Efficient and Effective Methodologies for Exploring and Prediction Movement Patterns in Large Networks

Thesis submitted in accordance with the requirements of
the University of Liverpool for the degree of Doctor in Philosophy
by

Mohammed AL-Zeyadi

October 2018

Contents

Illustrations

List of Figures

List of Tables

Notations

The following abbreviations are found throughout this thesis:

List Of Abbreviations:	
A_E	Edge attribute set
A_V	Vertex attribute set
D	Dataset
E	Edge Attribute set
F	From Attribute set
FET	A From, Edge, To tuple describing a MP
G	Graph
MP	Movemnet Pattern
MPM	Movement Pattern Mining
T	To Attribute set
V_E	Set of edge attribute values
V_V	Set of vertex attribute values

Declaration

I hereby certify that this dissertation constitutes my own product, that where the language of others is set forth, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions or writings of another. I declare that the dissertation describes original work that has not previously been presented for the award of any other degree of any institution.

Signed,

Moahmmed Al-Zeyadi

Abstract

In the era of Big Data the prevalence of networks of all kinds has grown dramatically, and analysing (mining) such networks to support decision-making processes has become an extremely important subject for research, typically with a view to some social and/or economic gain. This thesis describes research work within the theme of Movement Pattern Mining (MPM) as applied to large network data. MPM is a type of frequent pattern mining that provides observation into how information is exchanged between objects in large networks. In the context of the work described in this thesis, the focus is on how the concept of Movement Patterns (MPs) can be extracted from large networks efficiently and effectively, and how such movement patterns can best be utilised so as to predict future movement. The work describes how, by utilising big data facilities like Share/Distribute Memory Systems and Hadoop/MapReduce, novel data mining based techniques can be used, not only to extract MPs from large networks, but also how they can be utilised for prediction purposes. To this end, the works in this thesis are divided into two parts. The first part is concerned with an investigation of an efficient mechanism for MPM. The second part is concerned with the utilisation of the extracted MPs in the context of prediction. For evaluation purposes, two large network datasets were used: The Great Britain Cattle Tracking System database and the Jiayuan Social Network. The evaluation indicates that an efficient and effective mechanism for identifying and extracting MPs form large networks, and subsequently using then MPs for prediction purposes, has been established.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my wonderful supervisor Prof. Frans Coenen, for the continuous support of my PhD study and related research, patience, motivation, and immense knowledge. His guidance helped me through of my research and whilst writing this thesis. I could not have imagined having a better advisor and mentor for my PhD study. I also would like to express my appreciation for my second supervisor, Dr. Alexei Lisitsa, for his support, suggestions and valuable comments.

My thanks also go to Mr. David Shield. David has been very helpful in providing technical support, many times, during my research work. Also, I would like to thank my colleague Dr. Muhammad Tufail for his valuable suggestions.

I also would like to extend my gratitude and appreciation to the Ministry of Higher Education and Scientific Research in Iraq and The University of Al-Qadisiyah for the financial support they have given me the opportunity to undertake this PhD programme of research.

Last but not the least, I would like to thank my parents and all my family members. They helped me to pass through the challenging time of my life. Without their continuous encouragement and help, I would never have been able to reach the place where I am standing!

1 Introduction

Over the last decade “big network analysis” has attracted considerable attention because of the large amount of network data that is increasingly generated by a great variety of application domains: social networks [67], computer networks [20], Peer-to-Peer Networks [26], road traffic networks [36] and so on. Networks, by definition, facilitate the exchanging of information between objects. Networks can feature the virtual exchange of information as in the case of friendship communications in social networks; or can be physical networks, as in the case of transportation networks, where the information is goods moved between geographic locations. Whatever the case, such networks can be represented as graphs where the vertices represent senders and/or receivers of information, and the edges represent the information (traffic) flow. The analysis of such networks can take different forms, for example, we might wish to identify communities [83] or “influencers” [47] within such networks. In this thesis, the focus is on the analysis of how information moves through networks. More specifically the identification of the “movement patterns” that may exist in such networks; and how such patterns, when discovered, can be deployed.

Regardless of the nature of the analysis to be conducted, as the size of the networks we wish to analyse continues to increase, the established (traditional) analysis techniques have become increasingly unsuitable. One solution is to adopt a "big data" solution. Big data tools have shown to have great potential with respect to many real-world fields of application such as manufacturing, healthcare, finance, insurance, and retail. The work presented in this thesis is also directed at large networks; and, in the context of information movement patterns, examines how the tools of big data can be utilised with respect to movement analysis. Two specific tools are considered: Hadoop (comprised of MapReduce and the Hadoop Distributed File System) and the Message Pass Interface (MPI).

From the foregoing, the central research theme of this thesis is the discovery of movement patterns in large networks and their subsequent utilisation. Movement Patterns, as defined in this thesis, are three-part patterns, describing information flow, that occur frequently in a given network. The three parts comprise of From, Edge and To parts; in this thesis, the acronym FET is used to describe this data format. Each part in turn consists of a set of one or more attribute values. As such movement patterns, as conceived of in this thesis, can be thought of as a special form of a frequent itemset as popularised in the work on Frequent Pattern Mining (FPM) originating in the early 1990s [6].

The main distinction between movement patterns and traditional frequent patterns is that movement patterns are more prescriptive, as will become clear later in this thesis. Note that the term "movement pattern" as used in this thesis should not be confused with the way the term is used in the context of video surveillance of individuals, animals or road traffic.

The rest of this introductory chapter is organized as follows. In Section 1.1 the motivation for the research is discussed. Section 1.3 presents the research question and associated issues. The adopted research methodology is then presented in Section 1.4, together with the criteria used to evaluate the research outcomes and whether the achieved results are accurate and obtained in an efficacy and effective manner. The research contribution is presented in Section 1.5 followed by the organization of the thesis in Section1.6. The details of publications that have resulted from the work are given in Section 1.7. Finally, in Section 1.8, the chapter is concluded with a brief summary and a "look ahead".

1.1 Research Motivation

In the era of big data the prevalence of networks of all kinds has grown dramatically, and analysing (mining) such networks to support decision-making processes has become an extremely important subject for research, typically with a view to some social and/or economic gain. The challenge of analysing networks is how to process such networks so as to extract meaningful information (knowledge) in an efficient and effective manner. The basic motivation for the research presented in this thesis is the demand for knowledge relating to the behavior of traffic in large networks. Thus the need for techniques to predict the nature of "traffic" movements in networks. In particular the usage of the concept of movement patterns describing the traffic (communication) between vertices (nodes) in networks. Two examples domains where movement patterns are relevant, and those used as a focus with respect to the work presented in this thesis, are: (i) the GB cattle movement database and (ii) the Chinese Jiayuan Social Network. It is suggested that movement patterns extracted from the GB cattle movement network will provide significant knowledge for policy and decision makers who wish to monitor and address issues such as the future expectation of cattle movement; for example in the context of cattle disease. In the case of the Chinese Jiayuan Social Network, traffic movement patterns can be used to periodically make recommendations to existing and new users.

Given the increasing size of the networks that we wish to process alternative techniques to established mechanisms are required. Thus the motivation for the research presented in this thesis is not only the desire to analyse traffic movement in networks but to do this in the context of large networks. More specifically the need to utilize big data techniques so that large networks can be analysed in this manner. Both the GB cattle movement and the Chinese Jiayuan Social Network are examples of large networks.

1.2 Formal Definitions

Regardless of the precise nature of a network, the networks considered in this thesis adhere to the formalism presented in this section. Any given network G is defined in terms of a tuple of the form $\langle V, E\rangle$, where V is a set of vertices and E is a set of edges [33]. The vertices can represent individuals (as in the case of the Jiayuan Social Network), inanimate entities (as in the case of, say, computer networks) or locations (as in the case of CTS networks). The edges then indicate connections between vertices (virtual or actual). These edges might be indicative of some relationship, such as a friend relationship, as in the case of social networks; or a "hard" connection as in the case of a wired computer network or a road traffic network. There is a slightly different in interpretation between the terminology of Networks and Graphs. The term network is typically used in the case of transporting/sending "things" along the links between nodes, whether those things are physical objects (road networks and rail networks) or information (computer networks and social networks). However, in Graphs, the edges typically represent types of relationships between the vertices, for instance as in case of "interest graphs" [80] where the vertices are people and topics, and each edge links a person to a topic that they are interested in.

A Movement Pattern (MP) is defined as a form of Knowledge extracted from a given network G and this knowledge is represented as a tuple of the form $\langle F, E, T\rangle$

where F, E and T are sets of attribute values. The minimum number of attribute values in each part must be at least one; none of the sets can be empty.

1.3 Research Issues and Question

From the foregoing, the central research question that this thesis seeks to address is:

"How can the concept of movement patterns, as envisaged in this thesis, be efficiently and effectively extracted from large networks, and how can those movement patterns best be utilised"

The provision of an answer to the above necessitates the resolution of a number of subsidiary questions, as follows:

1. What is the most appropriate mechanism for preprocessing and representing large network data so that movement patterns can be extracted?
2. Given a solution to (1) what are the most appropriate mechanisms whereby movement patterns can be mined (learnt/extracted) from network data.
3. Following on from (2), given mechanisms for mining movement patterns from network data how can these mechanisms be scaled up to address movement pattern mining from very large networks.
4. Once we have a collection of movement patterns how can they best be applied to previously unseen network data and how do we know whether the manner in which movement patterns are applied produces the correct results?

The following section provides a description of the broad research methodology adopted to address the above research question and the associated subsidiary questions.

1.4 Research Methodology

The broad adopted research methodology was to focus on a specific primary, real-life, application domain, and then support this with a secondary application domain. The selected primary application domain was the Great Britain (GB) cattle movement database, whilst the selected secondary domain was the Chinese Jiayuan Social Network. The GB cattle movement database is maintained by the Department for Environment, Food and Rural Affairs (DEFRA) and it records all cattle that are moved between pairs of locations in GB. The locations can be viewed as network vertices and the cattle movement in terms of edges connecting vertices. The domain was selected because: (i) it exemplified a large network, (ii) it had not been significantly studied previously and (iii) it had a clear application for the concept of movement patterns in that they could be used to predict how cattle moved round related networks to support (say) bovine disease spread studies. The Jiayuan Social Network was selected, again because it was an exemplar of a large network, but more importantly, because it provided an alternative application of the movement pattern idea, as envisioned in this thesis, allowing more extensive investigation of the movement pattern concept.

To provide an answer to the first subsidiary question, *"What is the most appropriate mechanism for preprocessing and representing large network data?"* it was decided to investigate solutions to the second subsidiary question, *"what are the most appropriate mechanisms whereby movement patterns can be mined?"* and then backtrack to provide an answer to the first question. Given that, as already noted, movement patterns can be viewed as a special form of Frequent Itemset Mining (FIM) as popularised in association rule mining [6], the start point for providing an answer to the second subsubsidiary question was to look at existing FIM algorithms and attempt to adapt these, more specifically to look at the Apriori algorithm [6]. A frequently cited disadvantage of FPM is the significant computation time required to generate large numbers of patterns (many of which may not be relevant). The anticipation was that any adapted FIM algorithm would display the same disadvantage. Whatever the case an initial Apriori-based Movement Pattern Mining (MPM) algorithm would provide a benchmark with which to compare further algorithms. The fundamental idea behind the anticipated further algorithms was to utilise the known "shape" of the desired movement patterns, in that they would have three parts, to influence the MPM.

Once a number of MPM algorithms had been established, the next stage was to consider how these algorithms might be scaled up given a very large network, too large to be effectively processed using a single computer, to provide a solution to the third subsidiary question. Some form of distributed computing was envisioned. To this end a number of approaches were available but three obvious candidates, because of the frequency with which they are cited in the literature, were: (i) Hadoop/MapReduce [91], (ii) Shared Memory Systems and (iii) Distributed Memory Systems. It was thus decided that the operation of the identified algorithms would be considered in terms of these three frameworks (Hadoop/MapReduce, MPI, OpenMP).

Consideration was then given to how movement patterns, once identified could be applied. This was done by considering a number of application scenarios in the context of the identified primary and secondary application domains. The scenarios were as follows:

- Given a collection of holding areas similar to those feature in the cattle movement database what traffic might we expect (predict).
- Given a Social Network Data, such as the Jiayuan, how might movement patterns be used to make recommendations.

The second part of the fourth subsidiary question was *" how do we know whether the manner in which movement patterns are applied produces the correct results?"*. The answer to this question clearly depends on the nature of the application to which movement patterns are applied, in other words, the nature of the above-listed scenarios. Part of the research was therefore directed at devising mechanism to determine whether the right predictions (Scenario 1) or the right recommendations (Scenario 2) were being made. A number of additional criteria were also devised for measuring the overall success of the research work conducted and presented in this thesis, as follows:

- **Generality:** Any proposed MPM should be generic, it should operate with respect to networks of various kinds (large or small) and in the context of a variety of application domains.
- **efficiency:** The manner in which any proposed movement patterns are mined and applied should be as efficient as possible. In other words, the runtime and storage required for mining movement patterns should be minimised, as should the runtime and storage requirements for applying movement patterns. In other words, the memory heap size and runtime in second were considered as an important factor to measure the efficiency of any proposed movement pattern algorithm.
- **effectiveness:** Clearly any proposed MPM or movement pattern application should be effective in that the correct movement patterns should be discovered and they should be applied in the appropriate manner.

1.5 Contributions

The main contributions of the research work presented in this thesis are the movement pattern concept and how such patterns can be extracted and applied in the context of large networks. To the best knowledge of the authors there has been very little (no?) work on movement patterns as conceptualised in this thesis. The more specific contributions can be itemized as follows:

1. The Apriori-based Movement Pattern (AMP) algorithm.
2. The Shape-based Movement Pattern (ShaMP) algorithm.
3. Three big data variations of the ShaMP algorithm based on distribution/shared memory system techniques and Hadoop/MapReduce, with a comparison of their operation.
4. A mechanism for applying MPs to predict traffic in networks.
5. The RecoMP recommender system, which utilises MPs so as to provide recommendations in the context of Social Networks.
6. A mechanism for evaluating the effectiveness of the use of movement patterns in the context of prediction.
7. A mechanism for evaluating the effectiveness of the use of movement patterns in the context of recommendation.

