 and 4.13 it is evident to see that the shape of the distributions is almost exact with negligible differences observed between the synthetic and original histograms. This suggests that our approach performs well in terms accuracy for modelling the behaviour of traffic flow on each of the days of week.

Key findings

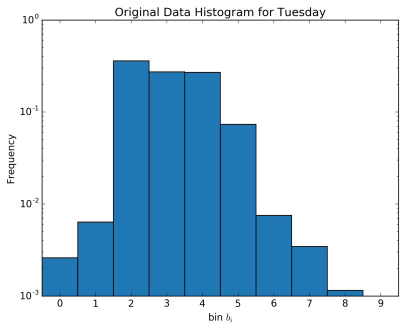
In this work we have identified a method of modelling traffic flow for road segments in the network. The method proposed examines the floating-car data to discover the probability that a floating-car will experience a particular travel time for a specific time of day on a specific day of the week. This



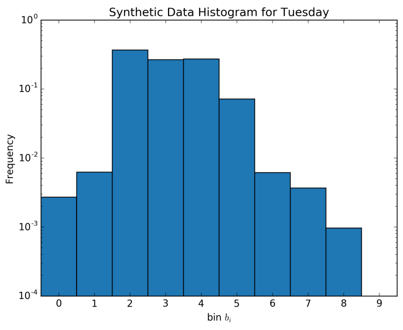
(a)



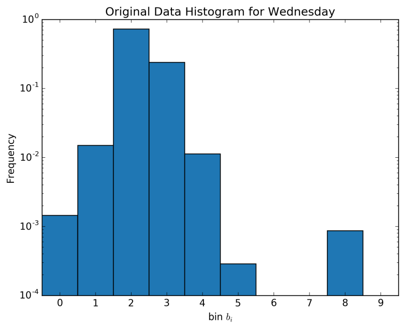
(a)



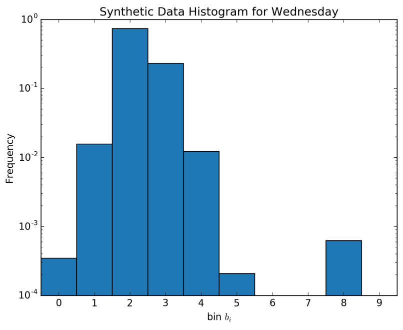
(b)



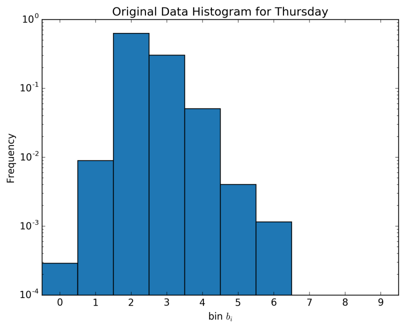
(b)



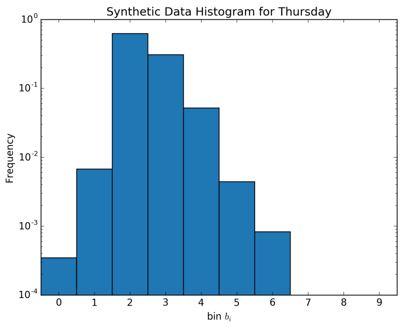
(c)



(c)



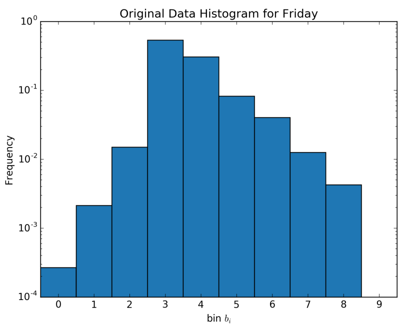
(d)



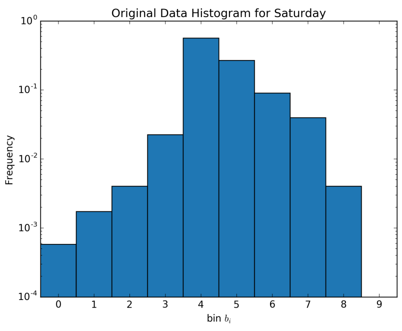
(d)

Figure 4.12: Original frequency distributions for Monday to Thursday

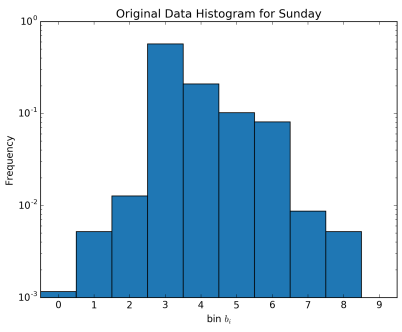
Figure 4.13: Synthetic frequency distributions for Monday to Thursday



(a)

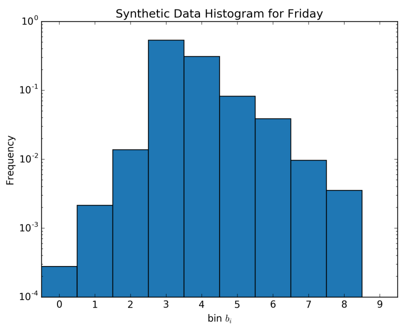


(b)

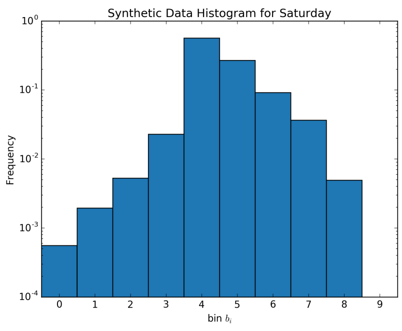


(c)

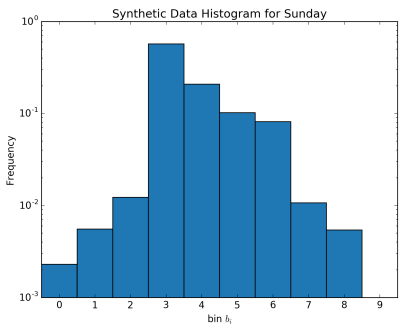
Figure 4.14: Original frequency distributions for Friday to Sunday



(a)



(b)



(c)

Figure 4.15: Synthetic frequency distributions for Friday to Sunday

Algorithm 10 prefixSumSelection: function to identify the next b_i based on the probability vector

Input: $\vec{v}$: a vector of probabilities for transitioning between travel time i and any other travel time
Output: b_i: the next travel time dependent on the probability vector $\vec{v}$

```
1: if $sum(\vec{v}) = 0$ then
2:     return $-1$
3: end if
4: $r \leftarrow rand(1, max(\vec{v}))$        ▷ extract a sample integer from a normal distribution
5: $b_j \leftarrow 0$
6: while $r > 0$ do
7:     $r \leftarrow r - \vec{v}_{b_j}$
8:     $b_j = b_j + 1$
9: end while
10: return $(b_j) - 1$
```

model was then used in a Markov Chain to produce a vector of synthetic travel times, and for this vector we computed the frequency distribution. The frequency distribution of synthetic travel times was compared to a frequency distribution for the original data under the same conditions, that is, for the same road segment and day of week. This comparison was used to confirm how well the model produced synthetic data reflects the actual observations. The results of this comparison suggested that the shape of the distributions were almost exact and the observed differences between the synthetic and original histograms are negligible, suggesting the model is an accurate way of representing the traffic flow.

4.5 Conclusions and future work

The study of road networks is still in its preliminary stages consider methods such as causality detection, traffic flow modelling, outlier detection etc.

In this work we have considered a data-driven approach to generate understanding of the traffic flow behaviour in a network. In particular we have identified a methodology, based on statistical analysis, to deliver insights into the patterns of traffic flow observed in a vehicular traffic network. The main motivation of this chapter was to generate understanding of the behaviour in an urban traffic network. This was achieved through a sequence of experiments. The first experiment helped identify a method of discovering self-similarity in a time-series describing traffic flow in a network. The second experiment identified a method of discovering and visualising the patterns in time-series describing traffic flow in a network, and helped identify influential factors such as time of day and day of week. The third experiment identified an accurate data-driven method for modelling and simulating traffic flow in a network. The methodology produced many novel algorithms, most of which can be applied to time series data sets in order to clean, transform and generate exposure to its inherent patterns. Furthermore, we propose a novel algorithm that can be used to model network traffic, and thus has applications in many fields and disciplines including Physics, Engineering, Psychology, Biology etc. In our future work we plan to consider 1) studying other parameters that influence traffic flow such as *weather*, *local events such as Football matches* etc., 2) aggregate traffic flow data for more road segments to provide a more complete analysis of the road network, 3) compare our model to known probability models such as Poisson for predicting traffic flow. These enhancements can provide a more comprehensive review of the traffic network, and generate clarity on the behavioural characteristics of traffic in the network.

5

Deterministic Rendezvous in Restricted Graphs

In the previous two Chapters of this thesis we have discussed the problems of discovering from a corpus of data what can be used to provide insight into the behaviour of a network, and when we have this knowledge, but the data set is limited in some way, what insight can we gain from it. In this chapter we consider the challenge of having exposure to increasingly less information; than is considered in the two previous chapters of the thesis, about the network, that is, the data describing the network is limited even further than that considered in Chapter 4, and we demonstrate this challenge by considering the rendezvous problem in a distributed setting. The rendezvous problem is a challenge in which two or more mobile entities, called later *robots* have the goal to meet at the same point and time in provided space. This space can be either a network of discrete nodes between which robots can move along existing connections, or a geometric environment in which the movement of robots is restricted by the topological properties of the space. As indicated in [102] *symmetry* plays a key role in determining the feasibility and efficiency of solutions in the rendezvous problem. It is quite commonly observed that *anonymous* (indistinguishable) players find themselves in a situation where the tools and advice given to each robot are identical and rendezvous may not be feasible [11]. In this context, determining even small pieces of information that would help to distinguish between participating robots often prove to be vital in achieving rendezvous. Rendezvous problems have been studied in a number of different settings. A vast literature includes several exhaustive surveys on the topic and other searching games, see, e.g., [10, 11, 134]. The work on rendezvous includes both deterministic algorithms surveyed recently in [134] as well as randomised approaches including classical work in [8, 9, 16, 24]. Another group of algorithms focus on geometric setting including earlier work on the line [24, 25] and the plane [14, 15] as well as more recent work on fat (with non-zero radius) robots [2, 50]. Another interesting group of rendezvous algorithms is designed for infinite (Euclidean) spaces for both synchronised and asynchronous solutions [43, 44, 52]. An important group of rendezvous algorithms have been considered for graph based environments, see, e.g., [51, 101, 124]. However, all previous work is devoted to the case when both robots have access to the same part of the network. An interesting version of rendezvous in which robots face different costs associated with traversed edges was considered recently in [60] where the authors consider scenarios with and without communication between participating robots.

Our work refers to the extreme communication-less case of [60] in which the costs imposed on edges are either unit or infinite. We also make reference to *blind rendezvous* considered recently in the context of cognitive radio networks [38, 108].

5.0.1 Model of Computation

We consider rendezvous of *anonymous* (indistinguishable also with respect to the control mechanism) robots in networks modelled by graphs. The network $G = (V, E)$ where the two robots are expected to rendezvous is a simple connected graph in which two nodes $s_A, s_B \in V$ are selected as the starting points for robots A and B respectively. Moreover, for each $X \in \{A, B\}$ we define its *reachability graph* also referred to as *the map* $G_X = (V_X, E_X)$, a subgraph of G in which V_X and E_X are respectively the sets of nodes and edges accessible from s_X. Moreover, agent X is only able to see its own map. Let $k_X = |V_X|$ be *the size* of map G_X and assume w.l.o.g. that $k_A \leq k_B$. While the robots are anonymous, we use extra assumptions with respect to the network nodes (and in some cases edges too). In particular, we assume that all nodes of the input network graph $G = (V = \{v_1, v_2, \ldots, v_n\}, E)$ are ordered, s.t., $v_i < v_{i+1}$ for all $i = 1, 2, \ldots n-1$ and this order is consistent with the order of nodes in G_X, for $X \in \{A, B\}$. In particular, if $V_X = \{v_1^{(X)}, v_2^{(X)}, \ldots, v_{k_X}^{(X)}\}$, $v_a^{(X)} = v_i$, and $v_b^{(X)} = v_j$, where $v_i, v_j \in V$ and $i \leq j$, we also get $v_a^{(X)} \leq v_b^{(X)}$. Finally, let $T(V_X)$ be a rooted tree that spans all nodes in V_X in which the starting point s_X is placed in the root of $T(V_X)$ and the order on children is consistent with the order of nodes in V_X.