1.6 Organization of the Thesis

The remainder of thesis is organized into three main parts: (i) Previous work (Chapter 2), (ii) Pattern mining methods for large networks (Chapters 3, 4 and 5) and (iii) Movement pattern application mechanisms (Chapter 6).

The connection between each part and the subsidiary research questions identified in Section 1.3 is presented in Table 1.1. The last, Chapter 7, of the thesis provides a summary of the work presented, the main findings in relation to the research question and associated subsidiary questions identified above, and some ideas for future work.

Table 1.1: Thesis structure.

Content	**Chapter**

Part I: Previous Work.

Research background	**Chapter**
Presents a literature review of the related research from different prospective starting for Data Mining in Social Network in a team of Movement Pattern, then Prediction Modelling in Social Network, finally, Processing large Networks by utilizing big data techniques	2

Part II: Movement Pattern Mining.

Research issue to investigate	**Chapter**
What is the most appropriate mechanism for preprocessing and representing large network data so that movement patterns can be extracted?	3
What are the most appropriate mechanisms whereby movement patterns can be mined (learnt/extracted) from network data?	4
Following on from last question, given mechanisms for mining movement patterns from network data, how can these mechanisms be scaled up to address movement pattern mining from very large networks?	5

Continued on next page

Table 1.1 Thesis structure continue

Content	**Chapter**

Part III: Prediction and Recommendation Modelling.

Research issue to investigate	**Chapter**
Once we have a collection of movement patterns how can they best be applied to previously unseen network data and how do we know whether the manner in which movement patterns are applied produces the correct results?	6
In this part, the thesis will summarized and discuss the main funding and the research contributions with a further vision for future work.	7

1.7 Published Work

The main contributions of this thesis have been published in three conference papers. These papers are itemized below.

1. Al-Zeyadi M., Coenen, F. and Lisitsa, A (2017). *"Mining Frequent Movement Patterns in Large Networks: A Parallel Approach Using Shapes"*. In: Bramer M. and Petridis M. (Eds), Research and Development in Intelligent Systems XXXIII, Proc. AI'2017, Springer, pp53-67. Conference paper that presented the Shape-based Movement Pattern (ShaMP) algorithm, an algorithm for extracting Movement Patterns (MPs) from network data that can later be used (say) for prediction purposes. The principal advantage offered

by the ShaMP algorithm is that it lends itself to parallelisation so that very large networks can be processed. The concept of MPs is fully described together with the realisation of the ShaMP algorithm. The algorithm is evaluated by comparing its operation with a benchmark Apriori-based approach, the Apriori-based Movement Pattern (AMP) algorithm, using large social networks generated from the Cattle Tracking Systems (CTS) in operation in Great Britain (GB) and artificial networks.

2. Al-Zeyadi M., Coenen, F. and Lisitsa, A (2017). *"On the mining and usage of Movement Patterns in large traffic networks"*. Proceedings of the IEEE International Conference on Big Data and Smart Computing (BigComp), pp. 135-142. This conference paper investigated the efficiency of the ShaMP algorithm when implemented using Hadoop/MapReduce and Massage Passing Interface (MPI). The paper also presented a prediction mechanism whereby the identified MPs can be used to predict the nature of movement in a previously unseen network.

3. Al-Zeyadi M., Coenen, F. and Lisitsa, A (2017). *"User-to-User Recommendation using The Concept of Movement Patterns: A Study Using a Dating Social Network"*. Proceeding of the international joint conference on knowledge discovery, knowledge engineering and knowledge management (IC3K). This conference paper makes the observation that the MP concept can equally well be applied in the context of recommender systems, more specifically recommender systems embedded into Social Networks (SNs). The idea was to conceive of a SN as a collection of vertices, each representing an individual, the interchange of messages between vertices can then be considered to represent the traffic (edges) between vertices. Frequently occurring MPs can then be extracted and used to generate recommendations (to existing users and new users).

1.8 Summary

In summary, this chapter has provided an overview and background for the research described in the reminder of this thesis, including details concerning the motivation for the work and the research question and subsidiary questions. It has also provided a brief description of the formalism of network and movement patterns; as well as the research evaluation criteria and the contributions of the work. In the following chapter a literature review, intended to provide more detail regarding the background concerning the research described in this thesis, is presented.

2 Literature Review

Real-world networks often involved millions of nodes and connections between them. Given the large amounts of interconnected data available, the challenge is the extraction of useful knowledge from this data in a manner that is both efficient and effective. The work presented in this thesis considers one aspect of this challenge, the extraction of patterns from this data, specifically movement patterns. This chapter provides a review of existing work in this field, and other background material, so as to contextualise the work presented later in this thesis and to provide the reader with necessary background knowledge. The chapter is divided into three main sections. The first section, Section 2.1, discusses social network mining and places the idea of Movement Pattern Mining (MPM) in this context. The main motivation for the concept of movement patterns, as espoused in this thesis, is to predict future movement (traffic) in a current network, or movement in a related network. The second section, Section 2.2, therefore discusses the nature of prediction in networks, including related work on link prediction and prediction in the context of recommender systems. The third section, Section 2.3, reviews the big data and related techniques that were adopted with respect to the work presented later in this thesis, particularly the Hadoop/MapReduce and Massage Passing Interface as used in Distributed Memory Systems so as to process large data collections. The chapter is concluded with a summary in Section 2.5.

2.1 Data Mining in Social Networks

We live in the era of the big data connected the world in which virtual networks are intertwined with our daily lives. The prevalence of such networks, of all kinds, has grown dramatically. The most common examples of virtual networks include: social networks [67], computer networks [20] and Peer-to-Peer Networks [26]. Another example of a virtual network and one used with respect to the evaluation reported on later in this thesis is social networks. However, networks can also be physical, the best examples are transportation networks (road, shipping, air, train) [36]. A specific example of a physical (transportation) network is the cattle movement network also used for evaluation purposes with respect to the work reported on later in this thesis.

Despite variability in semantics, the general theory of networks is the same. They share a common structure whereby entities (vertices or nodes) are linked through a specific relationship (edges or links). The entities can represent individuals (as in the case of social networks), inanimate entities (as in the case of computer networks) or locations (as in the case of distribution and road traffic networks). The edges then represent some relationship between these entities. Thus a network, represented in terms of vertices (nodes) and edges (links), can be considered to be a form of a graph $G(V, E)$ where V is the set of vertices and E is the set of edges. The total number of vertices n in a graph is then $n = |V|$ and the total number of edges is $m = |E|$. We can identify two types of edges: (i) directed and (ii) undirected. Directed edges, as the name suggests, imply a direction with respect to the relationship they represent, in other words, a direction associated with (say) the flow of information or the traffic they represent. A good example is the who-follows-whom relation found in the Twitter social network where the edge direction indicates the follower and the followee. An undirected edge does not have a specific direction associated with the relation represented. In this case, we can think of the flow of information or traffic as being reciprocal. For example, the messages exchanged using the Facebook social media network. Such reciprocal flow can, of course, be represented by two opposing directed edges. In the case of the example networks used for evaluation purposes with respect to the work presented in this thesis the cattle movement network featured directed edges (indicating the sender and receiver), while the social network features undirected or, to be exact, reciprocal edges.

Coinciding with the growth in networks of all kinds, there is a corresponding desire to analyse (mine) such networks and extract valuable, but hidden, knowledge, from them. *Network Mining* (NM) is the process of representing, analysing, and extracting actionable information (such as patterns) from network data where the typically objective is with respect to some social and/or commercial gain. NM has been investigated in the context of various application domains including: (i) how law enforcement and intelligence agencies fight organized crime, such as narcotics and money laundering [95] and terrorism [48], and (ii) in commerce so as to increase sales of products and services by exploiting the relationship between existing and potential customers [45]. Similarly, a variety of tools and methods have been proposed to maintain, manipulate and visualise network data to support NM, and research and business network understanding. Examples of such tools include: Ctoscape [79], Gephi [15], and JUNG [69].

NM is a branch of the more general domain of Data Mining (DM). Data Mining is an important element of the current Big Data trend and encompasses a wide range of techniques for extracting useful knowledge from data, such as trends, patterns and rules [42]. The research described in this thesis is directed at the provision of a particular form of NM, directed at the extraction and usage of Movement Patterns (MPs) from network data. We refer to this as Movement Pattern Mining (MPM). The concept of MPM, as conceived of in this thesis, and to the best knowledge of the author, has not been previously addressed in the literature. However, pattern mining, in general, has been extensively studied within the context of DM. More specifically the MPM concept has parallels with the concept of Frequent Pattern Mining (FPM) found in DM. FPM is therefore discussed in more detail in the following sub-section. The idea of MPM is then discussed further in Sub-section 2.1.2.

2.1.1 Frequent Pattern Mining

Frequent Patterns (FPs) play an important role with respect to many data mining applications [4] with the goal of discovering knowledge in the form of repeated patterns [9]. Many researchers hold the view that frequent pattern identification is a fundamental intermediate step with respect to a variety of problems [3]. FPs are of particular interest with respect to the work presented in this thesis because, as already noted above, the idea of movement patterns shares some similarities with the idea of frequent patterns.

Frequent Pattern Mining (FPM) is typically directed at Boolean (binary or zero-one) valued data. The main objective of frequent pattern mining is to discover sets of attributes that occur together frequently. The first frequent pattern mining algorithm was presented by Agrawal et al. in 1993 [6]; this was the catalyst for a wide range of follow-on research. Indeed it can be argued that the research on movement patterns considered in this thesis can be traced back to this early work on FPM.

FPM entails some frequently quoted disadvantages as follows:

1. The computation time required to generate large numbers of patterns is substantial given a large data set [9].

2. The size of the output (the frequent patterns) may be larger than the size of the input [71, 84].

3. Many of the patterns identified may not be relevant in the sense that they are sub/super sets of other patterns [4].

These are all issues which are equally valid in the context of MPM and which are returned to later in this thesis.

Movement patterns, as envisioned in this thesis, can be regarded as a special form of a frequent pattern in that, in the abstract, they can be thought of as three-part frequent patterns comprised of a “sender”, “movement” and “receiver” part. In other words, the movement patterns considered in this thesis are more prescriptive than standard FPs.

Given that the MPM concept as presented in this thesis shares similarities with the concept of FPM it was decided, within the context of the proposed research methodology, that a good benchmark MPM algorithm would be one founded on the Apriori algorithm cited above, namely the AMP algorithm presented later in this thesis in Chapter 4.

2.1.2 Movement Pattern Mining

The term movement pattern is encountered in various contexts in the literature, with respect to a number of different application domains, and with differing definitions of what a movement pattern is. Of note is the usage of the term with respect to various forms of video analysis. One example is in the context of surveillance and security where the “pattern of movement” of video objects is tracked across video data; for example, with respect to people, animals, and traffic [16, 28, 34, 93]. Another application domain where the term movement pattern is used is in the context of video event recognition [30]. A further example is with respect to simulations of animal behavior. For example in [87] what are referred to as "movement patterns" are extracted from video data and used to drive a multi-agent based simulation.

The movement patterns, as conceived of in this thesis, and as will become clearer later in the thesis, are different from the movement patterns found in video data analysis as described above. The movement patterns central to the work presented in this thesis can be thought of as network movement patterns as opposed to video movement patterns. In other words, patterns that describe how traffic or information moves (flows) round a network. To the best knowledge of the authors, there has been no previous work on movement patterns as conceived in this thesis.

2.2 Prediction Modeling in Networks

As noted earlier, the motivation for the MPM idea is to use such patterns to predict movements in future renditions of the same network to that from which the patterns were mined, or on closely related networks. There is existing reported work on prediction in the context networks. Popular examples include: (i) the prediction of disease spread in a human contact networks using metrics such as the degree of distribution, clustering and path length [8], (ii) prediction in the context of best viral marketing strategies by identifying "networks of influencers" [77], (iii) link prediction of various kinds [63] and (iv) the building of effective recommendation systems based on vector and edge weightings in dating networks [104]. All four examples, in one form or another, are potential applications for the work on MPs presented in this thesis. The last two are considered as exemplar applications with respect to the work presented in this thesis and thus are considered in further detail below (Sub-sections 2.2.1 and 2.2.2).

2.2.1 Link Prediction in Networks

Link Prediction (LP) can be define as the process of predicting links between pairs of vertices in a graph [17]. LP has recently received increased attention in both the physical and computer science communities. The idea is, given a new vertex x, to predict the existing vertices to which a new vertex will be linked. The idea of LP has some overlap with the problem of inferring missing links in incomplete graphs [46]. The goal of LP can be related to one of the applications which motivated the work on MPM and at which the MPM concept is directed; although the MPM concept is not just directed at whether a link exists, but also at the nature of the link. A further distinction between the work on LP and missing link resolution, and the work presented in this thesis, is that the first two are typically conducted according to graph structure, dynamic in the case of link prediction [17] and static in the case of missing link resolution [46], rather than using the concept of MPs. Whatever the case, because of the similarity with respect to the desired end goal of the proposed MPM, LP is discussed in some further detail in this subsection.

According to the survey conducted by the athores in [63] LP algorithms can be categorised as follows:

- **Similarity Based Algorithms**: LP algorithms in this category are the simplest and founded on the idea of a similarity score, S_{xy}, where x is a new vertex and y is an existing vertex, according to the linked node pairings that already exist in a given network [56, 63]. If the similarity score is high then a link is suggested. Many algorithm have been proposed that adopt this approach. There are many variations of the basic similarity based link prediction algorithm which can be classified into three groups according to the nature of the similarity indices: (i) Local Similarity Indices, (ii) Global Similarity Indices and (iii) Quasi-Local Indices. Examples of the first include: (i) Common Neighbours (CN) [25], (ii) Salton Index [24], Jaccard Index [64], (iii) *Srensen* Index [59], (iv) Hub Promoted Index [17], (v) Hub Depressed Index [63], (vi) Leicht Holme Newman Index [53], (vii) Preferential Attachment Index [66], (viii) Adamic Adar Index [1],(ix)Leicht-Holme-Newman Index [53] and (x) Resource Allocation Index [62]. Examples of the second include: (i) Katz Index [44], (ii) Leicht Holme Newman Index [53], (iii) Average Commute Time [31], (iv) Cosine based on L+ [31], (v)Random Walk with Restart [85], (vi) SimRank [41] and (vii) Matrix Forest Index [21]; while examples of the third include: (i) Local Path Index [61, 105], (ii) Local Random Walk [61] and (iii) Superposed Random Walk [60].

- **Maximum Likelihood Methods**: The algorithms in this category, instead of using some form of indexing, consider the organisation and structure of a given network expressed in terms of rules and parameters, which are then used to indicate the likelihood of a link with respect to a new node [78]. A disadvantage of the Maximum Likelihood approach is that it tends to be very time consuming and thus not applicable to Big Networks [63].

- **Probabilistic Models**: In this last category, the prediction of missing links is conducted by abstracting the underling structure from a given network so that a model is learned. A variety of models have been proposed, but from the literature the following three main models can be identified [63]: (i) the Probabilistic Relational Model [32], (ii) the Probabilistic Entity Relationship Model [40] and (iii) the Stochastic Relational Model [99].

The proposed MP approach, bearing in mind that we are not only interested simply in link prediction but also in the nature of the predicted links, does not fit well into any of these categories. The last category, the Probabilistic Model category, is arguably the most suited. The idea is to use identified MPs to predict the traffic (movement) in future incarnations of a given network, or closely related networks. More specifically, in the context of the cattle movement prediction scenario considered as an exemplar application in this thesis.

2.2.2 Recommendation in Network

Recommender systems are used extensively in the retail sector to recommend potential purchases (products and services) to customers. They operate by processing previous customer purchase data to make recommendations. Recommender systems have been found to provide a significant impact with respect to improving user satisfaction in online retail settings [82, 90]. Broadly, recommendation systems can be categorised as being either: (i) *Item-to-Item* or (ii) *User-to-User*. The main difference is that User-to-User recommendation systems need to make reciprocal recommendations while Item-to-Item systems do not; User-to-User recommendation system needs to satisfy both parties [72]. Well known examples of Item-to-Item recommendation systems are those embedded in Amazon, Netflix and Spotify [7, 58]; we are all familiar with the "users who bought X also bought Y" mantra. Well known examples of User-to-User recommendation systems are those embedded in Facebook and Linkedin; the "people you might know" mantra! Another example application domain and that of interest with respect to this thesis, where User-to-User recommendation features, is Dating Social Networks (DSNs).

DSNs have become an important platform for people looking for potential partners online. In a large dating network, finding potential partners is time consuming, therefore many DSNs give compatible partner suggestions; in the same manner as more general recommender systems, see for example [76]. Developing a recommender system for a DSN is more challenging because the recommender system must satisfy the preferences of pairs of users [73] as opposed to single users; DSN recommender systems thus fall into the User-to-User categorisation of recommender systems. Later in this thesis, to illustrate the utility of the MPM concept, a dating recommendation system is proposed that can be embedded into DSNs. The idea is to conceive of a DSN as a collection of vertices, each representing an individual, and the interchange of messages between vertices as representing the traffic (edges) between vertices. Frequently occurring MPs can then be extracted and used to generate recommendations (to existing users and new users).

There has been much work directed at DSN recommendation. Of key concern is the quality of the recommended matches; poor quality matching will result in people looking elsewhere. The context of matching is typically done using either: (i) user profiles, (ii) expressed preferences or (iii) user behaviour. For example, in [50] the authors propose a way of modelling both the duality of user similarity to each other and preferences towards other users, by using split-complex numbers. The authors demonstrated firstly that their unified representation was capable of modelling both notions of relations between users in a joint expression and secondly that their system could be applied in the context of recommending potential partners. In [94] the authors introduced a recommendation system that made use of profiles and references and provided a list of recommendations that a user might be compatible with by computing a reciprocal score that measured the compatibility between a user and each potential dating candidate. In [86], the authors proposed a DSN recommendation framework founded on a Latent Dirichlet Allocation (LDA) model that learns user preferences from observed user messaging behaviour and user profile features. Whatever the case, the majority of User-to-User DSN recommendation systems are founded on (graph based) Collaborative Filtering (CF) algorithms [49, 86] that focuses on user behaviour. The intuition is that user behaviour is a much better indicator for recommendations than user profiles or expressed preferences [49]. Examples, where CF filtering has been used in the context of DSNs, can be found in [19, 51]. Given the popularity, and claimed benefits, of the CF approach this is the approach with which the proposed MP based RecoMP algorithm is compared later in this thesis. For the purpose of the evaluation the authors developed a bespoke CF based DSN recommendation algorithm called RecoCF, this is described in further detail in Chapter 6. The distinguishing feature between the above DSN recommender systems and the RecoMP based system proposed in this thesis is the MP concept. To the best of the author's knowledge, there has been no work directed at user-to-user recommendation using MPs as presented in this thesis.

2.3 Processing Large Networks

As mentioned earlier, one of the research issues considered in this thesis is how to process large networks (in terms of both vertices and edges). In general, the challenges of processing large networks is two-fold. Firstly, network sizes have increased exponentially comparing to the available memory of a single computer. Secondly, the computation time required to process a given large network has also dramatically increased. The essential key tool for addressing these two challenges is "computational parallelism", where multiple computer resources are used simultaneously to solve a specific computational problem [14]. Processing large networks by applying computational parallelism tools offer two advantages. First, operations on a given large network become computationally tractable whereas it would not be possible to apply the same operations on a single machine. Second, computational parallelism gives an enormous speed up opportunities. Therefore, computational parallelism is seen as a key technology for addressing the two challenges identified above. The usage of such tools in the context of MPM, is reported on later in this thesis. This section, therefore, presents two mechanisms whereby computational parallelism can be achieved. Namely, the Shared Memory mechanism as used with respect to OpenMP and the Distributed Memory mechanism as used with respect to (i) Massage Pass Interface (MPI) and (ii) Hadoop/MapReduce. The objective of this section is thus to provide the reader with some necessary background concerning these approaches so that material presented later in the thesis can be readily understood.

The remainder of this section is organised as follows. The generic challenges of parallel algorithm design are considered in Sub-section 2.3.1, the principle of shared memory systems is discussed in Sub-section 2.3.2 with a focus on OpenMP as an example of a particular API to support the idea of shared memory system. The principle of distributed memory systems is then discussed in Sub-section 2.3.3 with a focus on two particular APIs to support the idea of Distributed Memory Systems, namely: the Message Passing Interface (MPI) API (Sub-section 2.3.4) and MapReduce coupled with Haddoop (Sub-section 2.3.5). The relevance of these APIs, is that these were adopted with respect to work presented later in the thesis.

2.3.1 Challenges of Parallel Algorithm Design

The design of a parallel algorithm involves a number of challenges, in addition to those associated with the design of more standard serial algorithms, namely: memory scalability, work partitioning, and load balancing. Memory scalability is fundamental when working with Big Data. However, as the number of processes increases, the memory required by each process decreases as indicated by the following Scalability Equation [9]:

$$\Theta\left(\frac{n}{p}\right) + O\left(p\right) \tag{2.1}$$

where: (i) n is the size of input dataset, (ii) p is the number of processes executed in parallel and (iii) Big O and Θ are symbols used in complexity theory, computer science and mathematics, to describe the asymptotic behaviour of functions (they indicate how fast a function grows or declines). Also by defination, any fucation $f(x)$ is $O(g)x))$ if some constant $c \times g(x)$ is greater than or equal to $f(x)$; $|f(x)| \leq c\,|g(x)|$ while $x > k$. Whereas a fucation $f(x)$ is $\Theta(g(x))$ if some constant $c_1 \times g(x)$ is less than or equal to $f(x)$ and some constant $c_2 \times g(x)$ is greater than or equal $f(x)$; $c_1.\,|g(x)| \leqslant |f(x)| \leqslant c_2.\,|g(x)|$ while $x > k$.

In the case of the movement patterns of interest with respect to this thesis, the real challenge is the distribution of the data and the data mining task across the available processes. Data partitioning has to be undertaken in such way that the data is distributed equally in manageable chunks. Work partitioning is concerned with dividing a given problem (the data mining task) into a set of independent tasks, where each task addresses part of the problem. Ideally, all tasks should be executed concurrently and thus they need to be independent. There are two fundamental data partitioning models that can be adapted: the shared memory model and the distributed memory model. Each is discussed in further detail in the following two sub-sections.

2.3.2 Principles of Share Memory Systems

A Shared Memory System is a Memory Model in which a single memory space is shared by all processes. The shared memory model is fast becoming a key model for designing parallel algorithms due to the increase in multi-core workstations. An issue with shared memory systems is when two processes attempted to write to the same memory location at the same time thus resulting in what is called a *race condition*. Also, there can be delay when a process running on one physical machine asks for a particular memory address on another physical machine. In the context of this thesis the shared memory concept has been implemented using Omp4j[1]; which is an open-source OpenMP preprocessor for Java that provides high scalability.

2.3.3 Principles of Distributed Memory Systems

A Distributed Memory System (DMS) consists of multiple independent processing units where each process has an associated local memory space and all processes are connected by a general interconnected network. The nature of DMS makes large computing processing possible. However, communication between processing nodes requires a message-passing model that entails explicit use of send/receive primitives [74]. There are several APIs that may be used in the context of DMS. However, in the context of this thesis, two widely used DMS have been used *Message Passing* and *Hadoop/MapReduce*. These are therefore further considered in the following two sub-sections.

2.3.4 Message Passing

Message Passing is a distribute programming API that provides (at a minimum) a send and a receive functions. Processes typically identify each other by “ranks” in the range $0, 1, \ldots, p-1$, where p is the number of processes [18]. Massage Passing is a very powerful and versatile form of API for developing parallel programs. there are a number of Message Passing implementations, but typical such APIs also provide a wide variety of additional functions. For example, there may be functions for various “collective” communications, such as “broadcast” or “reduction” [18]. The most widely used Message Passing API is the Message Passing Interface (MPI) which defines a library of functions that can be called from C, C++, Fortran and

[1]http://www.omp4j.org/home

Java programs to create parallel programs. There are several well-tested and efficient implementations of MPI, including some that are free or in the public domain; examples include: MPICH [38], MPICH-G2cite [43] and G-JavaMPI[22]. However, in the context of the research presented in this thesis MPJ Express has been used; this is an implementation of MPI in the Java programming language [12]. MPJ has two ways of configuring the same code; Multicore configuration and Cluster configuration. In the context of the work presented later in the thesis both the Multicore configuration (using a single multicore machine) and Cluster configuration (using a Linux cluster) are considered.

2.3.5 Hadoop/MapReduce

MapReduce is a programming model for processing large datasets, introduced by Google, to support distributed computing [27]. Hadoop [91] is an open source implementation of the MapReduce architecture that allows for the distributed processing of large data sets across a cluster of computers. At a high level, the MapReduce architecture comprises a master node and a number of worker nodes as shown in Figure 2.1. The master node is responsible for assigning scheduled tasks for execution by the worker nodes. To design an algorithm using MapReduce the user needs to create both a Map and a Reduce functions; the main computation functions in a MapReduce framework. When a MapReduce "job" is initialized, the master node first calls a "map" function; a function to filter, sort and produce a set of intermediate key/value pairs (such as sorting students by first name into queues, one queue for each name). And then schedules a set of "reduce" tasks that perform a summary operation by aggregate the values associated with keys (such as counting the number of students in each queue, yielding name frequencies). The significant benefit of MapReduce is to allow the processing of large datasets that can not fit into the memory associated with a single process. This is done by loading the data file to be processed into a distributed file system whereby the data is divided into small chunks or blocks. Figure 2.1 shows the flow of job executing using Hadoop/MapReduce, once the data has been uploaded into the Hadoop Distributed File System (HDFS) [29], the map function reads, in parallel, all the blocks in the file system and produces a set of intermediate key/value pairs. To this end, the MapReduce library collects all intermediate values related to the same intermediate key produced by the Map function and sends them to the reduce function for further processing.

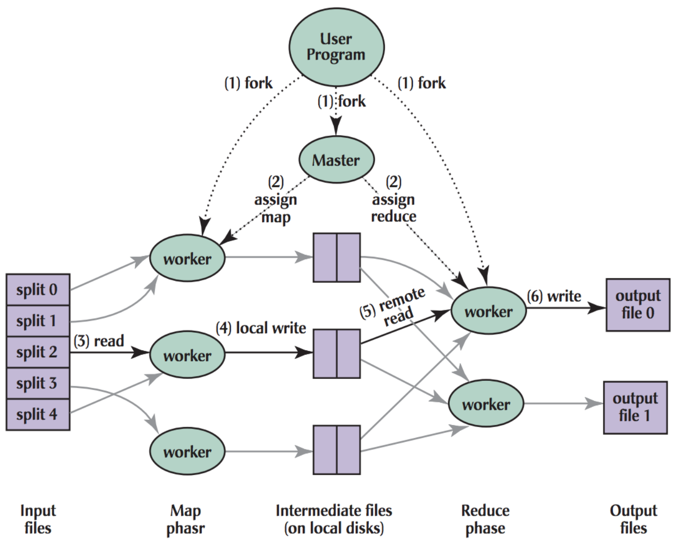


Figure 2.1: The Hadoop/MapReduce job execution flow.[27]

Given the above, MapReduce offers a number advantages:

1. Users are free from manually distributing the data across cluster machines because MapReduce does this automatically.
2. Since the MapReduce library is designed to help process very large amounts of data using hundreds or thousands of machines, the library tolerates machine failures "gracefully".
3. Total "throughput" is enhanced by re-assigning unfinished tasks of slower nodes to idle nodes (assuming a heterogeneous cluster).

The development of Hadoop/MapReduce have motivated many data mining researchers to improve the efficiency of existing data mining algorithms. For example in the case of Frequent Pattern Mining (see following section), which has similarities with the MPM proposed in this thesis, examples can be found in [29, 57, 68, 81, 97, 98].

2.4 Frequent Pattern Mining using Parallel and Distributed Algorithms

This section discusses recent research on Frequent Pattern Mining (FPM) based on the parallel computing concept. In [13, 89] the efficiency of FPM was improved by utilizing multi-core CPUs. In [98] the authors proposed an improvement on the Apriori algorithm based on Hadoop/MapReduce. Even though the proposed method used state-of-the-art parallel computing, like any other apriori-based algorithms, it suffered from a large number of candidates being produced which led to an increase in the number of times that the database of interest needed to be scanned. The well known FP-growth FPM algorithm was parallaized using the MapReduce architecture in [54], the proposed Parallel FP-growth algorithm (the PFP algorithm) was founded on the idea of dividing the dataset into subsets so that the tasks can be distributed between computing nodes independently. However, the PFP algorithm was unable to cope with big data because of the large number of data transmissions between computing nodes. The authors in [92] proposed an Improved Parallel FP-growth (IPFP) based on Hadoop/MapReduce. IPFP demonstrated a better efficacy compared to PFP, but IPFP also suffered from network transmission problems with big data.