In the synchronous setting two robots A and B have access to the global clock ticking in discrete *time steps* $0, 1, 2, \ldots$. Our algorithms start with the global clock set to time 0. During a single time step each robot assesses the node in which it resides (including check for co-location/rendezvous with the other robot). Then the robot decides whether to stay at the same node or to move to one of its neighbours via an available (edge) connection, where the neighbour is chosen at random. During the traversal between two connected nodes the "eyes" of the robot are closed. Consequently, since the robots cannot meet on edges rendezvous has to take place at some node. The running time of all algorithms is bounded, i.e., the robots stop within the time given to the respective rendezvous algorithms.

In the asynchronous setting each of the robots chooses its trajectory, i.e., the sequence of visited nodes, without access to the global clock. Instead, it is decided by the adversary which of the robots will make a move in every time step. To be consistent with the synchronous model we assume that rendezvous is possible only on nodes. Instead of the running time, as the efficiency measure, we take the maximum length of two robots trajectories.

In what follows we consider three **models of computation** with different levels of restriction imposed on maps provided to robots A and B.

1. **Edge Monotonic (EM) Model** This model is motivated by the case in which each robot $X \in \{A, B\}$ has weight w_X and each edge in E has weight restriction. This setting imposes an order on edges in $E = \{e_1, e_2, \ldots e_m\}$, in which for any $1 \le i < j \le m$ edge e_j tolerates weights non-smaller than e_i. Let i_X be the smallest integer, s.t., e_{i_X} tolerates weight w_X. One can conclude that robot X is only allowed to traverse edges with index $\ge i_X$. Consequently in this model if rendezvous is possible $E_A \subseteq E_B$ and $V_A \subseteq V_B$, i.e., G_A is a subgraph of G_B, for further details see section 5.1.1.

2. **Node Inclusive (NI) Model** In this model we only assume $V_A \subseteq V_B$, i.e., the relationship between edges spanning nodes in E_A and E_B remains unspecified.

3. **Blind Rendezvous (BR) Model** In this model we only assume $V_A \cap V_B \neq \emptyset$ and the relationship between E_A and E_B is unspecified.

Table 5.1: Summary of results

model	time complexity for synchronised setting	length complexity for asynchronous setting
EMM	$\Theta(k_A + k_B)$ (Thm 1)	$O((k_A + k_B)^2)$ (Thm 7)
NIM + $E_A \subset E_B$	$O((k_A + k_B)\log(k_A + k_B))$ (Thm 2)	feasible (Thm 8)
NIM		
BRM	not feasible (Thm 4)	not feasible (Thm 9)
BRM + explicit labels	$\min\{O((k_A + k_B)^3 \log\log n,$ $O((k_A + k_B)^2 \log n)\}$ (Thm 5)	

5.0.2 Our results

In this work we study rendezvous in three different restriction models. In Section 5.1 we study synchronised rendezvous. In particular, in subsection 5.1.1 devoted to EM model we present an optimal $O(k_A + k_B)-$time rendezvous algorithm 11; in Section 5.1.2 we present rendezvous algorithm 11 that meets two robots in almost linear time $O((k_A+k_B)\log(k_A+k_B))$ in NI model; and finally in Section 5.1.3 we show that in BR model rendezvous is not feasible. In order to overcome this deficiency we introduce explicit labels and present two rendezvous algorithms 14 and 15 whose superposition allows robots to meet in time $\min\{O((k_A + k_B)^3 \log\log n, O((k_A + k_B)^2 \log n)\}$.

In Section 5.2 we study asynchronous rendezvous. In subsection 5.2.1 we present a solution of length $O((k_A + k_B)^2)$ generated by algorithm 16. Section 5.2.2 provides the feasibility of the case where one of the edge sets is included in the other one. Without this assumption we show that the rendezvous is not feasible, which implies the infeasibility of asynchronous Blind Rendezvous. The summary of the results is given in Table 5.1. We conclude with the final comments in Section 5.3.

5.1 Synchronous rendezvous Algorithms

In this section we design and analyse several rendezvous algorithms in a synchronous setting for three restriction models.

5.1.1 Rendezvous in Edge Monotonic Model

Recall that in this model, we adopt the order of edges in $E = \{e_1, e_2, ..., e_m\}$ where $e_i < e_{i+1}$. For any $l \in \{1, \ldots, m\}$, we define a sequence of subgraphs $G(l) = (V(l), E(l))$, where $E(l) = \{e_l, e_{l+1}, ..., e_m\}$ and $V(l)$ is the set of nodes in V induced by the edges of $E(l)$, and $E(l + 1) \subset E(l)$. In this model each robot X is associated with the threshold index $i_X \in \{1, \ldots, m\}$ determining the set of edges $E(i_X)$ traversable by X. In other words, robot X can only walk along edges from $E(i_X)$. We also define a sequence of connected components $G_X(l) = \{V_X(l), E_X(l)\}$, for $l \in \{i_X, \ldots, m\}$, where $V_X(l)$ is the set of nodes reachable from s_X via edges in $E(i_X)$, and $E_X(l) \subseteq E(l)$ is the maximal set of edges spanning nodes in $V_X(l)$. So in this case $V_X = V_X(i_X)$, $E_X = E_X(i_X)$, and $k_X = |V_X(i_X)|$. The following Lemma holds.

Lemma 1. In Edge Monotonic Model either $(V_A \subseteq V_B)$ or $(V_B \subseteq V_A)$, or $V_A \cap V_B = \emptyset$.

Proof. The lemma (statement) would be false if all of the terms $(V_A \subseteq V_B), (V_B \subseteq V_A)$, and $V_A \cap V_B = \emptyset$ were false too. Assume w.l.o.g. that $V_A \cap V_B \neq \emptyset$, where $V_A = V_A(i_A)$ and $V_B = V_B(i_B)$, and $i_A \geq i_B$. Since $i_B \leq i_A$ (edges traversable by A are also traversable by B) and $V_A \cap V_B \neq \emptyset$ (the reachability graphs G_A and G_B coincide) all edges and points in $G_A(i_A)$ are also available to robot B, meaning $V_A \subseteq V_B$. $\square$

We define the concept of a *sleeve of graphs* with respect to X denoted by $SL(X)$.

Definition 5.1.1. The sleeve of graphs $SL(X)$ with respect to robot X is the maximal sequence of connected components $G_X(i_X), G_X(i_X{+}1), \ldots, G_X(l^*)$, in which $|V_X(l+1)| > |V_X(l)|/2$, for all $i_X \le l^* < m$. A subsequence $G_X(i_X{+}j), G_X(i_X{+}j{+}1), \ldots, G_X(l^*)$, for any $j \in \{0, 1, \ldots, l^* - i_X\}$, is called a tail of $SL(X)$ and the smallest (in the adopted order) node $v^*_X \in V_X(l^*)$ is called the target in $SL(X)$.

In what follows we present a pseudo-code of the proposed rendezvous algorithm in the monotonic model. If at any time step the two robots A and B meet, the rendezvous is achieved and the two robots *halt*.

Algorithm 11 RVI

```
   Input: $X \in \{A, B\}$: a robot
1: Step 1 Walk from $s_X$ to the target node $v^*_X$ in $SL(X)$
2: Step 2 Wait in $v^*_X$ until conclusion of time step $2k_X$
3: Step 3 Walk along the Euler tour of $T(V_X)$ and Halt
```

Theorem 1. If rendezvous is feasible, Algorithm 11 admits meeting in optimal time $O(k_A + k_B)$.

Proof. Recall that $k_A \le k_B$. According to Lemma 1 if rendezvous is feasible, i.e., $V_A \cap V_B \ne \emptyset$ we conclude that $V_A \subseteq V_B$. We consider two complementary cases:

Case 1 $[2k_A > k_B]$ Since $2k_A > k_B$ according to Definition 5.1.1 sleeve $SL(A)$ is a tail of $SL(B)$ and the two sleeves share the same target v^*. The robots A and B are initially placed in their own sleeves at distance at most $k_B < 2k_A$ from the joint target v^*. This admits rendezvous in **Step 1** in time at most k_B.

Case 2 $[2k_A \le k_B]$ In this case, robot A halts at the latest at time step $4k_A$ on the conclusion of **Step 3**, i.e., after $2k_A$ time steps devoted to

Step 1 and **Step 2**, followed by additional $2k_A - 2$ time steps devoted to the Euler tour traversal in $T(V_A)$) in **Step 3**. Note, however, that robot B enters **Step 3** in time step $2k_B + 1 > 4k_A$, when robot A is already immobilised. Since during **Step 3** robot B visits all nodes in V_B (that include also all nodes in V_A) rendezvous must occur.

□

5.1.2 Rendezvous in Node Inclusion Model

Recall that in this model we assume that all nodes are ordered and $k_A \leq k_B$, where $V_A \subseteq V_B$. In this model we have no order on edges and in turn the concept of sleeve of graphs cannot be applied here. Instead, one can focus on a different mechanism that will allow to distinguish between two robots and with this in mind we focus on the values of k_A and k_B. Note that if $k_A = k_B$ due to the inclusion assumption we also have $V_A = V_B$. In this case, since orders of nodes in V_A and V_B are consistent the robots can meet at the smallest (in order) node v^* in V_A and V_B that must coincide. Otherwise, the values of k_A and k_B differ and each robot X, for $X \in \{A, B\}$ can adopt k_X as its unique identifier. Furthermore, apart from unique identities there needs to be a synchronisation mechanism (sizes of k_A and k_B can be dramatically different) that will allow robots to coordinate their individual moves. The rendezvous mechanism for any robot X is based on synchronised waiting of the first stage that is long enough to accommodate actions reflecting the size k_X. In particular, we identify the power of two j_X, s.t., $2^{j_X - 1} \leq k_X < 2^{j_X}$ that provides a constant estimation and the upper bound on the size of k_X. The algorithm applied to robot X operates in stages $j = 1, 2, 3, ..., j_X$, where during stages 1 through $j_X - 1$ the robot remains immobilised and in the last stage j_X it actively participates (visiting all nodes in V_X) in the rendezvous process. Note that if $j_A < j_B$ (and $V_A \subset V_B$) in stage j_B, when robot A is already immobilised, B by visiting all nodes in V_B (that is a superset of V_A) must conclude rendezvous. In the complementary case, i.e., when the estimates j_A and j_B are the same we use binary expansions $b_A[0,\ldots,j_A]$ and $b_B[0,\ldots,j_B]$ (where positions j_A, j_B are the most significant) of k_A and k_B respectively to differentiate between the robots.

Lemma 2. If $j_A = j_B$ and $k_A < k_B$ there exists $i \in \{0,1,\ldots,j_A = j_B\}$, s.t., $b_A[i] = 0$ and $b_B[i] = 1$.

Proof. If for all $i \in \{0,1,\ldots,j_A = j_B\}$, $(b_A[i] = 0) => (b_B[i] = 0)$ would imply $k_A \geq k_B$. □

A pseudo-code of the rendezvous algorithm 12 in the inclusion model follows. If at any time step the two robots A and B meet, the rendezvous is achieved and the two robots *halt*.