The authors in [68] introduced two algorithms: Dist-Eclat based on the Eclat algorithm [101] and BigFIM based on Apriori [6]. The proposed algorithms both used Hadoop/Mapreduce technology to implement FPM over a large dataset. The Dist-Eclat algorithm focused on performance by using a simple load balancing scheme, while the BigFIM algorithm focused on running large datasets. During the process, DisEclat generated Tid-Lists; however, when the dataset to be processed is too large the generated Tid-Lists would override the memory size. Therefore, BigFIM, based on a hybrid method of Apriori and Eclat, was proposed to solve the issue of memory overriding. BigFIM operated using a three step process. During the first step, BigFIM used an Apriori-based method to extract frequent itemsets of length k. After computing the $k - prefixes$, the second step was to compute the possible extensions. During the last step, BigFIM switched to Eclat, and then utilizes "diff set" operations to mine the subtrees from which frequent itemsets are generated in a depth-first manner. With respect to the performance, and compared to [54], the proposed two algorithms were faster. However, BigFIM still had a memory shortage problem and high communication costs with poor computing performance.

The authors in [55] proposed Sequence-Growth, a distributed FIM algorithm, based on Hadoop/MapReduce technology. The algorithm constructs a tree, using lexicographical ordering, to implement FPM without requiring an exhaustive search of the databases. The algorithm is able to process up to 20 Million transactions within 24000 seconds of runtime.

In [96] the authors proposed a new approach (FiDoop-DP) for mining frequent patterns in large transaction datasets based on the Parallel FP-Growth Algorithm discussed in [54] and a novel data partitioning scheme. The proposed approach showed significant performance improvement in the runtime over the existing Parallel FP-Growth Algorithm by up to 31% with an average of 18%. The FiDoop-DP approach, focussed on distributing a large dataset across data nodes in a Hadoop cluster by partition transactions with high similarity together, and grouping highly correlated frequent items into a list.

The MPM parallel computation algorithms presented in this thesis are, in part, influenced by the previous work, described above, on parallel computation in the context of FPM.

2.5 Summary and Conclusions

This chapter has provided a review of the existing work relevant to this thesis and has provided details concerning the distributed systems technology referred to later in the work. More specifically this chapter has reviewed some previous work on network mining and data mining. In the context of the later, work on frequent pattern mining was considered in some detail because of its similarity with the proposed MPM. Previous work on link prediction and recommender systems was also considered because these are applications similar to those addressed later in this thesis. To provide a fuller understanding of the MPM approaches using distributed memory systems, also presented later in the thesis background information concerning well known parallel approaches was also presented, namely Shared/Distributed Memory Systems and Hadopp/MapReduce. In the next chapter, Chapter 3, the evaluation data sets referenced later in the thesis are considered in detail.

3 Evaluation Data

The idea of networks have become widespread since the rapid development of Web 2.0 [10] which has facilitated, amongst other things, the concept of social networks. Well, known examples include Facebook, Twiter, and Linkedln, where users are encouraged to link to friends, followers, and contacts to form a "social network" through direct and indirect connections to others. From a sociological perspective, the term social network has a much broader meaning beyond digital and online networks to include physical social networks representing face-to-face relationships, political associations and connections, and even co-authoring networks across academic communities. However, networks do not necessarily have to be social networks, the concept can be extended further to: computer networks; distribution networks; road, rail, and air travel networks; and so on. The work presented in the thesis considers networks in the terms of this widest possible context.

This chapter describes the particular network datasets used to drive the work presented in the thesis and consequently used for evaluating the Movement Pattern Mining (MPM) and application algorithms developed. A formalism defining the networks considered in this thesis is presented in Section 3.1. The datasets used with respect to the work presented in the thesis were drawn from three sources: (i) the GB Cattle Tracking System (CTS) operated by the UK Department for

Environment, Food and Rural Affairs (DEFRA), (ii) the Chinese Jiayuan Social Network and (iii) a synthetic data generator. The first was selected because of its size and because it represented an interesting application domain entailing networks different to the social network studies that featured in much previous work on network analysis. The second was selected because it represented a more "traditional" kind of social network. Both data sets represented real examples of networks where the movement pattern concept could offer genuine benefits. The third source was used to generate artificial network data sets where the parameters could be controlled and hence the effect of changes to these parameters, with respect to the proposed algorithms, could be analysed (not possible with the real data sets). These three data sources are considered in further detail in Sections 3.2, 3.3 and 3.4 below. The chapter is concluded with a summary in Section 3.5.

3.1 Network Formalism

Regardless of the precise nature of a network, the networks considered in this thesis adhere to the formalism presented in this section. Any given network G is defined in terms of a tuple of the form $\langle V, E \rangle$, where V is a set of vertices and E is a set of edges [33]. The vertices can represent individuals (as in the case of the Jiayuan Social Network), inanimate entities (as in the case of, say, computer networks) or locations (as in the case of CTS networks). The edges then indicate connections between vertices (virtual or actual). These edges might be indicative of some relationship, such as a friend relationship, as in the case of social networks; or a "hard" connection as in the case of a wired computer network or a road traffic network. In this thesis the way that edges are conceived includes the potential for them to have attributes of some kind; for example: (i) the number of messages sent from one individual to another in a social network, (ii) the volume of data exchanges between two computers in a computer network,

(iii) the quantity of goods sent in a distribution network or (iv) the amount of traffic flow from one location to another in a road traffic network. As such edges are directed (not necessarily the case in other forms of network), consequently vertices can be "sender" (from) vertices or "receiver" (to) vertices. In many cases, vertices will be both senders and receivers. Thus the set of (from) vertices and the set of (to) vertices are not necessarily disjoint; vertices can represent senders and receivers simultaneously.

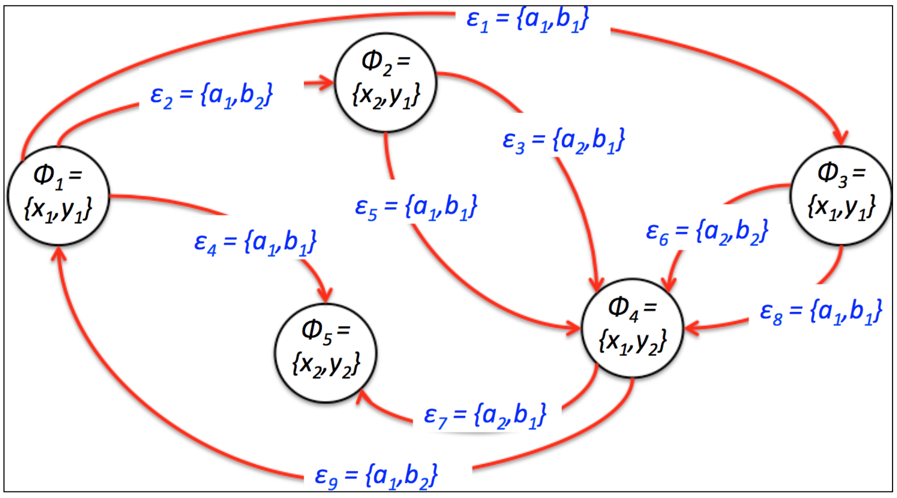


FIGURE 3.1: Example network ($V = \{\phi_1, \phi_2, \ldots, \phi_5\}$ and $E = \{\epsilon_1, \epsilon_2, \ldots, \epsilon_9\}$).

An example network is given in Figure 3.1 where $V = \{\phi_1, \phi_2, \ldots, \phi_5\}$ and $E = \{\epsilon_1, \epsilon_2, \ldots, \epsilon_9\}$. Note that some of the vertices featured are both sender (from) and receiver (to) vertices. Note also that some vertices in the figure are connected by more than one edge. This may simply indicate traffic flow in both directions as in the case the Jiayuan Social Network or it may indicate different types of traffic flow. The work presented in this thesis is not just concerned with traffic flow in a binary context (whether it exists or does not exist), but also in terms of the nature of the traffic flow. Thus where vertices in the figure are connected by more than one edge this might indicate different kinds of traffic flow. For example, in a distribution network, two edges connecting one vertex to another might indicate the dispatch of two different commodities.

Thus, from the foregoing, edges and vertices have sets of attributes associated with them, A_E and A_V respectively. The nature of these attribute sets will depend on the application domain, however, each attribute (where it exists) will have two or more values associated with it; if an attribute has only one value it will be a constant and thus not play any part in any form of MPM. Where necessary, in the remainder of this thesis, we will indicate a particular value j belonging to attribute i using the notation v_{i_j}.

Given a network dataset of some form, the idea was to represent this in a canonical manner. The selected formalism was in terms of "From-Edge-To" tuples, $\langle F, E, T\rangle$, where F and T comprise attribute values associated with the set A_V, and E comprises attribute values associated with the set A_E. We refer to such tuples as "FETs", from the acronym FET (From-Edge-To). A network D thus comprises a set of fets $\{F_1, F_2, \ldots\}$. We refer to such a database as a "FET database".

3.2 GB Cattle Tracking System (CTS) Database

The GB Cattle Tracking System (CTS) database records all the movements of cattle registered within or imported into Great Britain (GB). The database is administered by the Department for Environment, Food, Rural Affairs (DEFRA). The CTS database comprises a number of tables, the most significant of which are the animal, location and movement tables.

Each record in the CTS database describes the transportation of a single animal in terms of: (i) the nature of the animal being transported (bread, gender, dairy or beef, and so on), (ii) the sender location and (iii) the receiver location. The locations represent "holding areas" such as farms, markets, and slaughterhouses. Holding areas are identified by an ID number and geographic Cartesian coordinates

Table 3.1: CTS Database Location Table Attributes and Value Types

Attribute Name	**Value Type**
Sender ID	Number
Sender easting	Number
Sender northing	Number
Sender location type	Nominal
Sender county	Nominal
Sender country	Nominal
Reciver ID	Number
Reciver easting	Number
Reciver northing	Number
Receiver location type	Nominal
Reciver county	Nominal
Reciver country	Nominal

Table 3.2: CTS Database Animal Table Attributes and value types

Attribute Name	**Value Type**
Animal id	Number
Gender	Nominal
Birth Date	Date
Breed	Nominal
Beef	Boolean
Dairy	Boolean
Transmission date	Date

(eastings and northings). Figure 3.2 shows the distribution of different types of holding area across GB. Tables 3.1 and 3.2 list the attribute names and value types for the Location and Animal tables featured in the CTS database.

Data from 2003 to 2006 were extracted from the CTS database and used as the source for the CTS network datasets used for evaluation purposes (as reported later in this thesis). The data was stored in a single data warehouse where each record represented an instance of a single cattle movement. The records were grouped according to their year (episode) and month time stamp; thus $4 \times 12 = 48$ groups, a network dataset was generated for each which, for some experiments, were combined. It should be noted that the maximum number of cattle moved between any pair of locations at the same time was approximately 40 animals.

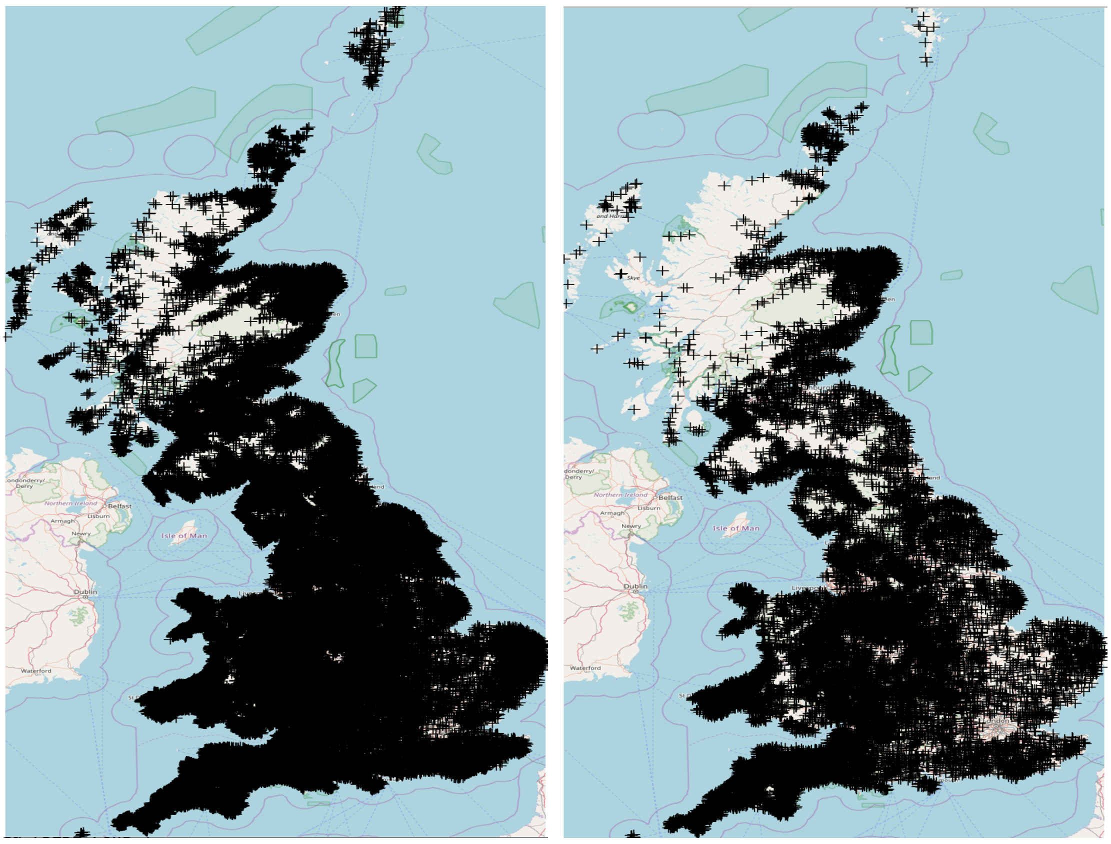
From the foregoing, and as indicated by the maps shown in Figure 3.2, the movement of cattle represents a “movement” network. Before network data could be extracted from this data store (warehouse) the data had to be pre-processed; the CTS database included many missing values. Sub-section 3.2.1 below considers this pre-processing in further detail. Once the data had been cleaned appropriate network data sets could be extracted. This is discussed in further detail in Sub-section 3.2.2 below.