Algorithm 12 RVII

```
    Input: $X \in \{A, B\}$: a robot
 1: Compute $j_X$ and $b_X[0,\ldots,j_X]$.
 2: for $j$ in $\{1,\ldots,j_X\}$ do
 3:     if $(j = j_X)$ then                                   ▷ active Stage
 4:         use $2^j$ time steps to walk to and wait in $v_X^*$.   ▷ $v_X^*$ smallest node
 5:         for $i$ in $\{0,\ldots,j\}$ do
 6:             if $(b_X[i] = 1)$ then
 7:                 (a) use $2 \cdot 2^j$ time steps to visit Euler tour in $T(V_X)$
 8:                 return to $v_X^*$
 9:             else
10:                 (b) wait $2 \cdot 2^j$ time steps in $v_X^*$
11:             end if
12:         end for
13:     else
14:         wait $2^j \cdot (2j+3)$ time steps where you are
15:     end if
16: end for
```

We prove the following theorem.

Theorem 2. If rendezvous is feasible Algorithm 12 admits meeting in time $O((k_A + k_B)\log(k_A + k_B))$.

Proof. The rendezvous algorithm runs in j_X stages controlled by the loop For in line 3. There are two cases. In the first case, where $j_A < j_B$, when robot B is in the active stage robot A is already immobilised, and B meets A during traversal of the Euler tour in $T(V_B)$, see line 8 of the code. Otherwise, when $j_A = j_B$ we have two subcases. In the first subcase when $k_A = k_B$ the robots meet in the shared smallest node v^*, see line 5. In the second subcase, where $k_A < k_B$, according to Lemma 2 there exists i, s.t., $b_A[i] = 0$ and $b_B[i] = 1$ when robot B visits the Euler tour in $T(V_B)$ and robot A is immobilised. Thus this subcase admits rendezvous too.

With respect to the time complexity we first observe that the execution time of algorithm 12 is bounded and it depends on the parameter j_X. The time complexity of each stage $j = 1, ..., j_X$ is bounded by $(2^j \cdot (2j + 3))$, as indicated in line 11 in the pseudocode, resulting in the total complexity $\sum_{j=1}^{j_X}(2^j \cdot (2j+3)) \leq \sum_{j=0}^{j_X}(2^j \cdot (2j_X + 3))$. This is equivalent to $(2j_X + 3)\sum_{j=1}^{j_X}(2^j) = (2j_X+3)\cdot(2^{j_X+1} - 2) = O(k_X \cdot \log k_X)$, since $2^{j_X} - 1 \leq k_X < 2^{j_X}$. This admits the time complexity $O((k_A + k_B)\log(k_A + k_B))$. □

Bounded diameter networks

Here we comment on the case when both agents know the common bound for the diameter of G_A and G_B, and lets assume this bound is defined by Equation 5.1. In this case there exists a simple and asymptotically optimal linear time solution in the sychronous model, see Algorithm 13.

$$\max\{\text{diam}(G_A), \text{diam}(G_B)\} \leq d. \tag{5.1}$$

Algorithm 13 RVIIb

```
   Input: $X \in \{A, B\}$: a robot
1: for $j$ in $\{1, \ldots, k_X\}$ do
2:     Walk to $j$-th node
3:     Wait until time $d \cdot j$
4: end for
5: Halt
```

Theorem 3. If both agents know the common bound d which limits the diameter of G_A and G_B then Algorithm 13 admits rendezvous in the asymptotically optimal time $O((k_A + k_B))$.

Proof. Recall that $V_A \subseteq V_B$ and the orders of the nodes in both sets are consistent. Lets u be the last node (in order) visited by agent A. Note that u belongs also to V_B. We observe that when agent B visits u the other agent A is already immobilised in u as agent B must visit leats the same number of nodes as A before arriving in u. Thus, rendezvous of A and B will take place at the latest in u in time bounded by $k_B \cdot d = O((k_A + k_B))$. □

5.1.3 Blind Rendezvous Model

In this section we consider rendezvous where the relationship between the maps of robots is more arbitrary. We first show that without any additional information, even if $V_A \cap V_B \neq \emptyset$, rendezvous cannot be reached.

Theorem 4. Blind rendezvous is not feasible.

Proof. Assume that for any $X \in \{A, B\}$ we have $V_X = \{v_1^{(X)}, v_2^{(X)}\}$ and $E_X = \{(v_1^{(X)}, v_2^{(X)})\}$, where node $v_2^{(A)}$ coincides with $v_1^{(B)}$ and where for each robot X the starting node s_X coincides with v_1^X on its own map. It is enough to observe that without any additional information the symmetry tie cannot be broken. And indeed, since the robots are anonymous (indistinguishable) whenever robot A visits $v_2^{(A)}$ robot B visits $v_2^{(B)}$, i.e., the two robots never visit the shared node simultaneously. □

One can adopt a natural assumption that the nodes in V_X apart from being ordered they also have *explicit labels*. In consequence, if a node $v_a^{(A)} \in V_A$ coincides with $v_b^{(B)} \in V_B$ they both possess the same explicit label. We assume that the labels are drawn from the set of integers $\{1, 2, \ldots, n\}$, and we use notation $b_i^{(X)}$ (or $b_i^{(X)}[0..\log n]$) to denote the binary expansion of the explicit label of $v_i^{(X)} \in V_X$.

We also assume that n is known to both robots. Otherwise no rendezvous algorithm would stop and report infeasibility of rendezvous when $V_A \cap V_B = \emptyset$, as robots are not aware of sizes of each others maps.

Before we present two rendezvous algorithms we show that the symmetry tie problem, see Theorem 4, can be overcome if the explicit labels are available. W.l.o.g. we also assume that the order of labels is consistent with the order imposed on nodes on each map. If this is not the case a new (consistent) order for nodes in V_A and V_B can be computed on the basis of explicit labels (we only care about nodes in $V_A \cap V_B$). The following result has been shown in [38]. Our proof, however, is much simpler and based on binary representation of explicit labels.

Lemma 3. Assume that the map of any robot $X \in \{A, B\}$ is an ordered pair of nodes $(v_1^{(X)}, v_2^{(X)})$ connected by a symmetric edge, where nodes $v_2^{(A)}$ and $v_1^{(B)}$ physically coincide and nodes $v_1^{(A)}$ and $v_2^{(B)}$ don't. In such network one can break the symmetry tie to reach rendezvous in time $O(\log \log n)$.

Proof. We first observe that according to the imposed order $b_1^{(A)} < b_2^{(A)} = b_1^{(B)} < b_2^{(B)}$. The case where $s_A = v_2^{(A)}$ and $s_B = v_1^{(B)}$ is trivial and another case where $s_A = v_1^{(A)}$ and $s_B = v_2^{(B)}$ can be easily resolved by an algorithm that alternates between the two nodes (e.g., in every other time step). Let $1 \leq r_A \leq \log n$ be the largest integer position, s.t., $b_1^{(A)}[r_A] \neq b_2^{(A)}[r_A]$. Since $b_1^{(A)} < b_2^{(A)}$ one can conclude that $b_1^{(A)}[r_A] = 0$ and $b_2^{(A)}[r_A] = 1$. Similarly let $1 \leq r_B \leq \log n$ be the largest integer position, s.t., $b_1^{(B)}[r_B] \neq b_2^{(B)}[r_B]$. Since $b_1^{(B)} < b_2^{(B)}$ one can also conclude that $b_1^{(B)}[r_B] = 0$ and $b_2^{(B)}[r_B] = 1$. We observe that since $b_2^{(A)} = b_1^{(B)}$ one can conclude that $r_A \neq r_B$ as the respective positions cannot contain 0 and 1 at the same time. Moreover binary expansions br_A and br_B of r_A and r_B respectively are limited to $\log\log n + 1$ bits.

We consider a symmetry breaking algorithm in which in time step i each robot $X \in \{A, B\}$ moves to the other node only if $i = 2 \cdot l$ (i is even) or if $i = 2 \cdot l - 1$ (i is odd) and $br_X[l] = 1$, for $l = 1, \ldots, \log\log n + 1$. Note that since $r_A \neq r_B$ for some $1 \leq l \leq \log\log n + 1$ we must have $br_A[l] \neq br_B[l]$ and if until now the rendezvous is not reached (all previous moves were symmetric and in the last odd time step, when the symmetry was broken robots occupy different nodes) in the next even step the rendezvous is accomplished. □

Corollary 4.1. Note that the lemma above applies to pairs of nodes at distance 1. In a more general case, where the distance between nodes in the pair is $d \geq 1$, the symmetry breaking rendezvous takes time $O(d \log\log n)$.

In the remaining part of this section we present two rendezvous algorithms followed by their superposition. The first algorithm 14 has the time complexity $O((k_A + k_B)^3 \log\log n)$ and its idea is based on the blind rendezvous algorithm from [38] where the problem was studied in complete graphs. The second algorithm 15 has the time complexity $O((k_A+k_B)^2 \log n)$ making it superior to 14 when $k_A + k_B > \frac{\log n}{\log\log n} = \tau$, where τ is the threshold value. This rendezvous algorithm resembles algorithm 12 however here the symmetry tie is broken with the help of explicit labels.

Blind rendezvous in time $O((k_A + k_B)^3 \log\log n)$

Similarly to its predecessor 12 the first blind rendezvous algorithm 14 operates in stages accommodating geometrically increasing estimates on sizes of the input maps. This is needed as the size of the map of one robot is not known to the other. The robot starts using active stages only when the current estimate is large enough to accommodate its map. The rendezvous process terminates in time $O((k_A+k_B)^3 \log\log n)$ if the maps of both agents are smaller than the threshold value τ. Otherwise, algorithm 12 is followed by execution of algorithm 15. If at any time step the two robots A and B meet, the rendezvous is achieved and the two robots *halt*.

Algorithm 14 RVIII

```
    Input: $X \in \{A, B\}$: a robot
 1: Compute $j_X$ and the threshold $\tau = \frac{\log n}{\log\log n}$.
 2: for $j$ in $\{1, \dots, \dots, \lceil \log \tau \rceil\}$ do
 3:     if ($j \geq j_X$) then                                   ▷ active stage
 4:         for all pairs $(a, b) \in \{1, \dots, 2^j\} \times \{1, \dots, 2^j\}$ ordered lexicographically do
 5:             if both of $v_a^{(X)}, v_b^{(X)}$ exist then
 6:                 run blind rendezvous in pair $(v_a^{(X)}, v_b^{(X)})$ in $O(2^j \log\log n)$ time steps
 7:             else
 8:                 wait relevant $O(2^j \log\log n)$ time steps in the current location
 9:             end if
10:         end for
11:     else
12:         wait relevant $O(2^{3j} \cdot \log\log n)$ time steps where you are
13:     end if
14: end for
```

Theorem 5. If $k_A + k_B < \tau = \frac{\log n}{\log\log n}$ and rendezvous is feasible, algorithm 14 admits rendezvous in time $O((k_A + k_B)^3 \log\log n)$.

Proof. The rendezvous algorithm runs in $\lceil \log \tau \rceil$ stages controlled by the loop **for** in line 3. Robot X starts executing active stages as soon as the stages can accommodate the size of X's map. If the size of the map is too big, robot X awaits execution of the second rendezvous algorithms 15, see line 9. During an active round all pairs (a,b) from the Cartesian product $\{1,\ldots,2^j\}\times\{1,\ldots,2^j\}$ are drawn in lexicographic order. Only certain pairs are valid, i.e., when either of $v_a^{(X)}$ and $v_b^{(X)}$ exists. In each valid pair if only one node exists robot X remains in this node for the duration of the symmetry breaking procedure. Otherwise, if both nodes exist the breaking symmetry procedure is executed with the distance between the two nodes bounded by 2^j.

If rendezvous is feasible we must have nodes $v_a^{(A)}\in V_A$ and $v_b^{(B)}\in V_B$ that coincide by sharing the same label. If the pair $(v_a^{(X)},v_b^{(X)})$ exists in both maps thanks to the symmetry breaking procedure eventually robot A will visit $v_a^{(A)}$ at the same time when entity B visits $v_b^{(B)}$ and the rendezvous is reached. If only one element of the pair $(v_a^{(X)},v_b^{(X)})$ exists, i.e., either $v_a^{(A)}$ for A or $v_b^{(B)}$ for B the respective robot is asked to wait in the existing node of the pair resulting in rendezvous too. Otherwise the robots await the next pair from the Cartesian Product without movement for the period corresponding to execution of the symmetry breaking procedure. Thus the actions performed by robots A and B remain fully synchronised.

The time complexity of stage $j=1,...,\lceil\log\tau\rceil$ is bounded by $O(2^{3j}\cdot\log\log n)$. Note that if rendezvous takes place for some $j'\le\lceil\log\tau\rceil$, where $j'\le j_B$, the total time complexity is bounded by $\sum_{j=1}^{j'}O(2^{3j}\cdot\log\log n)=O(2^{3\cdot j'}\cdot\log\log n)$. And in turn since $2^{3\cdot j'}\le 2^{3\cdot j_B}=k_B^3=O((k_A+k_B)^3)$, the total time complexity is $\mathrm{O}((k_A+k_B)^3\cdot\log\log n)$.

$\square$

Blind rendezvous in time $O((k_A + k_B)^2 \log n)$

We start with the proof of the following fact.

Lemma 4. One can impose a periodic order $\pi(X)$ on nodes of a spanning tree $T(V_X)$, s.t., the walking distance (the number of edges to be visited) between two consecutive nodes in order $\pi(X)$ is at most 3.

Proof. We say that the nodes located at an even distance from the root s_X are on an even level and all the remaining nodes are on an odd level. The ordering of nodes π is created according to the following principle. Starting from the root s_X we visit all nodes in $T(V_X)$ using depth-first search algorithm. The root gets label 0. When we arrive (from the parent) to an even level the currently visited node gets the next available label. In other words at even levels we use *pre-order numbering principle*. And when we arrive (from the last child) to an odd level the currently visited node gets the next available label. I.e., at odd levels we follow *post-order numbering principle*

We need to show that the labeling (ordering) procedure proposed above generates at least one new label in three consecutive steps. And indeed, if we follow the route determined by the depth-first search algorithm and we visit for the first time a node v at an even level (when the new label is generated): (case 1) if the first child of v has a child w then w (which is at distance 2 from v) gets the new label; (case 2) if the first child of v is a leaf this child (which is at distance 1 from v) gets the new label; (case 3) if the node v is a leaf but not the last child of its parent the next label goes to the (next) sibling of v (which is at distance 2); and (case 4) if v is the last child the next label goes to its parent (which is at distance 1).

Similarly, if v is visited for the last time on an odd level it gets a new label. Now (case 5) if v is the last child and its parent w is not the last child the next sibling of the parent (which is at distance 3 from v) gets the new label; (case 6) if v is the last child and its parent w is also the list child then the parent of w (at distance 2 from v) gets the new label; (case 7) and if v is the last child and its parent is the root, the periodic order is established (and the next label is at distance 1). In the remaining cases when v is not the last child (case 8) if its next sibling (at distance 2) is a leaf it gets the new label; and (case 9) if the next sibling of v has children the next label go to the first child (at distance 3 from v) of this sibling. □ □

Algorithm 15 RVIV

```
    Input: $X \in \{A, B\}$: a robot
 1: Determine $j_X$, the threshold $\tau = \frac{\log n}{\log\log n}$, and the label $b_i^{(X)}$ of $s_X$;
 2: for $j$ in $\{\lceil \log \tau \rceil, 2, \dots, \log n\}$ do
 3:     if $(j \geq j_X)$ then                                        ▷ active stage
 4:         walk to and wait in $s_X$) in $2^j$ time steps
 5:         for $2^{2j} \times 3$ time steps do                        ▷ test all bits
 6:             if $b_i^{(X)}[l] = 1$ then                          ▷ walk all the time
 7:                 for $2^{2j} \times 3$ time steps do
 8:                     walk to the next node in order $\pi(X)$
 9:                 end for
10:             else
11:                 repeat $2^j$ times                    ▷ walk and wait for another
12:                 walk to the next node in order $\pi(X)$ and wait there) in
    $2^j \times 3$ time steps
13:             end if
14:         end for
15:     else
16:         wait appropriate $O(2^{2j} \cdot \log n)$ time steps where you are
17:     end if
18: end for
```

The last rendezvous algorithm 15 operates on the following principle. At the start of each active stage robot X returns (if moved before) to the starting point s_X. If the two starting points in V_A and in V_B coincide rendezvous is accomplished. Otherwise the algorithm controls further movement of robots, s.t., during long enough ($\geq 2^j \times 3$ time steps) interval of an active stage j

one of the robots, say w.l.o.g. A, visits all nodes in V_A in the periodic order $\pi(A)$ with frequency of one visit per three time steps. While the other robot B visits consecutive nodes with frequency of $2^j \times 3$ time steps. So when (eventually) robot B resides in the node that belongs to $V_A \cap V_B$ there is enough time for robot A to arrive in this node before B moves away. If at any time step the two robots A and B meet, the rendezvous is achieved and the two robots *halt*.

Theorem 6. If $k_A + k_B \geq \tau = \frac{\log n}{\log \log n}$ and rendezvous is feasible, algorithm 15 admits rendezvous in time $O((k_A + k_B)^2 \log n)$.

Proof. Lets consider the first stage that is active for both robots A and B, i.e., when $j = j_B$. Note that line 13 of the pseudo-code accommodates for the waiting time needed for two robots to stay synchronised prior to this stage. In this active stage loop for in line 6 compares consecutive bits of labels $b_i^{(A)}$ adopted by A and $b_{i'}^{(B)}$ adopted by B. There must be at least one position l on which the two labels differ. In consequence, there is a spell of $2^{2j} \times 3$ time steps during which one of the robots, say w.l.o.g. A with the bit $b_i^{(A)}[l] = 1$, visits periodically all nodes in V_A with frequency of 3 time steps per node. During the same time spell the other robot B with the bit $b_{i'}^{(B)}[l] = 0$ waits long ($\geq 2^j \times 3$ time steps) periods of time in every node of V_B. So when (eventually) robot B visits the node that belongs to $V_A \cap V_B$ the other robot A has enough time to arrive at this node before B moves on.

The time complexity of this first active stage is $O(2^{2j_B} \cdot \log n) = O(k_B^2 \log n)$. Since the duration of stages grows exponentially we conclude that the total time complexity is also $O(k_B^2 \log n) = O((k_A + k_B)^2 \log n)$. □

Corollary 6.1. In the Blind Rendezvous Model two robots can rendezvous in time $\min\{O((k_A + k_B)^3 \log \log n, O((k_A + k_B)^2 \log n)\}$.

Proof. The result follows directly from the superposition of 14 and 15. □

5.2 Asynchronous Rendezvous Algorithms

In this section we design and analyze several rendezvous algorithms in a asynchronous setting for three restriction models.

5.2.1 Rendezvous in Edge Monotonic Model

For $l \in \{1, 2, \ldots m\}$ let $\bar{T}(l)$ be a spanning forest in $G(l)$ computed by Kruskal's algorithm with weight for edge e_i set to $-i$.

Lemma 5. For any $i < j$ the forest $\bar{T}(j)$ is a subforest of $\bar{T}(i)$ in $G(i)$.

Proof. Having in mind that $E(j) \subset E(i)$ the lemma is straightforward from the definition of Kruskal's algorithm. □

The idea behind our solution is to construct a walk along all accessible nodes, for every l and each component of $\bar{T}(l)$ in such a way that if j-th component of $\bar{T}_{l+1}$ is a subtree of some i-th component of $\bar{T}_l$ then the walk for the former one is a fragment of the latter.

To be more precise let $W_l = (\tau_l(1), \tau_l(2), \ldots, \tau_l(s_{l-1}), \tau_l(s_l))$, where τ_l is a function from $\{1, 2, \ldots s_l\}$ to $\{e_l, e_{l+1}, \ldots e_m\}$ and let $\bar{P}(l)(e_x, e_y)$ be the path connecting e_x and e_y in $\bar{T}(l)$.

We will say that e_l is *incident* to some $W_{l'}$ if e_l is incident to $\tau_{l'}(i)$ for some $i \in \{1, 2, \ldots s_{l'}\}$. Moreover, W_l is a *fragment* of $W_{l'}$, if there exist an index μ such that for each edge, say i-th, in W_l we have $\tau_l(i) = \tau_{l'}(i + \mu)$; W_l is *maximal walk incident* to $e_{l'}$ if W_l is incident to $e_{l'}$ and for all l'' such that $l > l'' > l'$ the walk $W_{l''}$ is not incident to $e_{l'}$. Let also $NIMW(l)$ be the number of maximal walks incident to e_l.

Now, we define W_l. If $e_l \notin \bar{T}(1)$ then W_l is empty other way we define it recursively ($W_{l'}$ and $W_{l''}$ are maximal walks incident to e_l):

$$W_l = \begin{cases} (e_l) & \text{if } NIMW(l) = 0 \\ W_{l'} + \bar{P}(l)(\tau_l(s_{l'}), e_l) & \text{if } NIMW(l) = 1 \\ W_{l'} + \bar{P}(l)(\tau_l(s_{l'}), e_l) - e_l & \\ +\bar{P}(l)(e_l, \tau_l(s_1)) + W_{l''} & \text{if } NIMW(l) = 2 \end{cases}$$

The following lemmas are straightforward

Lemma 6. If $l < l'$ and W_l, $W_{l'}$ are nonempty then either:

- $W_{l'}$ is a fragment of W_l or
- they have no edge in common and every pair $e \in W_l$, $e' \in W_{l'}$ of edges are not incident.

Lemma 7. The definition of W_l is unambiguous and complete.

Lemma 8. Each edge from $\bar{T}(l)$ belongs to some $W_{l'}$ where $l' \geq l$.

Lemma 9. For any two edges e, e' in $\bar{T}(l)$ the length of $\bar{P}(l)(e, e')$ is no larger than $|V(\bar{T}(l))|$.

Lemma 10. The length of W_l is $O(|V(\bar{T}(l))|^2)$.

We have constructed W_{i_X}, $X \in \{A, B\}$ in such a way that if the rendezvous is possible then the walk constructed for one agent is a fragment of the walk constructed for the second one. Now, it is enough to force agent X to move along the whole corresponding walk W_{i_X}:

Having in mind Lemma 6, Lemma 10 we obtain the following

Theorem 7. If rendezvous is feasible Algorithm 16 admits meeting in length $O((k_A + k_B)^2)$.

Algorithm 16 RVV

```
   Input: $X \in \{A, B\}$: a robot
1: Step 1 Compute $W_{i_X}$
2: Step 2 Find such an edge $\{s_X, u\}$ that $\{s_X, u\} \in \bar{T}(i_X)$
3: Step 3 Walk along $W_{i_X}$ backward from $\{s_X, u\}$ to $\tau_{i_X}(1)$
4: Step 4 Walk along the whole $W_{i_X}$
5: Step 5 Halt
```

5.2.2 Rendezvous in Node Inclusion Model

First assume that either $E_A \subseteq E_B$ or $E_B \subseteq E_A$. In this case we are able to construct walks in the similar way as in the Edge Monotonic Model. The subtle but important difference is that we are not able to point out one edge which is not present in the preceding set of edges – we need to consider all possible subsets of edges.

We need also a pre-specified order of edge sets. Such an order might be a lexicographic order induced from the order of nodes. Let $W_{E'}$ be the walk constructed for the set E'.

If $|E'| = 1$, say $E' = \{e'\}$ then $W_{E'} = (e)$. If $|E'| = m' > 1$ then $W_{E'}$ is constructed as a sum of walks constructed for all sets F' such that $F' \subset E'$ and $|F'| = m' - 1$ connected by pre specified (again the order might be a lexicographic order induced from the order of nodes) paths from E'.

Of course, in general such a construction gives the superpolynomial length of a walk as the number of edge subsets is superpolynomial with respect to the number of edges in E_X.

Theorem 8. Asynchronous rendezvous in the node inclusion model is feasible with an additional assumption $E_A \subseteq E_B$ or $E_B \subseteq E_A$.

Suppose now that we have $V_A \subset V_B$ or $V_B \subset V_A$ but not necessarily $E_A \subset E_B$ or $E_B \subset E_A$. We will show that in this case the rendezvous is not feasible.

Theorem 9. Asynchronous rendezvous in node inclusion model is not possible even if nodes are equipped with explicit common identifiers.