3.2.1 Data Preprocessing

The CTS database featured many missing values indicated by the presence of the literal “NULL”. From a generic perspective there were two potential ways whereby this issue could be addressed. The first was to simply remove, and therefore ignore, all records that had missing values. The second was to infer missing values from the values held in the most similar record that did not feature missing values. The disadvantage of the first was that data would be thrown away. The disadvantage of the second was that there was no guarantee that the substitutions were correct which might in turn mean that any future analysis would be flawed.

In the context of the CTS data for 2003 to 2006, a particular issue was the county attribute associated with sender and receiver locations. In approximately 50% of cases this was set to the value NULL. The evidence for this is presented in Figure 3.3 which shows the distribution values for the county attribute. From the Figure 3.3, it can be seen that the value “NULL” is the dominant value. Therefore, discarding records that have a “NULL” value will results in half of the records being to removed.

It was thus decided to adopt the second option and infer the missing values for the county attribute. Further inspection of the CTS database indicated that


(a) Agricultural Holding Area Locations (b) Slaughterhouse Red Meat Area Locations

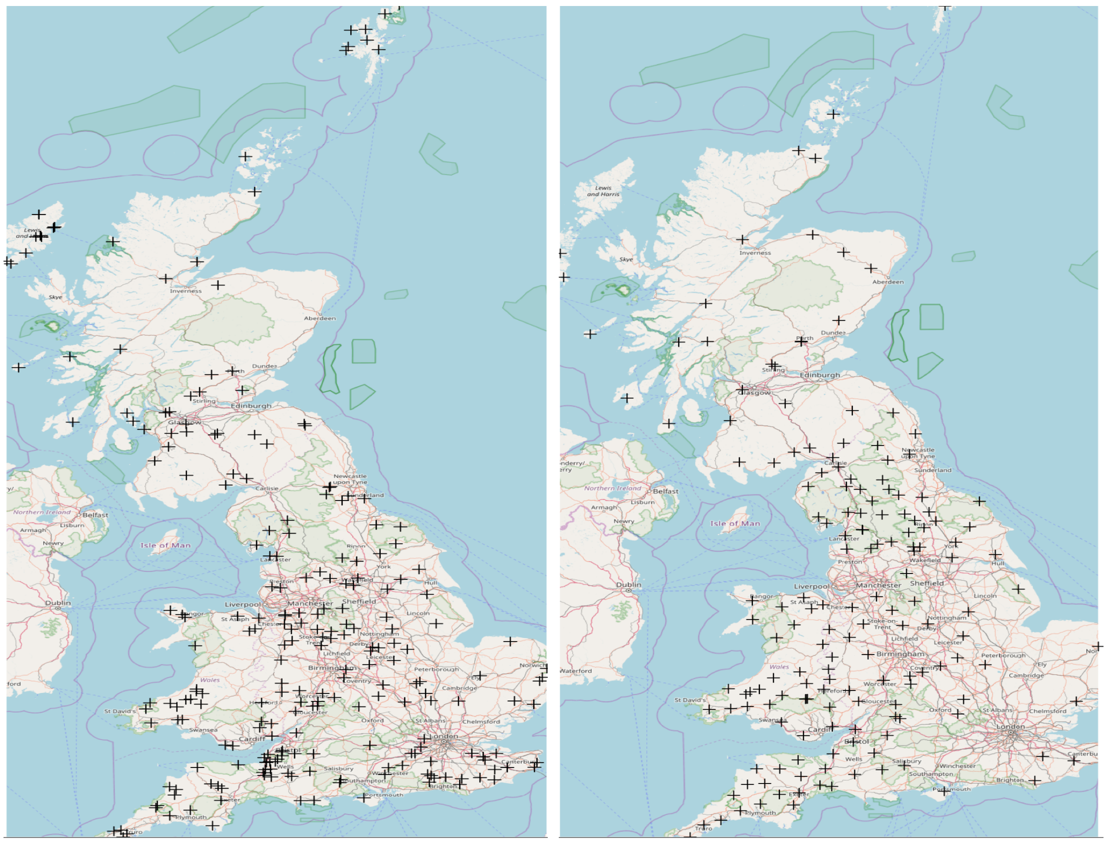
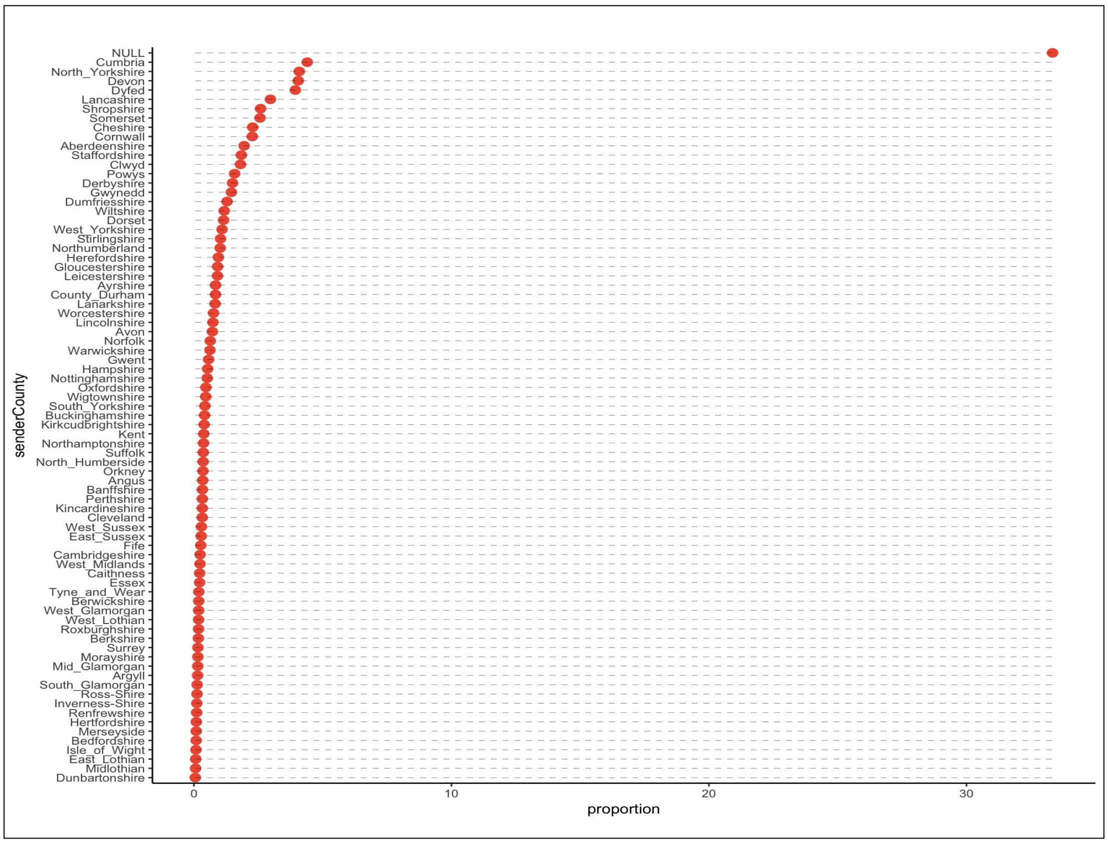
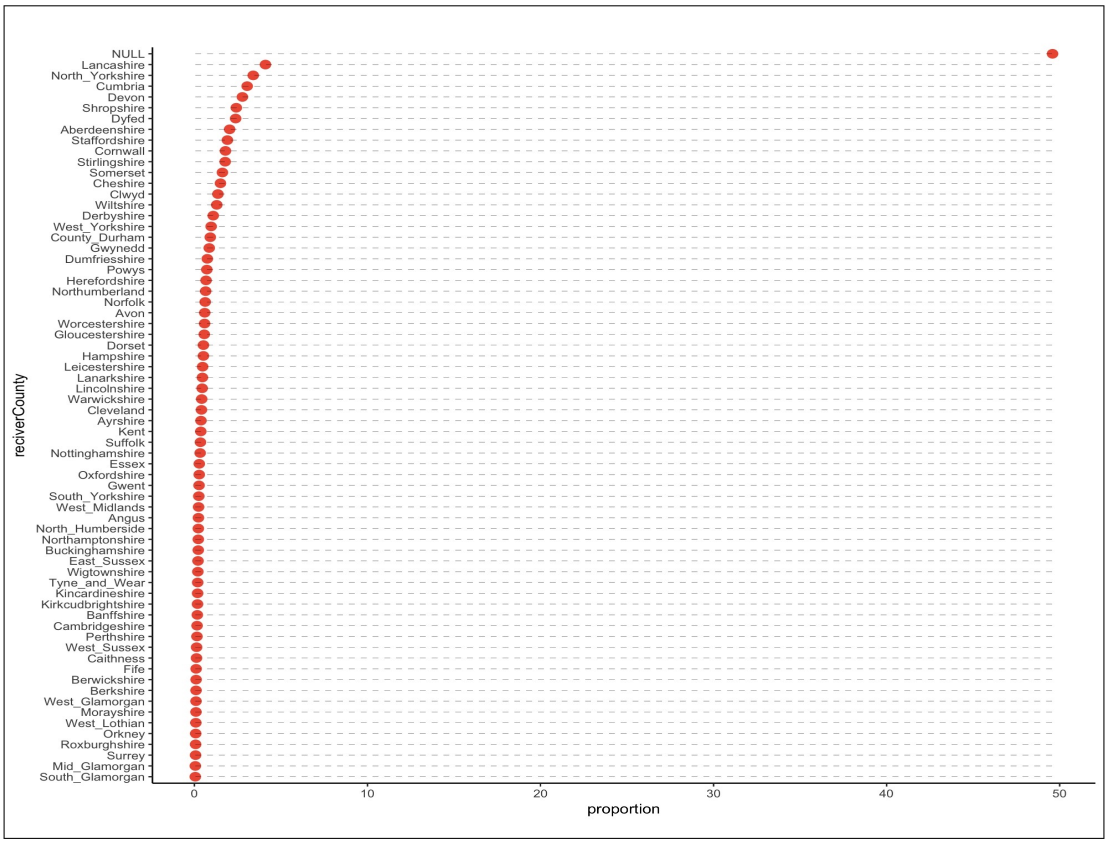

(c) Show Ground Area Locations (d) Market Area Locations

FIGURE 3.2: The geographic distribution of holding areas of different type for the CTS database in 2006