Proof. Suppose that there exists an algorithm $\mathcal{A}$ that guarantees rendezvous in the node inclusion model. Let us consider the family graphs: $G_i = (V_i, E_i)$, $i \in \{1, 2, \ldots\}$, where $V_i = \{v_0, v_1, \ldots, v_i\}$ and $E_i = \{\{v_0, v_i\}, \{v_1, v_i\} \ldots \{v_{i-1}, v_i\}\}$. So, G_i is a star with i edges and $i + 1$ nodes, for each $i < j$ we have $V_i \subset V_j$ and for each i, j such that $i \neq j$ we have $E_i \cap E_j = \emptyset$.

For each of the graphs G_i and the agent placed in v_i the walk computed by algorithm $\mathcal{A}$ might be finite or infinite. There are two possible cases: for at least two graphs among G_2, G_3, G_4 the walk is infinite (case 1) or for at lease two graphs the walk is finite (case 2).

Case 1: As both agent do not halt, $s_A \neq s_B$ and the set of edges for both agents is disjoint an adversary might easily prevent the rendezvous.

Case 2: Suppose w.l.o.g. that G_B is such a graph with at least 4 nodes ($G_B = G_3$ or $G_B = G_4$) and B eventually stops in node x. As the number of nodes in G_B is at least 4 there are two different nodes $u \neq v$ such that $u, v \notin \{x, s_B\}$. Let us now consider the graph $G_A = (\{u, v\}, \{\{u, v\}\})$. Whatever the algorithm is $\mathcal{A}$ for G_A, the agent cannot stay in s_A it needs to move and while it starts moving, the adversary might slow its down until agent B will halt in x making the rendezvous impossible. □

5.3 Conclusion

In this work we studied deterministic rendezvous of two robots in the network environment with restrictions imposed on network edges. The restrictions prevent robots from visiting certain parts of the network. We considered three restriction models as described in section 5.0.1.

For the synchronised setting we provided four efficient solutions in Section 5.1. One of the open problems is to establish the exact complexity of rendezvous in the considered models and to answer whether the use of randomisation helps in this case. The case when robots are asked to meet asynchronously has been studied in Section 5.2. In the Node Inclusive Model with the additional assumption about inclusion of edge sets the lengths of respective trajectories become exponential, but the problem is feasible. It would be good to understand if the complexity in this case might be reduced. One can also consider models in which maps are not known to the robots. Another interesting question refers to better understanding (including time complexity) of gathering more than two robots. In this setting while robots could meet in pairs, one mutually accessible location for gathering may not be available.

6

Conclusion and Future Work

The topic of networks has been extensively studied in the last few decades and it is still gaining popularity. The topic of networks has been extensively studied in the last few decades and it is still gaining popularity. In this thesis we study the challenge of understanding networks when information about the network is unknown or limited in some way, and demonstrate the difficulty of this challenge by considering a rendezvous problem. Furthermore, we try to overcome this difficulty by using algorithmic and statistical approaches. The results presented in Chapter3 and Chapter 5 are the result of published work carried out by the author, their supervisors and several collaborators from different institutions. Following, is a detailed observation of the conclusions that can be obtained from the main results presented in this thesis.

6.1 Understanding and Managing Data Lakes

Chapter 3 introduces a novel method of inspecting and managing data lakes in view of delivering insights into the relationships between assembled data assets. The technique (or methodology) is based on methods and algorithms derived from network theory and social network analysis (SNA). The method involves building an abstract graph of the data assets denoted $G = (V, E)$, where the vertices represent the data assets, and the edges represents the logical relationships between a pair of data assets. The join relationship is based on some inherent property of the data assets, in the context of this work this is the quantity of columns in common. The graph is then investigated using community detection algorithms in order to discover the emergent communities and structure in this network. This analysis presented some interesting results about the structure. Of particular note is the observation that in our graph the natural community structure does not follow the initially imposed structure by the end users through the creation of logical databases. Instead, there is a richer, emergent structure which perhaps was not appreciated when the data was first assembled or which evolved opportunistically over time. A further observation is that our Hive metadata graph followed a power-law degree distribution, therefore suggesting it to be scale-free. This implies that the content of a small number of Hive tables is influential in the structure of the network. In turn, this means that lineage relationships might be apparent between a small set of Hive tables and the rest of the population in the cluster.

Future work on data discovery can consider improving the accuracy of the inferred joins between data assets through efficient indexing methods such as locality sensitive hashing (LSH) [151], where LSH is a mechanism that could allow join relationships to be considered based on the content of the data assets themselves. A further enhancement could be to harvest real-world user interactions with data assets to identify data assets that can be joined together, and this information can be aggregated by searching or monitoring Hive query logs to extract the joins that are actively being made by end users. Similar work has been discussed in [157]. The same real-world interactions will also help to address the construction of a network that captures the relationships between users and the data they make use of. Through this we hope to develop a rich picture of how people and tasks are organised around data, or conversely how the organisation of data influences the structure of organisation.

6.2 Vehicular Traffic Network Analysis

Following the work of applying SNA to generate insights into the structure of a data lake, with a view to discover from a corpus of data what can be used to provide insight into the behaviour of a real-world network. It was a natural progression to consider the case when we have access to the data set that can provide this insight, however the data is limited in some way. The results of this investigation are presented in Chapter 4, where initially we design a methodology, based on common methods in the literature, of cleaning time series data. In the context of this work, we consider the time series data to be Global Positioning System (GPS) based data that describes the travel times on road segments an urban network. The process involves a series of novel algorithms that used to identify and impute missing values. and thus transform the data into a clean and usable data set. The clean input data is used in an exploratory study with a goal to develop insights into the behaviour of traffic flow in an urban network. More specifically, the work involves a series of experiments that are focussed on generating an understanding into the self-similarity, and regularity of traffic flow for individual road segments in the network. This intelligence is then used to model and simulate traffic flow on specific road segments.

Future work on this problem would be associated with addressing the limitations of the data. Although this work covers a larger period of time than other studies on vehicular traffic such as such as [165, 166]. The limitations in our data arise in the coverage across the network. By gaining access to data for more road segments in the network, one could produce a more comprehensive study into the behaviour of traffic flow in an urban network. A further enhancement could be to consider the work in [165, 166] to investigate the propagation of traffic across the network. This could provide valuable intelligence that can be applied to traffic management systems in order to mitigate the impact of traffic in the network. Alternatively, it can be applied to routing algorithms such as Dijkstras to provide a more sophisticated routing mechanism.

6.3 Deterministic Rendezvous in Restricted Graphs

In Chapter 4 we investigated a real-world network using a data-driven approach, where the data used was limited in terms of observations and coverage of the network. In Chapter 5, we consider the challenge of having exposure to increasingly less information about a network, and in particular we demonstrate the difficulty of the challenge by considering the rendezvous problem in a distributed setting. More specifically, we consider an extention of the work in [61], that is the challenge for two anonymous mobile entities (robots) to meet at the same time and point in space. However, in our work we consider the movement of mobile entities (or robots) to be restricted by a combination of the topological properties of the graph and the intrinsic characteristics of robots, thus preventing robots from visiting certain edges in the graph. The robots have an additional constraint, and that is during the traversal between two connected nodes, the eyes of the robot are closed. In this work three models of computation were considered: In the

Edge Monotonic model each mobile entity $X \in \{A, B\}$ has weight w_x and each edge has a weight restriction. This restriction dictates where a robot X can traverse with respect to their intrinsic weight w_x. In the remaining two models of computation the graph is un-weighted and restrictions refer to more arbitrary subsets of traversable nodes and edges. In particular, for the *Node Inclusive* model the set of nodes denoted V_x available to a robot X satisfy the condition $V_A \subset V_B$ or vice versa, and in the *Blind Rendezvous* model the relationship between $V_A and V_B$ is arbitrary.

In this work we consider the following question for each of the computational models, what is the worst case rendezvous time for two anonymous mobile entities in a restricted graph environment? The question is answered for two distinct settings, namely synchronous and asynchronous. In the synchronous setting the mobile entities have access to a global clock that is ticking in discrete time steps $0, 1, 2, \cdots$, and during a single time step each robot assesses the node in which it resides within (checking for co-location and rendezvous). In the asynchronous setting, the robots do not have access to a global clock, instead it is decided by the adversary which of the robots will make a move at every time step. In order to help answer this question for each of the models a number of Algorithms are presented. For the synchronous setting the results are: In the Edge Monotonic model it is shown that the rendezvous between two mobile entities can be achieved in optimal $O(k_A + k_B)-$time. In the Node Inclusive model we present an algorithm that meets two mobile entities in almost linear time $O((k_A+k_B)\log(k_A+k_B))$, and finally we show that rendezvous is not feasible without additional assumptions. In the case that explicit labels are used rendezvous can be achieved in $\min\{O((k_A + k_B)^3 \log\log n, O((k_A + k_B)^2 \log n)\}$. through the super-position of two algorithms. For the asynchronous setting the results are presented in terms of trajectory instead of time, as in the asynchronous case there is no global clock: In the Edge Monotonic model it is shown that rendezvous can be achieve through a trajectory of length $O((k_A + k_B)^2)$. For the Node Inclusive model it is proven that rendezvous can not be achieved without additional assumptions, and without this assumption it is also proven that rendezvous in the Blind Rendezvous model is infeasible.

When thinking of the next step for the research presented in Chapter 5, where we consider the case in which robots have restrictions imposed on where they can go in the network, and these restrictions are based on the intrinsic properties of the robots. The natural progression of this work in order to provide further limitations on knowledge in the network would be to consider models in which maps are unknown to robots, and robots can only discover small regions of the environment that are within some bound r of their current position. Furthermore, the work presented considers the rendezvous of two robots, so it would be interesting to extend this to more than two robots, where the robots can meet in pairs. This problem can be studied in a variety of environments including geometrical, similar to [44], that could vary significantly in the complexity.

6.4 Final Remarks

Over the course of this project we have felt that we have contributed to a new class of problems that are concerned with *Data Mining* and *Distributed Computing*. Through this contribution we have gained an appreciation of the surrounding problems in the *Search and Mobility* and *Data discovery*. It is our hope that the work in this thesis and the literature that it draws from can also spark an interest in the community for *Network Analysis* as there is such a close link to *Data Mining* and *Data Discovery*. Network analysis has been extensively researched over the last few decades, and as demonstrated in this thesis by applying the established techniques from the community, it is likely that many interesting questions can be proposed, especially when looking into the behaviour of network entities, which can lead to significant insights being discovered about phenomena such as vehicular traffic.

Appendix A

Similarity

Similarity measures are essential to discovering patterns or motifs in pattern recognition problems such as classification, clustering etc [37]. This appendix is designed to give the reader insight into two similarity measures that we use within this thesis, these are: Euclidean Distance and Cosine Similarity. The following sections of this appendix describe both measures.