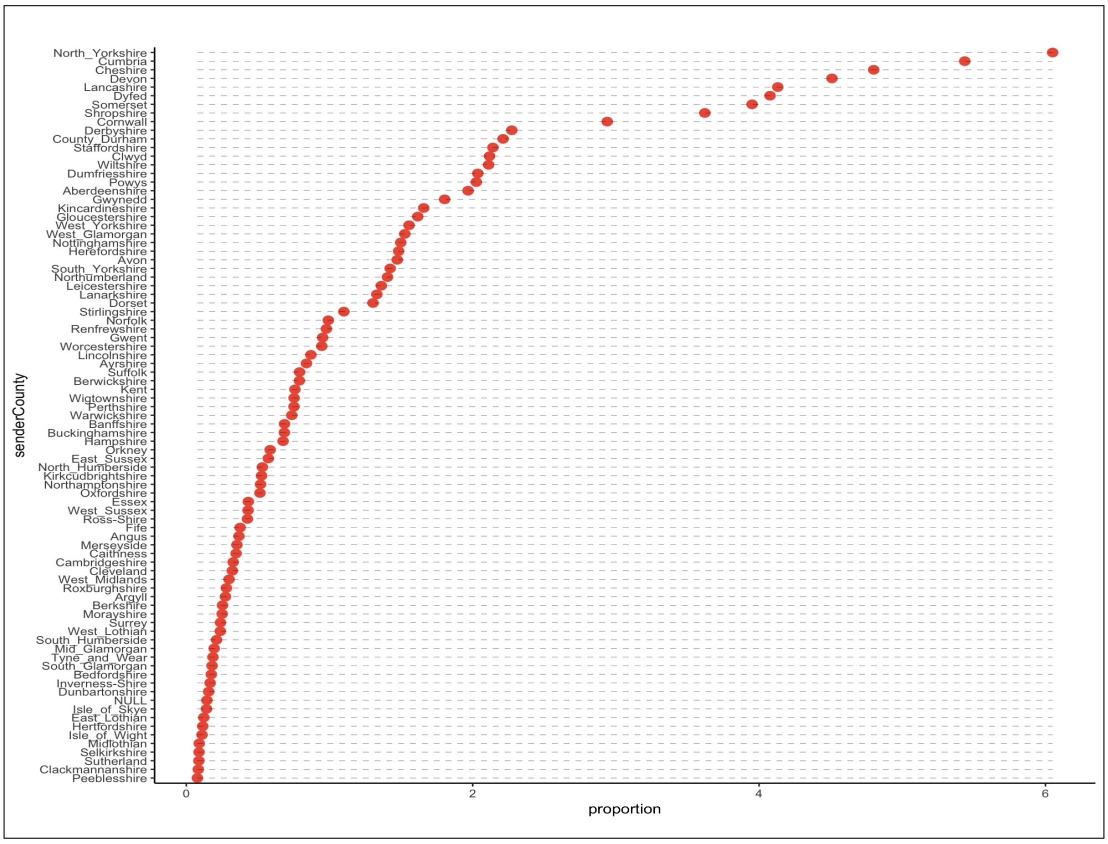


(a) Sender counties

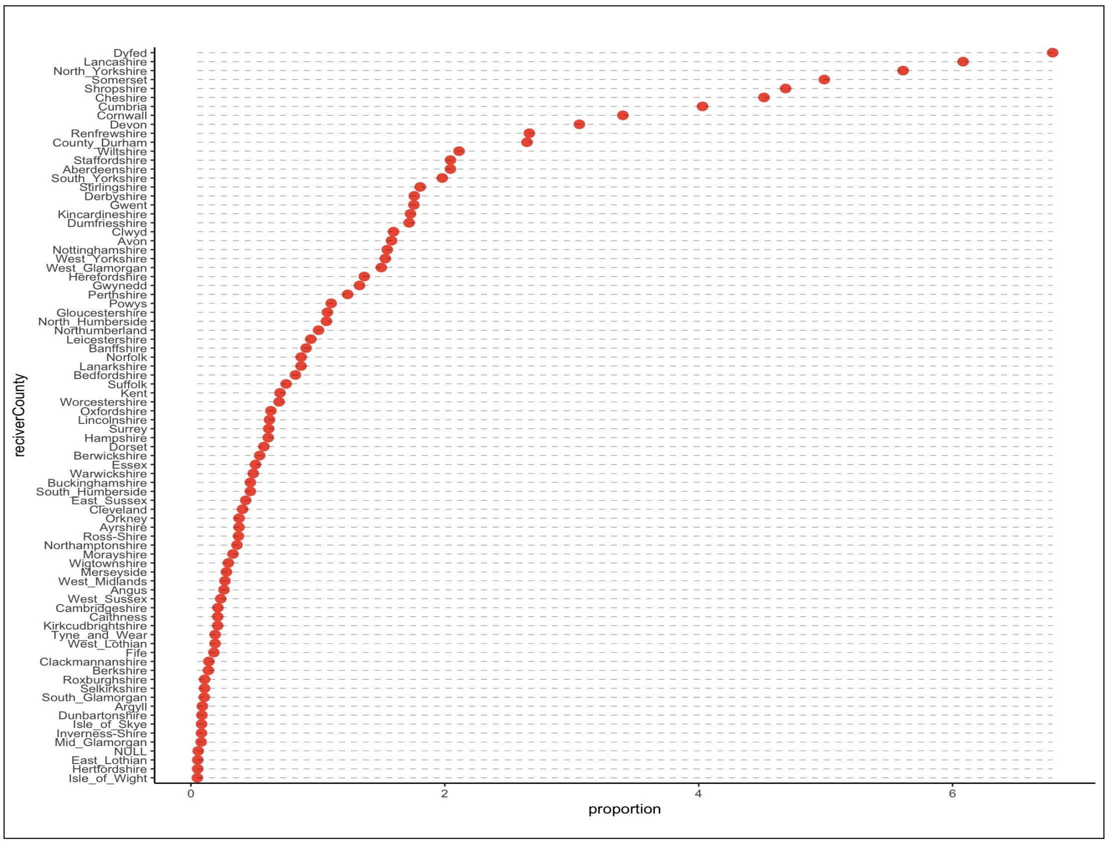


(b) Reciver counties

FIGURE 3.3: The distribution values for the county attribute before preprocessing (sender and receiver counties, 2006 data).



(a) Sender counties



(b) Reciver counties

FIGURE 3.4: The distribution values for the county attribute after preprocessing (sender and receiver counties, 2006 data).

values for the Easting and Northing attributes were always present. These values were OSGB (Ordinance Survey of GB) grid coordinates for the location. The adopted mechanism to identify missing values for the county attribute was thus to find the nearest holding area, using a simple Euclidean distance calculation, with a known county name and use this value. Euclidean distance calculation was selected because the numeric values available for the of Easting and Northing attributes made this an easy calculation to make. The result is a shown in Figure 3.4.

3.2.2 Network Data Generation

Once pre-processing, as described above, was complete, network data generation (extraction) could commence. The objective was to produce FET datasets of the form described in Section 3.1. An issue here was that the CTS database comprised one record per animal. As already noted, up to 40 animals were sometimes moved at the same time. Because the work described in the thesis was directed at patterns describing movement (traffic) between network vertices, records that were identical, except for the ID of the animal moved, were merged and a "number of animals moved" attribute included. In total 48 FET (network) datasets were created, one for each month for the years 2003 to 2006 ($4 \times 12 = 48$). Not all attributes included in the CTS database were included in the cattle movement FET datasets. Those selected are given in Tables 3.3 and 3.4. These were selected because of their generic nature, it was considered that more specific attributes would not contribute to the generation of generally applicable Movement Patterns. For example, the holding area ID attribute would not be generic enough. Some statistics concerning the generated FET datasets, grouped according to year, are given in Table 3.5.

TABLE 3.3: CTS FET data Location Attributes and Value Types the set (A_V)

Attribute Name	**Value Type**
Sender and Reciver location type	Nominal
Sender and reciver county	Nominal

TABLE 3.4: CTS FET data Edge Attributes and Value Types (the set (A_E)

Attribute Name	**Value Type**
Gender	Nominal
Breed	Nominal
Beef	Boolean
Dairy	Boolean
Number of cattle moved	Number

TABLE 3.5: Summary of the CTS FET Datasets

Year	**No of Vertices** V	**No of Types of Edges** E	**No of Edges** E
2003	59560	515	7129102
2004	39710	332	12568342
2005	48501	320	5999983
2006	56913	331	3106109

3.3 Jiayuan Social Network

This section discusses the nature of the Jiayuan Social Network (SN) database and the process of extracting a FET dataset from this SN. The source database was provided by Jiayuan.com[1], the largest internet social network website in China with more than 170 million users. The database provided comprised of $548,395$ users ($344,552$ men and $203,843$ women) and details concerning which user had messaged which other users (no information quantifying the messaging activity was available). Unlike the CTS data, the Jiayuan dataset did not feature any missing attribute-values. Each user had a profile and a set of preferences associated with it. The user profiles comprise: age, height, education, location, occupation, place of work, income, home ownership, car ownership and so on. Preferences include things like: age range, height range, education, and location.

[1]http://www.jiayuan.com

Table 3.6: Jiayuan SN Vertex Attributes and Value Types (the set A_V)

Attribute Name	**Type**
Year of birth	Nominal
Work Locatio	
Work Sub Location	
Status	
MS Mobile	
Education	
Houseing	
Auto	
Marred	
Childern	
Industry	
Privacy	
Nation	
Range Of Income	
Belief	
Match Min Age	
Match Max Age	
Match Marriage	
Match Education	
Match Work Location	
Match Sub Location	
High	Integer
Match Min Height	
Match Max Height	

Table 3.7: Jiayuan SN Edge Attributes and Value Types (the set A_E)

Attribute Name	**Value Type**
Type of edge	Nominal (Reciprocal, Non Reciprocal)
Number of messseg	Integer

The data was processed firstly so as to identify the set A_V. A total of 25 attributes were identified, thus $|A_V| = 25$. The complete set of identified user profile (Vertex) attributes, the set A_V, are listed in Table 3.6. While the two identified edge attributes, the set A_E are listed in Table 3.7.

The resulting FET database contains $3,311,076$ records. The normal distribution of the users' activity, in terms of the number of messages sent, is presented in Figure 3.5. From the figure, it can be seen that a typical of users sent 100

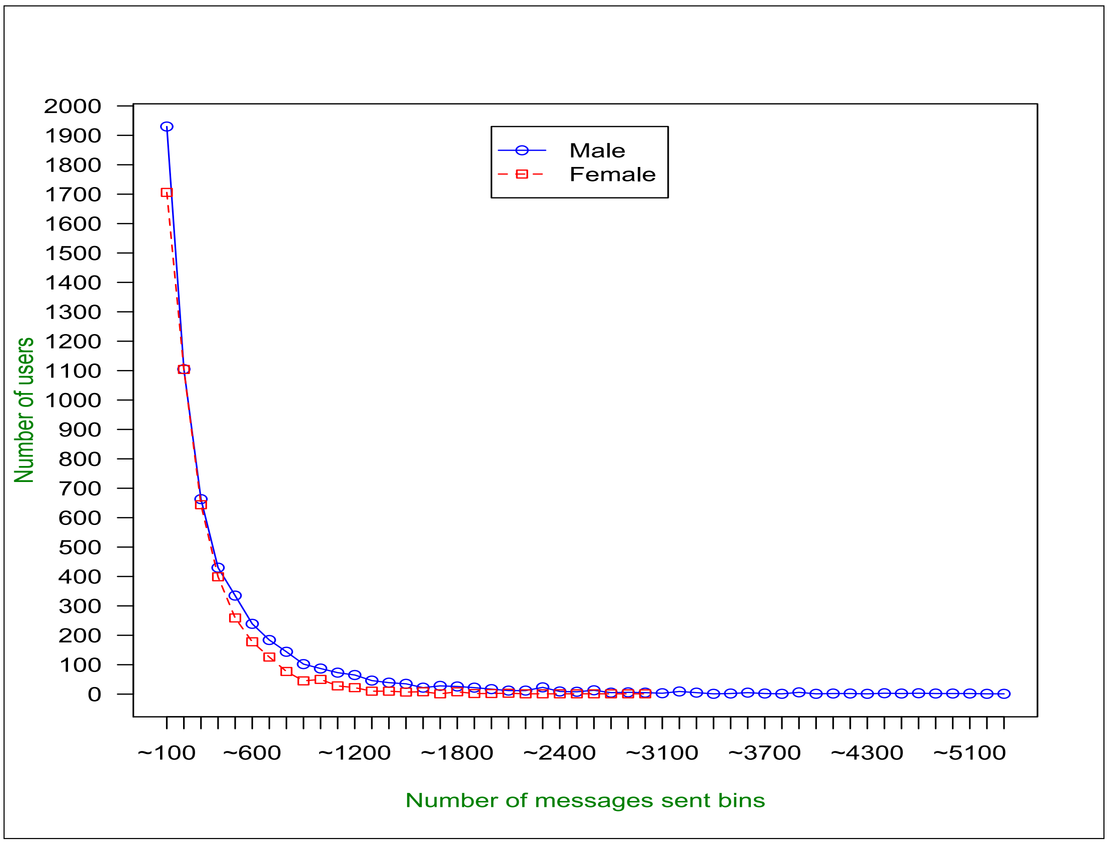


FIGURE 3.5: Male and Female Normal Distribution.

messages over the time frame covered by the data received from jiayuan.com.

3.4 Simulated Data

In addition to the above, for experimental purposes, a FET dataset generator was also constructed. The artificial FET datasets were not processed as described above but were generated so as to be in the appropriate network form so that they mimicked the nature of the CTS network discussed in 3.2.2. The pseudo code for the simulated data generator is given in Algorithm 1. The algorithm takes as input the attribute sets A_V and A_E, and outputs a set of FET. The algorithm operates by selecting randomly F and T values from A_V as vertices and E values from A_E. The purpose of the artificially generated networks was that the parameters could be controlled, especially the number of vertices and edges, and the number of attributes in the sets A_V and A_E. The *degree* for each vertex was generated randomly but under the influence of the prescribed parameters; the average number of inward or outward edges could be calculated using $|V|/|E|$. As in the case of the CTS networks (see above) the artificially generated networks were typically not fully connected in that they featured some vertices with degree zero. Further detail concerning individual, or groups, of artificial networks, will be given where relevant in the evaluation sections of Chapters 4 and 5, where results obtained with respect to the proposed MPM algorithms are considered.

Algorithm 1: Simulated Data Generator

```
   Input:
1  $A_V$ Set of vertex attribute values
2  $A_E$ Set of edge attribute values
3  $Max$ Number of iteration
   Output:
4    $G$ network inform of $FET$
5  Start:
6  $M = \emptyset$
7  while $Max \neq 0$ do
8      $F$ = select rondom vertex attribute values from $A_V$
9      $T$ = select rondom vertex attribute values from $A_V$
10     if $F \nsubseteq T$ then
11     $E$ = select rondom edge attribute values from $A_E$
12     $G = G \cup \langle F, E, T \rangle$
13     End
```

3.5 Summary

This chapter has described the nature of the datasets used for evaluating the proposed Movement Pattern Mining algorithms and Prediction Modelling presented later in this thesis. The data sources considered were: (i) the GB Cattle Tracking System (CTS) database, (ii) the Jiayuan Social Network and (iii) simulated data. The first data source represents a real example of a distribution style network where vertices represented geographic locations (holding area) and edges cattle movement. The second represented what might be considered to be a traditional social network where the vertices represented users and the edges messaging. The third provided for the generation of abstract networks where the parameters could be controlled and consequently, the effect of changing these parameters could be analysed with respect to the proposed algorithms. The next chapter considers the proposed Movement Pattern Mining algorithms.

4 Movement Pattern Mining

As noted in Chapter 1 the motivation for the research described in this thesis is the demand for knowledge relating to the behaviour of traffic in large networks, more specifically, the need for techniques to predict the nature of “traffic” movements in such networks. Consequently, this thesis proposes the usage of the concept of Movement Patterns (MPs) describing the existing traffic (communication) between vertices (nodes) in networks so as to predict future or related movement, and consequently provide support for decision and policy makers. Referring back to the research question and issues presented previously in Section 1.3, this chapter focuses on the second subsidiary question: what are the most appropriate mechanisms whereby movement patterns can be mined (learnt/extracted) from network data. The term “appropriate” in this context is defined in terms of the functionality and efficiency of the MPM. In other words are the “correct” MPs identified, and are they generated in reasonable time. Correctness in this context was measured by comparing the MPs generated using the two competing MPM algorithms proposed in this chapter; in both cases, they should produce the same set of MPs. Efficiency was measured simply in terms of runtime.

An obvious start point for the work presented in this chapter was the existing work on Frequent Pattern Mining (FPM), because of the similarity between the idea of FPs and that of MPs, and particularly the Apriori algorithm presented in

[35] that utilised the “downward closure property” of frequent itemsets. The Apriori algorithm was selected because, although well established, subsequent FPM algorithms have been similarly founded on the downward closure property. The first algorithm presented in this chapter is, therefore, an MPM algorithm founded on the Apriori algorithm, namely the Apriori Movement Pattern (AMP) algorithm. The second MPM algorithm considered is the Shape Movement Pattern (ShaMP) algorithm, founded on the observation that knowledge of the three-part nature of MPs allows for efficiency gains to be acquired by utilising this knowledge. Both algorithms are fully described and evaluated.

The remainder of this chapter is organised as follows. A formalism, used throughout the rest of this chapter, is first presented in Section 4.1. Section 4.2 then considers the proposed AMP and SHAMP algorithms in turn. The results obtained from evaluating the two proposed algorithms are then presented and discussed in Section 4.3. Finally, the chapter is concluded with a summary in Section 4.4.

4.1 Movement Pattern Formalism

This section presents the formalism for the MPM mining concept which used throughout the rest of this chapter. As noted previously in Section 3.1, a network G is conceived of as comprising a set of tuples $\langle F, E, T\rangle$. The $\langle F, E, T\rangle$ tuple then defines as a set of attribute values that represent the details of the sender (F), the receiver (T) and edge linking the two (E). The network can thus be descried in terms of a dataset D where each record is a tuple of the above form; we use the term FET data to describe such data. For example, the network presented in Figure 3.1 from Chapter 3 can be presented as a FET dataset as shown in

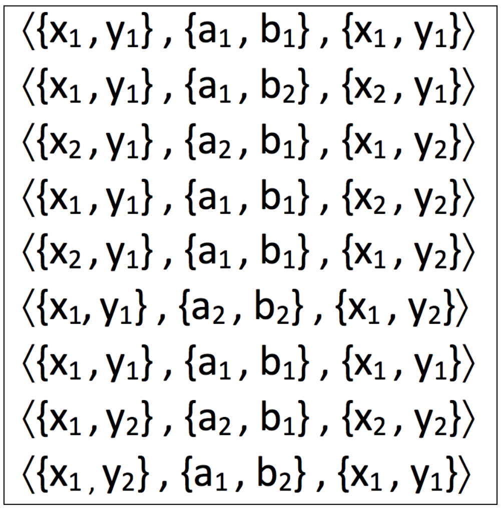
Figure 4.1 (for ease of understanding the rows are ordered according to the edge identifiers used in the figure although this is not a requirement for MPM).

An MP is thus also defined as a tuple of the form $\langle F, E, T\rangle$ where F, E and T are sets of attribute values. The minimum number of attribute values in each part must be at least one; none of the sets can be empty. The maximum number of values depends on the size of the attribute sets to which F, E, and T subscribe. An MP can only feature a maximum of one value per attribute. Thus, the maximum number of movement patterns for a given network G is thus dependent on the number of vertex and edge attributes as well as the values that these attributes can have. The maximum number of MPs given a particular data configuration is given by Equation 4.1 where V_V(sender or receiver attribute values) is the set of values associated with the attribute set A_V, and V_E (edgs attribute values) is the set of values associated with the attribute set A_E where F, $T \subset A_V$ and $E \subset V_E$.

$$|MP| = (2^{V_V} - 1) \times (2^{V_E} - 1) \times (2^{V_V} - 1) = (2^{V_V} - 1)^2 \times (2^{V_E} - 1) \qquad (4.1)$$

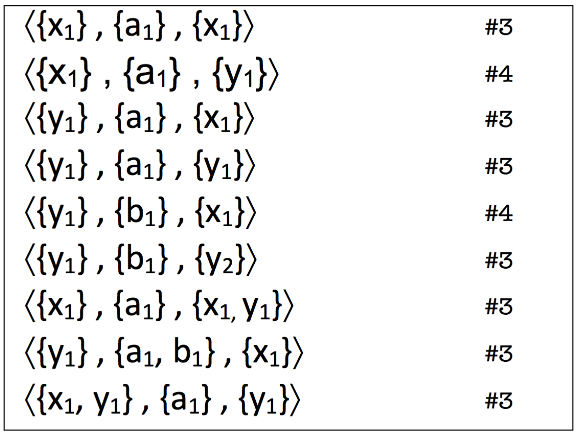
Given the above, MPM can be simplistically thought of as the process of extracting a set M of frequently occurring MPs from a dataset D (representing G) so as to build a model of D that can then be used to predict traffic or make recommendations in some previously unseen dataset D' representing a network G' comprised solely of vertices (no known edges, the edges are what we wish to predict). An MP is said to be frequent, as in the case of traditional FPM [3], if its occurrence count in D is in excess of some threshold σ expressed as a proportion of the total number of FETs in D.

Thus, with reference to the exampe network given in Figure 3.1, and assuming $\sigma = 30\%$, the set of frequent MPs, M, will be as listed in Figure 4.2, the numbers indicated using the # symbol are occurrence (frequency) counts.



$\langle\{x_1, y_1\}, \{a_1, b_1\}, \{x_1, y_1\}\rangle$
$\langle\{x_1, y_1\}, \{a_1, b_2\}, \{x_2, y_1\}\rangle$
$\langle\{x_2, y_1\}, \{a_2, b_1\}, \{x_1, y_2\}\rangle$
$\langle\{x_1, y_1\}, \{a_1, b_1\}, \{x_2, y_2\}\rangle$
$\langle\{x_2, y_1\}, \{a_1, b_1\}, \{x_1, y_2\}\rangle$
$\langle\{x_1, y_1\}, \{a_2, b_2\}, \{x_1, y_2\}\rangle$
$\langle\{x_1, y_1\}, \{a_1, b_1\}, \{x_1, y_1\}\rangle$
$\langle\{x_1, y_2\}, \{a_2, b_1\}, \{x_2, y_2\}\rangle$
$\langle\{x_1, y_2\}, \{a_1, b_2\}, \{x_1, y_1\}\rangle$

FIGURE 4.1: FET dataset, presented in tabular form, for the network given in Figure 3.1.



$\langle\{x_1\}, \{a_1\}, \{x_1\}\rangle$	#3
$\langle\{x_1\}, \{a_1\}, \{y_1\}\rangle$	#4
$\langle\{y_1\}, \{a_1\}, \{x_1\}\rangle$	#3
$\langle\{y_1\}, \{a_1\}, \{y_1\}\rangle$	#3
$\langle\{y_1\}, \{b_1\}, \{x_1\}\rangle$	#4
$\langle\{y_1\}, \{b_1\}, \{y_2\}\rangle$	#3
$\langle\{x_1\}, \{a_1\}, \{x_1, y_1\}\rangle$	#3
$\langle\{y_1\}, \{a_1, b_1\}, \{x_1\}\rangle$	#3
$\langle\{x_1, y_1\}, \{a_1\}, \{y_1\}\rangle$	#3

FIGURE 4.2: The set of frequent MPs (M) extracted from the data listed in Figure 3.1 using $\sigma = 35\%$.

4.2 Movement Pattern Mining

Two different MPM approaches are presented in this chapter, the Apriori Movement Pattern (AMP) approach, and the Shape Movement Pattern (ShaMP) approach. This section details both approaches. The two approaches are similar in the types of data they can handle and also the types of patterns they can extract from the input data, the only distinguishing feature between them is the mechanism for extracting the Movement Patterns as described in the next two sub-sections. Given that both approaches are directed at the same end result, the main issue of concern is their respective runtimes with respect to the size of the network under consideration and the threshold σ.

4.2.1 Apriori Movement Pattern Mining Algorithm

This section details the proposed Apriori Movement Pattern (AMP) mining algorithm. The first algorithm developed within the context of the research methodology presented earlier in Chapter 1. As noted above the AMP algorithm is founded on the idea of Apriori FPM as presented in [75]. Because of the overlap between the MP concept, and more traditional FPs, this seemed a natural place to start.

The concept of Apriori, as already noted above, is founded on the downward closure property of item sets. The property that if an itemset is not frequent none of its supersets can be frequent. The property can be usefully employed to limit candidate frequent itemset generation. The approach operates by starting with a set C_k of candidate itemsets of length $k = 1$. Those items in C_k that are not frequent are pruned and the remainder used to generate the $k = k + 1$ candidate item sets. The process continues until C_k is empty. Thus Apriori FPM operates level by level in a breadth-first search manner.

The pseudo code for the AMP algorithm is presented in Algorithm 2. The input is: (i) a binary valued dataset (where each value represents the presence or otherwise of a particular attribute value), represented in terms of a FET dataset D; (ii) a desired support threshold σ; (iii) a set of vertex attribute values V_V and (iv) a set of edge attribute values V_E. The output is a set of frequently occurring MPs stored in a set M, together with their support values, represented as a tuple of the form $\langle MP_i, count_i \rangle$. As in the case of the Apriori FPM algorithm, the AMP algorithm adopts a candidate generation, occurrence count and prune cycle. However, the distinction is that we are dealing with three-part MPs ($FETs$) and that none of these parts should be empty. Thus, where the variable k in the traditional Apriori algorithm refers to frequent itemset size (starting with $k = 1$), the variable k in the AMP algorithm refers to levels and the term itemset to a set of MPs.

Returning to Algorithm 2 the process commences (line 9) by generating the level 1 ($k = 1$) candidate items sets; the pseudo code for this is given in Algorithm 3. We then enter a count-prune-generate loop (lines 10 to 16) and build up the set M holding the desired frequent MPs, candidate MPs whose occurrence (frequency) count is greater or equal to *sigma*. At the end of each iteration the next level, $k + 1$, of candidates MPs is generated by adding an attribute value to one of the components of the previously identified MPs. Because of the three-part nature of MPs this is not as straightforward as in the case of traditional FPM; it is not simply a matter of uniformly "growing" K-itemsets to give $K + 1$-itemsets. The candidate generation process is presented in Algorithm 4. The count-prune-generate loop repeats until no more candidates can be generated.

Algorithm 2: Apriori based Movement Pattern (AMP) algorithm

```
   Input:
1  $D$ = Binary valued input data set
2  $\sigma$ = Support threshold
3  $V_V$ Set of vertex attribute values
4  $V_E$ Set of edges attribute value
   Output:
5  $M$ = Set of frequently occurring MPs $\{\langle MP_1, count_1\rangle, \langle MP_2, count_2\rangle, \ldots\}$
6  Start:
7  $k$ = level 1
8  $M = \emptyset$
9  $C_k$ = generateFirstSetOfCandidates( $V_V$, $V_E$) (see Algorithm 3)
10 while $C_k \neq \emptyset$ do
11     $S_k$ = Set of occurance counts for candidates in $C_k$ with reference to $D$
12     $C_k$ = Set of candidate in $C_k$ where occurrence counts $S_k$ are greater
         than $\sigma$
13     $M = M \cup C_k$
14     $k = k + 1$
15     $C_k$ = generateCandidates($C_{k-1}$, $V_V$, $V_E$) (see Algorithm 4)
16 end
```

Algorithm 3, called from Algorithm 2, shows the pseudo code for the generation of the $k = 1$ candidate MPs. The inputs are the set of location (vertex) attribute values A_V and the set of edge attribute values V_E. The output is a set of $k = 1$ candidate MPs, C_k. The generation process comprises a set of nested loops where each loop steps through one of the attribute value sets and incrementally builds up each candidate MP in a depth-first manner.

Algorithm 3: Generate First Set Of Candidates

```
procedure generateFirstSetOfCandidates($V_V$, $V_E$)
    Input: $V_V$, $V_E$
    Output: $C_k$
  1 Set $C_k = \emptyset$
forall $a_i \in V_V$ do
    forall $a_j \in V_E$ do
        forall $a_k \in V_V$ do
            $C_k = C_k \bigcup \langle a_i, a_j, a_k \rangle$
    return $C_k$
end procedure
```

Algorithm 4, also called from Algorithm 2, gives the pseudo code for the candidate MP generation process where $k > 1$. The input is the set of frequent MPs from the previous iteration, C_{k-1}, and the sets V_V and V_E. The output, as in the case of Algorithm 3, is the level k set of candidate MPs, C_k. To generate the candidate set we add an attribute value to the frequent MPs from the previous iteration in such a way that the overall size (number of attribute values) of the MPs increases by one. Of course, we can only add an attribute value if it is not already contained in the current MP.

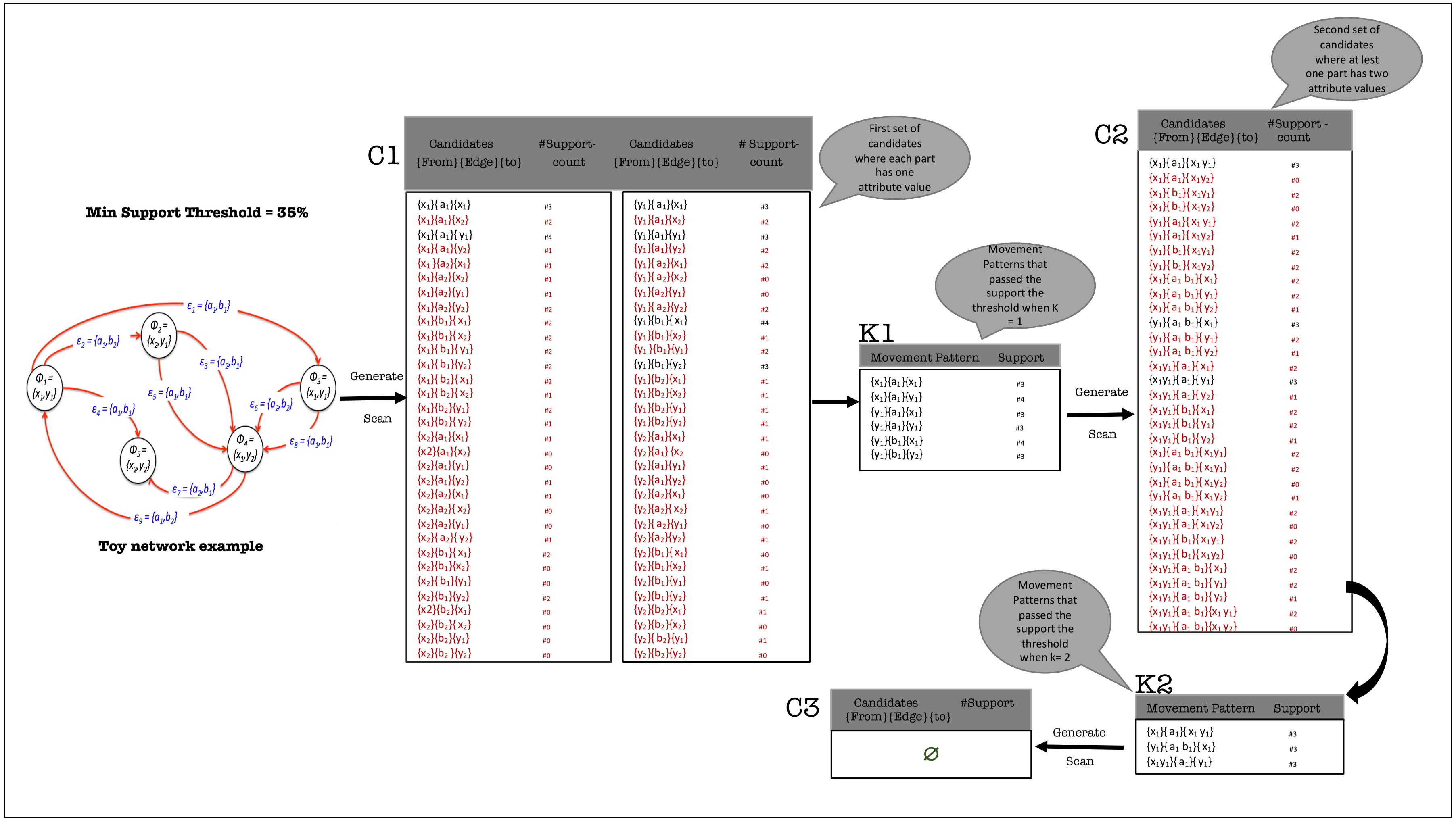
The operation of the AMP algorithm is illustrated in Figure 4.3 using a worked example. As already noted the algorithm takes as input a network in FET form. In the figure the network is shown on the left-hand side, the FET representation is not included. On the first iteration of the approach, the $k = 1$ set of candidates, C_k, is generated, where each part has one attribute value. This set of candidates, a total of 64 possibilities, is listed in the center of the figure. The FET encoded