A.1 Squared Euclidean Distance

The Euclidean distance is the distance between two points in Euclidean space. Euclidean distance was referred to as *Pythagorean metric*, where it was derived from the Pythagorean formula. In particular, given two points $P = (p_1, p_2, \ldots, p_n)$, and $Q = (q_1, q_2, \ldots, q_n)$ in n-dimensional Euclidean space, the Euclidean distance between points p and q is the length of the line segment connecting them, and is denoted

$$\begin{aligned} d(P,Q) &= \sqrt{(p_1-q_1)^2+(p_2-q_2)^2+\cdots+(p_n-q_n)^2} \\ &= \sqrt{\sum_{i=1}^{n}(p_i-q_i)^2}. \end{aligned} \tag{A.1}$$

The squared Euclidean distance can be used to place increasingly greater weight on objects that are farther apart, and is denoted

$$\begin{aligned} d(P,Q) &= (p_1-q_1)^2+(p_2-q_2)^2+\cdots+(p_n-q_n)^2 \\ &= \sum_{i=1}^{n}(p_i-q_i)^2. \end{aligned} \tag{A.2}$$

A.2 Cosine Similarity

Cosine similarity is a measure of similarity between two vectors P and Q, that measures the cosine of the angle between them. This similarity metric was derived from the dot (inner) product denoted, $P \cdot Q = ||P||\ ||Q||\ cos\ \theta$, where θ is the measure of the angle between the vectors P and Q. In particular, the cosine similarity $cos(\theta)$ is defined as,

$$\begin{aligned} cos(\theta) &= \frac{P \cdot Q}{||P||\ ||Q||} \\ &= \frac{\sum_{i=1}^{n} p_i \cdot q_i}{\sqrt{\sum_{i=1}^{n} p_i^2}\sqrt{\sum_{i=1}^{n} q_i^2}}. \end{aligned} \tag{A.3}$$

The cos(θ) of $0°$ is 1, and is less than 1 for any other angle. Therefore, this measures orientation that ranges from -1 to 1, suggesting that two vectors

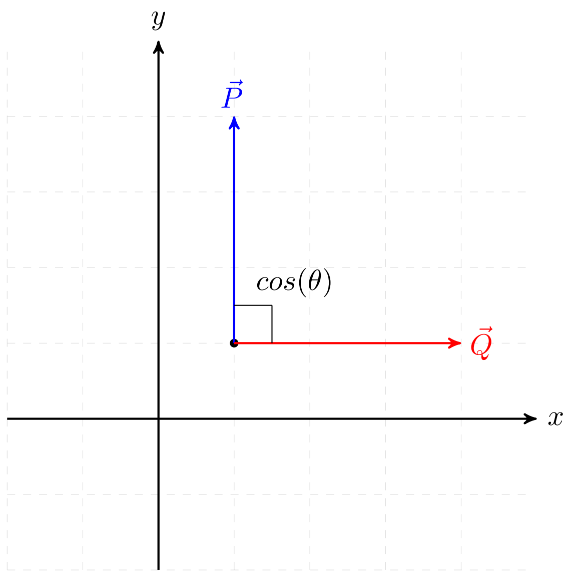


Figure A.1: Cosine Similarity between P and Q, $\cos(\theta) = 0$

with the same orientation have a cosine similarity of 1, two vectors at 90° (orthogonal) have similarity of 0, and two vectors diametrically opposed have a similarity of -1.

Appendix B

Clustering

Clustering is a technique that is used to partition the data into specific classes (*clusters*), such that for each cluster, the data contained within it is most accurately represented by that cluster, and this is determined by the attributes or characteristics of the datum in question. In this thesis we focus on one clustering method in particular and this is known as *k-means*, the following section of this appendix discusses k-means in detail.

B.1 k-means

The *k-means* clustering algorithm is a popular method used in the data mining community to partition N data points in a m-dimensional space into k clusters, and each cluster is parameterised by a vector $m^{(k)}$ called its mean. This results in partitioning the data space into *Voronoi* cells, where each cell represents a region of the plane (data space) based on distance to points (seeds) in a specific subset of that plane. In particular, the points are specified before the algorithm is initialised, and for each Voronoi cell R_k the region in space comprises every point whose distance to p_k is less than or equal to its distance to any other p_k. In the context of clustering, k-means aims to partition N observations x_1, x_2, ..., x_n into $k(\leq n)$ sets $S = \{S_1, S_2, \ldots, S_k\}$, s.t. each datum is closest to its clusters C_k mean, denoted μ_k than the mean of any other cluster C_k. The function that is used to determine the cluster that a given datum x should be assigned to is known as within-cluster sum of squares (WCSS), where the cluster that yields the least WCSS is considered to be the cluster that x belongs to, and is denoted:

$$\arg\min_{s} \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2 \tag{B.1}$$

where μ_i is the mean of point S_i. The most common algorithm to solve the k-means problem was presented in [111], where the authors propose an algorithm for least squares quantisation in pulse-code modulation (PCM). In particular, *Lloyd's algorithm* uses vector quantisation (where the space is divided into Voronoi cells), to identify the cluster centers (c_1, ..., c_k), such that the WCSS minimised. The algorithm presented in the paper is used in a multitude of domains including signal processing. However, more commonly it is used as a heuristic that can be applied to large data sets for cluster analysis, where data is partitioned into k regions. In [141] the authors suggest that the K-means algorithm is a local optimisation strategy, and consequently the solution is sensitive to the choice of initial cluster centers. Common methods of initialising cluster centers are *Forgy* [81] and *Random Partition* [81], where the Forgy method is considered to be the *de facto* technique for initialising cluster centers. It is common practice to apply the initialisation technique within Lloyds algorithm; discussed below, in order to yield clustering of the data.

The *Lloyds algorithm* is defined as: Given an initial set of cluster centers $c_1^{(1)}, c_2^{(1)}, \ldots, c_k^{(1)}$; that are identified using the chosen method, the algorithm proceeds by alternating between two steps 1) *Assignment Step*, and 2) *Update Step*, until the WCCS has converged.

Assignment Step: Assign each datum to the cluster whose population mean μ yields the least WCSS, given that the sum of squares; method used to determine partition sums of squared deviations, is the distance from a datum x to the mean μ is the deviation denoted, $\sum_{i=1}^{n}(x-\mu)^2$, is the squared *Euclidean Distance*, this is intuitively the nearest mean to x.

$$S_i^{(t)} = \{x : \|x - \mu_i^{(t)}\|^2 \leq \|x - \mu_j^{(t)}\|^2 \forall j, 1 \leq j \leq k\} \tag{B.2}$$

where each x is assigned to exactly one $S^{(t)}$, even if it could be assigned to more than one.

Update Step: For each set S_i calculate the new means, and consider this to be the centroid of the observations, this is denoted by

$$\mu_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j \tag{B.3}$$

given it was proven by Gauss, if the conditions of the Gauss-Markov theorem apply then the arithmetic mean is considered optimal, and is therefore the best least-square estimator. Therefore this step also minimises the WCSS, and thus satisfies the objective of the k-means problem.

Furthermore, considering both steps optimise the WCSS objective, and there only exists a finite number of partitioning the data space, the algorithm must converge to a local optimum.[1]

[1] There is no guarantee that the global optimum is found using this algorithm.

Appendix C

Random Walk

A random walk experiment is a process by which randomly-moving objects wander away from their starting position. A simple random walk experiment is conducted in 1-dimensional space. Wlog assume the space is a number line and pointer A (random object) is assigned to position 0. At the point the experiment starts A will take a step forward (or backward) with equal probability p. As the time progresses for the experiment the object will be taking steps forward or backwards, and this continues until the experiment finishes. Random walks are generally used to determine where an object will be in N time steps, and this identified by computing the distance that A is away from the position A is at the previous step in the experiment. This is typically computed using *root-mean squared* distance denoted.

$$RMS(X) = \sqrt{\frac{1}{N}\sum_{i=0}^{N}(x_i)^2} \tag{C.1}$$

Appendix D

Markov Chain

A Markov chain is a stochastic process, or discrete time process, that comprises a set of states $S = \{s_1, s_2, \ldots s_r\}$. At the point the process starts, it begins in one of these states, and for each successive move called a *step* the state moves to another. That is, if the current state is s_i, then it will move to state s_j at the next step with a probability denoted by $p_{i,j}$, where this probability does not depend on which state the chain was in before the current state. The probabilities $p_{i,j}$ are known as *transition probabilities*, and these represent the likelihood that state s_i will move to s_j. The transition probabilities are typically stored in a matrix called a *transition matrix.* The process can also remain in the current state and this is determined based on probability $p_{i,i}$. More formally, for a discrete process to be considered to be Markov, it has to satisfy the Markov property denoted,

$$P(X_n = x_n | X_{n-1} = x_{n-1}, \ldots, X_0 = x_0) = P(X_n = x_n | X_{n-1} = x_{n-1}) \tag{D.1}$$

where X_n is the current state of the process, and X_{n+1} is the previous state of the process, suggesting that current states in the process are dependent on the past state only.

Appendix E

Software

E.1 i-MOVE Platform

E.1.1 Description

The platform for i-MOVE was considered to be the most fundamental component required in the 18-24 months that the project was live. The platform was designed to be an interoperable information hub that provided functionality to provide access to a multitude of data feeds describing the transportation network in and around Liverpool.

Adapting to changing requirements is one of the predominant challenges in distributed environments, by applying the eco-system concepts into a digital environment, we were able to create a versatile domain that adopted Service Oriented Architecture (SOA) [67]. SOA is an architectural pattern that comprises four main characteristics, however in this project we focus on only 2 of these, and these include: 1) vendor neutral - helps evolve the solution in response to changing requirements, and 2) composition centric - enables the system to adapt to new and changing requirements within a reduced amount of time, by reusing system features e.g., web services. This architectural pattern is based on the Feature-Driven Development (FDD) methodology. FDD considers system features to be separate entities that are specified, created and tested in an isolated environment using established testing methods e.g., unit, regression etc, before including them into the spine of the system. i-MOVE adopted this architectural pattern to ensure that web services are considered to be individual system elements so that risk management and testing can be performed in isolation, preventing the risk of system outages.

SOA allows web-services to be created easily and efficiently by reusing pre-implemented components. In the development phases of the project, i-MOVE core was developed so that developers can reuse existing components to easily create new system features e.g., web services.

E.1.2 Architecture

Typical SOA suggests that system components are connected over a network, forming a distributed system. i-MOVE follows this recommendation where the core system functionality was implemented and located on a master server, the web services were created on a slave server, and when a request is made to the core it communicates to the slave in order to answer the request.

The platform inherits server roles from SOA-web services, these roles are: *Service Provider* and *Service Registry*. The service provider role creates a web-service, publishes its interface and accesses information from the service registry, where the service registry will handle accessibility to the service for a specific user through an authentication service. At the point a service is implemented the service registry will allocate a category to the service e.g., road, rail, weather etc., decide what agreements are needed to use this service e.g., terms and conditions, and the privacy policy e.g., public or private. Furthermore, the service registry will expose the catalogue of web services that are available in the platform; providing basic descriptions, accessibility instructions, source, data format etc.

The service registry is an implementation of the the HyperCat [125] standard. It is based upon basic HTTP principles to allow the secure distribution of data assets, and handles individual transactions using user-centric access control between multiple information hubs using RESTful services, thus creating an interoperable platform. Completely dependent on user permission a connected user can make GET, PUT and UPDATE requests to add, update or remove data assets from a specific hub respectively. The idea behind the interoperability feature is to allow information hubs to search within their local service registry for a specific type of service, before searching all other service registries within the distributed network of information hubs, or any other hub that adopts the HyperCat standard [1]. This feature allows Machine to Machine (M2M) interaction between hubs, allowing them to freely search other hubs for a specific asset, and return it to the user as though that particular hub contains such asset. Furthermore, the HyperCat standard specifies a security layer for information platforms by restricting access to data assets to authorised users only.

In order for users to understand the catalogue of assets contained within

[1]The catalogue can be used by other information hubs that adopt the HyperCat standard to automatically identify the list of services available on that particular hub. A specific hub can store multiple hub addresses, and can use these addresses to perform query-based searches across multiple information hubs simultaneously in order to identify the desired data feed quickly.

a specific hubs service registry, a user would need to direct to the catalogue service that will appear on `http://{hub-domain}/cat`. This will present the user with a JSON formatted document that will describe a list of the data assets available, and provide information about each individual asset, including a link to the developer API; providing instructions on how to access that particular asset, and what the response means. Once a user has identified a data asset that they wish to consume, they can access it through `http://imove-project.org/api/{feedid}`, where the feedid is provided in the catalogue.

Bibliography

[1] A.Agresti and B.Finlay (1997). *Statistical Methods for the Social Sciences*. Prentice Hall.

[2] Agathangelou, C., Georgiou, C., and Mavronicolas, M. (2012). A distributed algorithm for gathering many fat mobile robots in the plane. *CoRR*, abs/1209.3904.

[3] Aggarwal, C. C. (2015). *Data mining: The Textbook*. Springer International Publishing Switzerland 2015, IBM T.J. Watson Research Center, Yorktown Heights, New York, USA.

[4] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[5] Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., and Cayirci, E. (2002). A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114.

[6] Albert, R. and Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Rev. Mod. Phys.*

[7] Alpaydin, E. (2010). *Introduction to Machine Learning*. The MIT Press, 2nd edition.

[8] Alpern, S. (1995). The rendezvous search problem. *SIAM Journal on Control and Optimization*, 33(3):673–683.

[9] Alpern, S. (2002). Rendezvous search on labeled networks. *Naval Research Logistics (NRL)*, 49(3):256–274.

[10] Alpern, S., Fokkink, R., Gasieniec, L., Lindelauf, R., and Subrahmanian, V. S. (2013). *Search theory*. Springer-Verlag New York.

[11] Alpern, S. and Gal, S. (2003). *The theory of search games and rendezvous*. Kluwer Academic Publishers.

[12] Alzahrani, T. and Horadam, K. J. (2016). *Community Detection in Bipartite Networks: Algorithms and Case studies*, pages 25–50. Springer Berlin Heidelberg, Berlin, Heidelberg.

[13] Anderberg, M. R. (1973). Cluster analysis for applications. In Anderberg, M. R., editor, *Cluster Analysis for Applications*, Probability and Mathematical Statistics: A Series of Monographs and Textbooks. Academic Press.

[14] Anderson, E. J. and Fekete, S. P. (1998). Asymmetric rendezvous on the plane. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*, SCG '98, pages 365–373, New York, NY, USA. ACM.

[15] Anderson, E. J. and Fekete, S. P. (2001). Two dimensional rendezvous search. *Operations Research*, 49(1):107–118.

[16] Anderson, E. J. and Weber, R. R. (1990). The rendezvous problem on discrete locations. *Journal of Applied Probability*, 27:839–851.

[17] Anderson, T. W. and Sclove, S. (1974). *Introductory Statistical Analysis*. Houghton Mifflin Company.

[18] André, C. and Pinheiro, R. (2012). Community detection to identify fraud events in telecommunications networks.

[19] Apache. Apache hadoop. `http://hadoop.apache.org`. Accessed: 22-05-2016.

[20] Apache. Hive. `http://hive.apache.org/`. Accessed: 08-10-2015.

[21] Babu, S. (2010). Towards automatic optimization of mapreduce programs. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 137–142, New York, NY, USA. ACM.

[22] Barabási, A. L. and Oltvai, Z. N. (2004). Network Biology: Understanding the Cell's Functional Organization. *Nature Genetics*, 5:101–114.

[23] Bastian, M., Heymann, S., and Jacomy, M. (2009). Gephi: An open source software for exploring and manipulating networks. In *Proceedings of the International AAAI Conference on Web and Social Media*.

[24] Baston, V. and Gal, S. (1998). Rendezvous on the line when the players' initial distance is given by an unknown probability distribution. *SIAM Journal on Control and Optimization*, 36(6):1880–1889.

[25] Baston, V. and Gal, S. (2001). Rendezvous search when marks are left at the starting points. *Naval Research Logistics (NRL)*, 48(8):722–731.

[26] BBC. Talk talk data breach. `http://www.bbc.co.uk/news/business-37565367`. Accessed: 08-10-2016.

[27] Becchi, M. From poisson processes to self-similarity: a survey of network traffic models.

[28] Berners-Lee, T. Information management: A proposal. `https://www.w3.org/History/1989/proposal.html`. Accessed: 06-10-2016.

[29] Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*, 284(5):34–43.

[30] Berry, D. S. and Belmont, D. M. (1951). Distribution of vehicle speeds and travel times. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 589–602, Berkeley, Calif. University of California Press.

[31] Blondel, V. D., loup Guillaume, J., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks.

[32] Boccaletti, S., Latora, V., Moreno, Y., Chavez, M., and Hwang, D.-U. (2006). Complex networks: Structure and dynamics. *Physics Reports*, 424(4?5):175 – 308.

[33] Bóna, M. (2006). *A Walk Through Combinatorics: An Introduction to Enumeration and Graph Theory*. World Scientific Pub.

[34] Bulkley, N. and Alstyne, M. V. (2006). An empirical analysis of strategies and efficiencies in social networks.

[35] Cantini, L., Medico, E., Fortunato, S., and Caselle, M. (2015). Detection of gene communities in multi-networks reveals cancer drivers. *Scientific Reports*, 5:17386+.

[36] Catapult, T. S. Transport systems catapult. `https://ts.catapult.org.uk`. Accessed: 28-08-2016.

[37] Cha, S.-H. (2007). Comprehensive survey on distance/similarity measures between probability density functions.

[38] Chen, S., Russell, A., Samanta, A., and Sundaram, R. (2014). Deterministic blind rendezvous in cognitive radio networks. *CoRR*, abs/1401.7313.

[39] Cheng, T., Haworth, J., and Manley, E. Markov chain topological route selection. `http://www.geocomputation.org/2013/papers/80.pdf`. Accessed: 06-10-2015.

[40] Clauset, A., Newman, M. E. J., and Moore, C. (2004). Finding community structure in very large networks. *Phys. Rev. E*, 70:066111.

[41] Cloudera. Impala. `http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html`. Accessed: 08-10-2015.

[42] cnet. Virgin mobile voicemail madness: privacy breach as customers' gain access to strangers' messages. `https://www.cnet.com/au/news/virgin-mobile-australia-voicemail-error-privacy/`. Accessed: 08-10-2016.

[43] Collins, A., Czyzowicz, J., Gasieniec, L., Kosowski, A., and Martin, R. (2011). Synchronous rendezvous for location-aware agents. In *Proceedings of the 25th International Conference on Distributed Computing*, DISC'11, pages 447–459, Berlin, Heidelberg. Springer-Verlag.

[44] Collins, A., Czyzowicz, J., Gasieniec, L., and Labourel, A. (2010). *Tell Me Where I Am So I Can Meet You Sooner*, pages 502–514. Springer Berlin Heidelberg, Berlin, Heidelberg.

[45] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition.

[46] Cortes, C., Pregibon, D., and Volinsky, C. (2001). *Communities of Interest*, pages 105–114. Springer Berlin Heidelberg, Berlin, Heidelberg.

[47] Cox, J. Apple, intel, google employee accounts exposed in data breach of developer forum. `http://motherboard.vice.com/read/apple-intel-google-employee-accounts-exposed-in-data-breach`. Accessed: 08-10-2016.

[48] Crovella, M. E. and Bestavros, A. (1996). Self-similarity in world wide web traffic: Evidence and possible causes.

[49] Csardi, G. and Nepusz, T. (2006). The igraph software package for complex network research. *InterJournal*, Complex Systems:1695.

[50] Czyzowicz, J., Gasieniec, L., and Pelc, A. (2009). Gathering few fat mobile robots in the plane. *Theor. Comput. Sci.*, 410(6-7):481–499.

[51] Czyzowicz, J., Kosowski, A., and Pelc, A. (2012). How to meet when you forget: log-space rendezvous in arbitrary graphs. *Distributed Computing*, 25(2):165–178.

[52] Czyzowicz, J., Labourel, A., and Pelc, A. (2010). How to meet asynchronously (almost) everywhere. *CoRR*, abs/1001.0890.

[53] de Solla Price, D. J. (1965). Networks of scientific papers. *Science*, 149(3683):510–515.

[54] Dean, J. and Ghemawat, S. (2008). Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113.

[55] deeplearning.net. Convolutional neural networks (lenet). `http://deeplearning.net/tutorial/lenet.html`. Accessed: 22-05-2016.

[56] Degener, B., Kempkes, B., and auf der Heide, F. M. (2010). A local o(n2) gathering algorithm. In *Proceedings of the Twenty-second Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '10, pages 217–223, New York, NY, USA. ACM.

[57] Degenne, A. and Forsé, M. (1999). *Introducing Social Networks*. ISM (London, England). SAGE Publications.

[58] Demaine, E. (2007). Advanced data structures - lecture 5. `http://courses.csail.mit.edu/6.851/spring07/scribe/lec05.pdf`. Accessed: 10-07-2016.

[59] Deng, J., Dong, W., Socher, R., jia Li, L., Li, K., and Fei-fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *In CVPR*.

[60] Dereniowski, D., Klasing, R., Kosowski, A., and Kuszner, L. (2014). Rendezvous of heterogeneous mobile agents in edge-weighted networks. *CoRR*, abs/1406.2008.

[61] Dessmark, A., Fraigniaud, P., Kowalski, R. D., and Pelc, A. (2006). Deterministic rendezvous in graphs. *Algorithmica*, 46(1):69–96.

[62] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271.

[63] D.Napoleon and S.Pavalakodi (2011). A new method for dimensionality reduction using k-means clustering algorithm for high dimensional data set. *International Journal of Computer Applications*, 13(8):41–46.

[64] Driver and (DVLA), V. L. A. How many people hold licences in the uk. `https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/397430/FOIR4341_How_many_people_hold_licences_in_the_UK.pdf`. Accessed: 10-10-2016.

[65] Duch, J. and Arenas, A. (2005). Community detection in complex networks using extremal optimization. *Phys. Rev. E*, 72:027104.

[66] Egmont-Petersen, M., de Ridder, D., and Handels, H. (2002). Image processing with neural networks?a review. *Pattern Recognition*, 35(10):2279 – 2301.

[67] Erl, T. (2004). *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[68] Farrugia, A. (2013). i-Move Eco-system demonstrator project. `http://imove-project.org`.

[69] Farrugia, A., Gasieniec, L., Kuszner, Ł., and Pacheco, E. (2015). *Deterministic Rendezvous in Restricted Graphs*, pages 189–200. Springer Berlin Heidelberg, Berlin, Heidelberg.

[70] Ferrara, E., Meo, P. D., Catanese, S., and Fiumara, G. (2014). Detecting criminal organizations in mobile phone networks. *CoRR*, abs/1404.1295.

[71] for National Statistics, O. Annual survey of hours and earnings: 2015. `http://www.ons.gov.uk/employmentandlabourmarket/peopleinwork/earningsandworkinghours/bulletins/annualsurveyofhoursandearnings/2015provisionalresults`. Accessed: 10-10-2016.

[72] Fortunato, S. (2010). Community detection in graphs. *Physics Reports*, 486:75 – 174.

[73] Fortunato, S. and Barthelemy, M. (2007). Resolution limit in community detection. *Proceedings of the National Academy of Sciences.*

[74] Foundation, R. The r project for statistical computing. `https://www.r-project.org/`. Accessed: 21-05-2015.

[75] Freeman, L. C. (1977). A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41.

[76] Freund, J. E. (2001). *Modern elementary statistics.* Prentice-Hall.

[77] Google. Tensorflow. `https://www.tensorflow.org`. Accessed: 22-05-2016.

[78] Greenshields, B. D., Thompson, J. T., Dickinson, H. C., and Swinton, R. S. (1933). The Photographic Method of Studying Traffic Behavior. In *The Photographic Method of Studying Traffic Behavior*, volume 13, pages 382–399.

[79] Gross, J. L. and Yellen, J. (2005). *Graph Theory and Its Applications, Second Edition (Discrete Mathematics and Its Applications).* Chapman & Hall/CRC.

[80] Gutman, R. (2004). Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. In *Proceedings 6th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 100–111. SIAM.

[81] Hamerly, G. and Elkan, C. (2002). Alternatives to the k-means algorithm that find better clusterings. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, CIKM '02, pages 600–607, New York, NY, USA. ACM.

[82] Harary, F. (1995). *Graph Theory.* Addison-Wesley, Cambridge, MA.

[83] Haworth, J. (2013). *Spatio-temporal forecasting of network data.* PhD thesis, University College London (UCL).

[84] Heimbigner, D. and McLeod, D. (1985). A federated architecture for information management. *ACM Trans. Inf. Syst.*, 3(3):253–278.

[85] Heistracher, T., Kurz, T., Marcon, G., and Masuch, C. (2006). Collaborative software engineering with a digital ecosystem. In *IEEE International Conference on Global Software Engineering.*

[86] Holme, P. and Saramäki, J. (2012). Temporal networks. *Physics Reports*, 519(3):97 – 125. Temporal Networks.

[87] Horvitz, E., Apacible, J., Sarin, R., and Liao, L. (2012). Prediction, expectation, and surprise: Methods, designs, and study of a deployed traffic forecasting service. *CoRR*, abs/1207.1352.

[88] Hunt, P., Robertson, D., and Bretherton, R. (1985). Scoot - a traffic responsive method of coordinating signals.

[89] Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science Engineering*, 9(3):90–95.

[90] INRIX. Economic & environmental impact of traffic congesting in europe and the us. `http://inrix.com/economic-environment-cost-congestion/`. Accessed: 28-08-2016.

[91] Inrix. Inrix - driving intelligence. `http://inrix.com/`. Accessed: 01-09-2014.

[92] Jain, S., Fall, K., and Patra, R. (2004). Routing in a delay tolerant network. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04, pages 145–158, New York, NY, USA. ACM.

[93] Jaskowiak, P. A., Campello, R. J., and Costa, I. G. (2014). On the selection of appropriate distances for gene expression data clustering. *BMC Bioinformatics*, 15(2):S2.

[94] Johnson, R. and Bhattacharyya, G. (1992). *Statistics: Principles and Methods*. Wiley.

[95] Johnson, R. E. and Schneider, F. B. (1985). Symmetry and similarity in distributed systems. In *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing*, PODC '85, pages 13–22, New York, NY, USA. ACM.

[96] Jones, E., Oliphant, T., Peterson, P., et al. (2001). SciPy: Open source scientific tools for Python. [Online; accessed 2016-09-03].

[97] Jousselme, A.-L. and Maupin, P. (2012). Distances in evidence theory: Comprehensive survey and generalizations. *International Journal of Approximate Reasoning*, 53(2):118 – 145. Theory of Belief Functions (BELIEF 2010).

[98] Kallas, S. (2012). Smart Cities and Communities European Innovation Partnership (SCC). `http://www.connectedliverpool.co.uk/blog/tag/congestion/`. [Online; accessed 28-june-2013].

[99] Kernighan, B. W. and Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307.

[100] Kopetz, H. (2011). *Internet of Things*, pages 307–323. Springer US, Boston, MA.

[101] Kowalski, D. R. and Malinowski, A. (2008). How to meet in anonymous network. *Theor. Comput. Sci.*, 399(1-2):141–156.

[102] Kranakis, E., Krizanc, D., and Rajsbaum, S. (2006). *Mobile Agent Rendezvous: A Survey*, pages 1–9. Springer Berlin Heidelberg, Berlin, Heidelberg.

[103] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

[104] Krogh, B., Andersen, O., and Torp, K. (2012). Trajectories for novel and detailed traffic information. In *Proceedings of the Third ACM SIGSPATIAL International Workshop on GeoStreaming*, IWGS '12, pages 32–39, New York, NY, USA. ACM.

[105] Leduc, G. (2008). Jrc technical notes: Programming according to the fences and gates model for developing assured, secure software systems. Technical Report JRC 47967, Institute for Prospective Technological Studies, Edificio Expo. c/ Inca Garcilaso, s/n. E-41092 Seville, Spain.

[106] Leland, W. E., Taqqu, M. S., Willinger, W., and Wilson, D. V. (1994). On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Trans. Netw.*, 2(1):1–15.

[107] Li, W., Shiyin, P., Yongzhong, Z., and Zhengxi, L. (2010). Urban traffic complex network hub node analysis and signal control optimization strategy research. In *Control and Decision Conference (CCDC), 2010 Chinese*, pages 4172–4175.

[108] Lin, Z., Liu, H., Chu, X., and Leung, Y. W. (2011). Jump-stay based channel-hopping algorithm with guaranteed rendezvous for cognitive radio networks. In *INFOCOM, 2011 Proceedings IEEE*, pages 2444–2452.

[109] Linial, N. (1992). Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201.

[110] Liu, W., Zheng, Y., Chawla, S., Yuan, J., and Xing, X. (2011). Discovering spatio-temporal causal interactions in traffic data streams. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 1010–1018, New York, NY, USA. ACM.

[111] Lloyd, S. (1982). Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137.

[112] Lovász, L. (1993). Random walks on graphs: A survey.

[113] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150–1157 vol.2.

[114] Lynch, N. A. (1996). *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[115] Machay, J. (2013). How does google detect traffic congestion? `http://smallbusiness.chron.com/google-detect-traffic-congestion-49523.html`. Accessed: 2016-05-01.

[116] Macqueen, J. B. (1967). Some Methods for classification and analysis of multivariate observations. In *Procedings of the Fifth Berkeley Symposium on Math, Statistics, and Probability*, volume 1, pages 281–297. University of California Press.

[117] Mandelbrot, B, B. (2008). How fractals can explain what's wrong with wall street. `http://www.scientificamerican.com/article/multifractals-explain-wall-street/`.

[118] Manley, E., Cheng, T., and Haworth, J. (2013). Markov chain topological route selection. `http://www.geocomputation.org/2013/papers/80.pdf`. Accessed: 18-05-2016.

[119] McDonald, A. B. (1997). Survey of adaptive shortest-path routing in dynamic packet-switched networks.

[120] McKinney, W. (2010). Data structures for statistical computing in python. In van der Walt, S. and Millman, J., editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56.

[121] McQuillan, J. M. (1974). *Adaptive Routing Algorithms for Distributed Computer Networks*. PhD thesis, Harvard University.

[122] Metoffice. Met office - datapoint. `http://www.metoffice.gov.uk/datapoint`. Accessed: 01-09-2014.

[123] M.Frechet (1906). Sur quelques points du calcul fonctionne. *Rendiconti del Circolo Mathematico di Palermo*.

[124] Miller, A. and Pelc, A. (2014). Time versus cost tradeoffs for deterministic rendezvous in networks. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, pages 282–290, New York, NY, USA. ACM.

[125] Monnickendam, N. Hypercat. `http://www.hypercat.io/`. Accessed: 02-08-2016.

[126] Murthy, S. (1996). *Routing in packet-switched networks using path-finding algorithms*. PhD thesis, University of California, Santa Cruz.

[127] Nagurney, A. (1999). *Network Economics*. Springer US.

[128] Nevatia, R. and Babu, K. R. (1980). Linear feature extraction and description. *Computer Graphics and Image Processing*, 13(3):257 – 269.

[129] Newman, M., Barabasi, A.-L., and Watts, D. J. (2006). *The Structure and Dynamics of Networks: (Princeton Studies in Complexity)*. Princeton University Press, Princeton, NJ, USA.

[130] Newman, M. E. J. (2003). The structure and function of complex networks. *SIAM Review*, 45(2):167–256.

[131] Newman, M. E. J. and Girvan, M. (2004). Finding and evaluating community structure in networks. *Phys. Rev. E*, 69:026113.

[132] Olston, C., Reed, B., Srivastava, U., Kumar, R., and Tomkins, A. (2008). Pig latin: A not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1099–1110, New York, NY, USA. ACM.

[133] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[134] Pelc, A. (2012). Deterministic rendezvous in networks: A comprehensive survey. *Netw.*, 59(3):331–347.

[135] Pinheiro, C. A. R. Community detection to identify fraud events in telecommunications networks. `http://support.sas.com/resources/papers/proceedings12/106-2012.pdf`. Accessed: 20-10-2016.

[136] Pons, P. and Latapy, M. (2004). Computing communities in large networks using random walks. *J. of Graph Alg. and App. bf*, 10:284–293.

[137] Python. Python. `https://www.python.org/`. Accessed: 02-08-2016.

[138] Rahm, E. and Do, H. H. (2000). Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23:2000.

[139] Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., and Keogh, E. (2012). Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 262–270, New York, NY, USA. ACM.

[140] Ranzato, M., Hinton, G., and LeCun, Y. (2015). Guest editorial: Deep learning. *International Journal of Computer Vision*, 113(1):1–2.

[141] Redmond, S. J. and Heneghan, C. (2007). A method for initialising the k-means clustering algorithm using kd-trees. *Pattern Recognition Letters*, 28(8):965 – 973.

[142] Rossum, G. (1995). Python reference manual. Technical report, Amsterdam, The Netherlands, The Netherlands.

[143] RStudio, Inc (2014). shiny: Easy web applications in r. URL: `http://shiny.rstudio.com`.

[144] Ruohonen, K. (2013). *Graph Theory*. 1 edition.

[145] Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach (3rd Edition)*. Prentice Hall, 3 edition.

[146] Salathé, M. and Jones, J. H. (2010). Dynamics and Control of Diseases in Networks with Community Structure. *PLoS Comput Biol*, 6(4):e1000736+.

[147] Schelling, T. C. (1960). The strategy of conflict.

[148] Scott, J. (2000). *Social Network Analysis: A Handbook*. Sage Publications, second. edition.

[149] Shlens, J. (2014). A tutorial on principal component analysis. *CoRR*, abs/1404.1100.

[150] Siemens. Siemens - urban traffic control system. `http://www.siemens.co.uk/traffic/en/index/productssolutionsservices/systems/pcscoot.htm`. Accessed: 01-09-2014.

[151] Slaney, M. and Casey, M. (2008). Locality-sensitive hashing for finding nearest neighbors [lecture notes]. *IEEE Signal Processing Magazine*, 25(2):128–131.

[152] Sowa, J. F., editor (1991). *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. The Morgan Kaufmann Series in Representation and Reasoning. Morgan Kaufmann.

[153] Steen, M. v. (2010). *"Graph theory and complex networks : an introduction"*. Maarten van Steen, "Lexington".

[154] Tenenbaum, J. B., Silva, V. d., and Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323.

[155] Thieurmel, B. Introduction to visnetwork. `https://cran.r-project.org/web/packages/visNetwork/vignettes/Introduction-to-visNetwork.html`. Accessed: 01-05-2016.

[156] Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., and Murthy, R. (2009). Hive: A warehousing solution over a map-reduce framework. *Proc. VLDB Endow.*, 2(2):1626–1629.

[157] Thusoo, A., Shao, Z., Anthony, S., Borthakur, D., Jain, N., Sen Sarma, J., Murthy, R., and Liu, H. (2010). Data warehousing and analytics infrastructure at facebook. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 1013–1020, New York, NY, USA. ACM.

[158] Treiber, M. and Kesting, A. (2013). *Traffic Flow Dynamics*. Springer-Verlag Berlin Heidelberg 2013.

[159] Trosset, M. W. (2008). Representing clusters: K-means clustering, self-organizing maps, and multidimensional scaling. Technical report, Indiana University, Bloomington, IN.

[160] UK, I. Internet of things ecosystem demonstrator. `https://connect.innovateuk.org/web/internet-of-things-ecosystem-demonstrator/article-view/-/blogs/the-list-of-8-internet-of-things-clusters?ns_33_redirect=/web/internet-of-things-ecosystem-demonstrator/articles`. Accessed: 01-08-2016.

[161] van der Walt, S., Colbert, S. C., and Varoquaux, G. (2011). The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22–30.

[162] van Lint, J., Hoogendoorn, S., and van Zuylen, H. (2005). Accurate freeway travel time prediction with state-space neural networks under missing data. *Transportation Research Part C: Emerging Technologies*, 13(5?6):347 – 369.

[163] Vassiliadis, P. and Simitsis, A. (2009). *Extraction, Transformation, and Loading*, pages 1095–1101. Springer US, Boston, MA.

[164] Wagner, S. and Wagner, D. (2007). Comparing clusterings- an overview.

[165] Wang, Z., Lu, M., Yuan, X., Zhang, J., and Wetering, H. V. D. (2013). Visual traffic jam analysis based on trajectory data. *IEEE Trans. Vis. Comput. Graphics*, pages 2159–2168.

[166] Wang, Z., Ye, T., Lu, M., Yuan, X., Qu, H., Yuan, J., and Wu, Q. (2014). Visual exploration of sparse traffic trajectory data. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1813–1822.

[167] Wasilewski, P. and Góra, P. (2014). Traffic-related knowledge acquired by interaction with experts. In *CSP*.

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

[169] Weiss, A. N. (1999). *Introductory Statistics*. Addison Wesley.

[170] West, D. B. (2001). *Introduction to Graph Theory (2nd Edition)*. Prentice Hall.

[171] White, T. (2009). *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition.

[172] Witten, I. H., Frank, E., and Hall, M. A. (2011). *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition.

[173] Yegnanarayana, B. (2004). *Artificial Neural Networks.* Prentice-Hall of India Pvt.Ltd.