Algorithm 4: Generate Candidates

```
procedure generateCandidates($C_{k-1}$,$V_V$, $V_E$)
    Input: $C_{k-1}$ Set of frequent MPs from previous generation
$V_V$ Set of vertex attribute values
$V_E$ Set of Edges attribute value
    Output: $C_k$ Set of level $K$ candidates
  1 Set $C_k = \emptyset$
forall $\langle x, y, z\rangle \in C_{k-1}$ do
    forall $a_i \in V_V$ do
        if $a_i \notin x$ then $C_k = C_k \bigcup \langle\{x, a_i\}, y, z\rangle$ ;
    forall $a_j \in V_E$ do
        if $a_j \notin y$ then $C_k = C_k \bigcup \langle x, \{y, a_j\}, z\rangle$ ;
    forall $a_k \in V_V$ do
        if $a_k \notin z$ then $C_k = C_k \bigcup \langle x, y, \{z, a_k\}\rangle$ ;
    return $C_k$
end procedure
```

network is then scanned so as to generate frequency counts, also included in the table given in the middle of the Figure. Any candidate set whose support count is below the threshold σ is then pruned, these are indicated in red in the figure ($\sigma = 35$ is assumed). This results in a set k_1, of six MPs. We then find the $k + 1 = 2$ set of candidates, C_2, where at least one part of each MP has two attribute values, as shown on the right hand side of the figure. The process is repeated until no more candidates can be generated, $C_k = \emptyset$. In the example, this happens when $k = 3$. Therefore, the complexity of Apriori-based Movement Pattern (AMP) algorithm is:

1. Candidate generation: Generation of C_k candidate itemset = $\mathcal{O}(\ |F_{k-1}| \times |F_1|\ )$; where F = Frequent itemsets.
2. Support counting = $\mathcal{O}(|C_k| \times n)$; where n = number of records.
3. Complexcity of one iteration $k = \mathcal{O}\left((|F_{k-1}| + |F_1) + (|C_k| + n)\right)$.
4. Overall complexity = $\mathcal{O}\left(|F_1| \times \sum_{k=2}^{k=max} (|F_{k-1} \times |F_1|) + (|c_k| + n)\right)$


Min Support Threshold = 35%
$\Phi_1=\{x_1,y_1\}$
$\Phi_2=\{x_2,y_1\}$
$\Phi_3=\{x_1,y_1\}$
$\Phi_4=\{x_1,y_2\}$
$\Phi_5=\{x_2,y_2\}$
$\varepsilon_1=\{a_1,b_1\}$
$\varepsilon_2=\{a_1,b_2\}$
$\varepsilon_3=\{a_2,b_1\}$
$\varepsilon_4=\{a_1,b_1\}$
$\varepsilon_5=\{a_2,b_1\}$
$\varepsilon_6=\{a_2,b_2\}$
$\varepsilon_7=\{a_2,b_1\}$
$\varepsilon_8=\{a_1,b_1\}$
$\varepsilon_9=\{a_1,b_2\}$
Toy network example
Generate
Scan
C1
First set of candidates where each part has one attribute value
K1
Movement Patterns that passed the support the threshold when K = 1
Generate
Scan
C2
Second set of candidates where at lest one part has two attribute values
K2
Movement Patterns that passed the support the threshold when k= 2
Generate
Scan
C3

C1

Candidates {From}{Edge}{to}	#Support-count	Candidates {From}{Edge}{to}	# Support-count
$\{x_1\}\{a_1\}\{x_1\}$	#3	$\{y_1\}\{a_1\}\{x_1\}$	#3
$\{x_1\}\{a_1\}\{x_2\}$	#2	$\{y_1\}\{a_1\}\{x_2\}$	#2
$\{x_1\}\{a_1\}\{y_1\}$	#4	$\{y_1\}\{a_1\}\{y_1\}$	#3
$\{x_1\}\{a_1\}\{y_2\}$	#1	$\{y_1\}\{a_1\}\{y_2\}$	#2
$\{x_1\}\{a_2\}\{x_1\}$	#1	$\{y_1\}\{a_2\}\{x_1\}$	#2
$\{x_1\}\{a_2\}\{x_2\}$	#1	$\{y_1\}\{a_2\}\{x_2\}$	#0
$\{x_1\}\{a_2\}\{y_1\}$	#1	$\{y_1\}\{a_2\}\{y_1\}$	#0
$\{x_1\}\{a_2\}\{y_2\}$	#2	$\{y_1\}\{a_2\}\{y_2\}$	#2
$\{x_1\}\{b_1\}\{x_1\}$	#2	$\{y_1\}\{b_1\}\{x_1\}$	#4
$\{x_1\}\{b_1\}\{x_2\}$	#2	$\{y_1\}\{b_1\}\{x_2\}$	#1
$\{x_1\}\{b_1\}\{y_1\}$	#2	$\{y_1\}\{b_1\}\{y_1\}$	#2
$\{x_1\}\{b_1\}\{y_2\}$	#2	$\{y_1\}\{b_1\}\{y_2\}$	#3
$\{x_1\}\{b_2\}\{x_1\}$	#2	$\{y_1\}\{b_2\}\{x_1\}$	#1
$\{x_1\}\{b_2\}\{x_2\}$	#1	$\{y_1\}\{b_2\}\{x_2\}$	#1
$\{x_1\}\{b_2\}\{y_1\}$	#2	$\{y_1\}\{b_2\}\{y_1\}$	#1
$\{x_1\}\{b_2\}\{y_2\}$	#1	$\{y_1\}\{b_2\}\{y_2\}$	#1
$\{x_2\}\{a_1\}\{x_1\}$	#1	$\{y_2\}\{a_1\}\{x_1\}$	#1
$\{x2\}\{a_1\}\{x_2\}$	#0	$\{y_2\}\{a_1\}\{x_2$	#0
$\{x_2\}\{a_1\}\{y_1\}$	#0	$\{y_2\}\{a_1\}\{y_1\}$	#1
$\{x_2\}\{a_1\}\{y_2\}$	#1	$\{y_2\}\{a_1\}\{y_2\}$	#0
$\{x_2\}\{a_2\}\{x_1\}$	#1	$\{y_2\}\{a_2\}\{x_1\}$	#0
$\{x_2\}\{a_2\}\{x_2\}$	#0	$\{y_2\}\{a_2\}\{x_2\}$	#1
$\{x_2\}\{a_2\}\{y_1\}$	#0	$\{y_2\}\{a_2\}\{y_1\}$	#0
$\{x_2\}\{a_2\}\{y_2\}$	#1	$\{y_2\}\{a_2\}\{y_2\}$	#1
$\{x_2\}\{b_1\}\{x_1\}$	#2	$\{y_2\}\{b_1\}\{x_1\}$	#0
$\{x_2\}\{b_1\}\{x_2\}$	#0	$\{y_2\}\{b_1\}\{x_2\}$	#1
$\{x_2\}\{b_1\}\{y_1\}$	#0	$\{y_2\}\{b_1\}\{y_1\}$	#0
$\{x_2\}\{b_1\}\{y_2\}$	#2	$\{y_2\}\{b_1\}\{y_2\}$	#1
$\{x2\}\{b_2\}\{x_1\}$	#0	$\{y_2\}\{b_2\}\{x_1\}$	#1
$\{x_2\}\{b_2\}\{x_2\}$	#0	$\{y_2\}\{b_2\}\{x_2\}$	#0
$\{x_2\}\{b_2\}\{y_1\}$	#0	$\{y_2\}\{b_2\}\{y_1\}$	#1
$\{x_2\}\{b_2\}\{y_2\}$	#0	$\{y_2\}\{b_2\}\{y_2\}$	#0

K1

Movement Pattern	Support
$\{x_1\}\{a_1\}\{x_1\}$	#3
$\{x_1\}\{a_1\}\{y_1\}$	#4
$\{y_1\}\{a_1\}\{x_1\}$	#3
$\{y_1\}\{a_1\}\{y_1\}$	#3
$\{y_1\}\{b_1\}\{x_1\}$	#4
$\{y_1\}\{b_1\}\{y_2\}$	#3

C2

Candidates {From}{Edge}{to}	#Support - count
$\{x_1\}\{a_1\}\{x_1 y_1\}$	#3
$\{x_1\}\{a_1\}\{x_1y_2\}$	#0
$\{x_1\}\{b_1\}\{x_1y_1\}$	#2
$\{x_1\}\{b_1\}\{x_1y_2\}$	#0
$\{y_1\}\{a_1\}\{x_1 y_1\}$	#2
$\{y_1\}\{a_1\}\{x_1y_2\}$	#1
$\{y_1\}\{b_1\}\{x_1y_1\}$	#2
$\{y_1\}\{b_1\}\{x_1y_2\}$	#2
$\{x_1\}\{a_1 b_1\}\{x_1\}$	#2
$\{x_1\}\{a_1 b_1\}\{y_1\}$	#2
$\{x_1\}\{a_1 b_1\}\{y_2\}$	#1
$\{y_1\}\{a_1 b_1\}\{x_1\}$	#3
$\{y_1\}\{a_1 b_1\}\{y_1\}$	#2
$\{y_1\}\{a_1 b_1\}\{y_2\}$	#1
$\{x_1y_1\}\{a_1\}\{x_1\}$	#2
$\{x_1y_1\}\{a_1\}\{y_1\}$	#3
$\{x_1y_1\}\{a_1\}\{y_2\}$	#1
$\{x_1y_1\}\{b_1\}\{x_1\}$	#2
$\{x_1y_1\}\{b_1\}\{y_1\}$	#2
$\{x_1y_1\}\{b_1\}\{y_2\}$	#1
$\{x_1\}\{a_1 b_1\}\{x_1y_1\}$	#2
$\{y_1\}\{a_1 b_1\}\{x_1y_1\}$	#2
$\{x_1\}\{a_1 b_1\}\{x_1y_2\}$	#0
$\{y_1\}\{a_1 b_1\}\{x_1y_2\}$	#1
$\{x_1y_1\}\{a_1\}\{x_1y_1\}$	#2
$\{x_1y_1\}\{a_1\}\{x_1y_2\}$	#0
$\{x_1y_1\}\{b_1\}\{x_1y_1\}$	#2
$\{x_1y_1\}\{b_1\}\{x_1y_2\}$	#0
$\{x_1y_1\}\{a_1 b_1\}\{x_1\}$	#2
$\{x_1y_1\}\{a_1 b_1\}\{y_1\}$	#2
$\{x_1y_1\}\{a_1 b_1\}\{y_2\}$	#1
$\{x_1y_1\}\{a_1 b_1\}\{x_1 y_1\}$	#2
$\{x_1y_1\}\{a_1 b_1\}\{x_1 y_2\}$	#0

K2

Movement Pattern	Support
$\{x_1\}\{a_1\}\{x_1 y_1\}$	#3
$\{y_1\}\{a_1 b_1\}\{x_1\}$	#3
$\{x_1y_1\}\{a_1\}\{y_1\}$	#3

C3

Candidates {From}{Edge}{to}	#Support
$\varnothing$	

FIGURE 4.3: Working example illustrating the operation of the AMP algorithm ($\sigma = 35\%$).

Also, the major limitations of the AMP algorithm are:

- The large number of candidate patterns that are generated during the process, which limits the size of the networks that can be reasonably considered.
- The excessive runtimes (partly as a consequence of the above) required to effectively mine large FET datasets.

In the context of runtime it should be noted that AMP runtime is inversely proportional to the σ support threshold value used, however low values of σ are required to ensure that no significant MPs are missed. Evidence concerning the above limitations will be presented in the evaluation section given later in this chapter. The proposed SHAMP algorithm, presented in the following sub-section, seeks to address the disadvantages associated with the AMP approach.

4.2.2 Shape Movement Pattern Mining Algorithm

The ShaMP algorithm was motivated by the above listed disadvantages associated with the AMP algorithm. More specifically, because the thesis is aimed at investigating large networks, the identification of MPs without candidate set generation. The novel element of the ShaMP algorithm is that it is founded on the idea of candidate shapes rather than candidate itemsets. A shape in this context is a movement pattern template with a particular configuration of attributes taken from A_V and A_E (note that individual shape definitions do not specify particular attribute values or combinations of values). Thus, for example, we might have a shape:

$$\langle \{A_1, A_2\}, \{E_1, E_2\}\{A_3, A_4\} \rangle$$

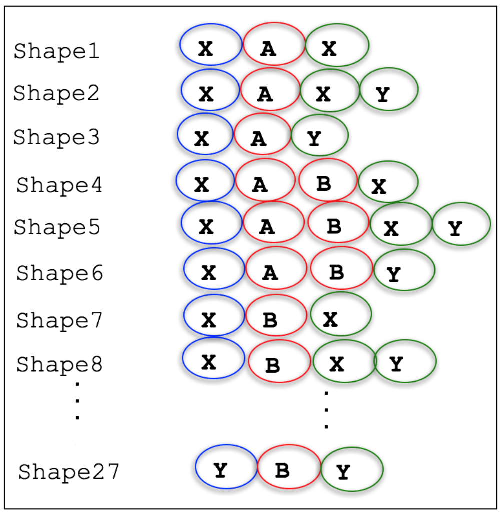


FIGURE 4.4: The 27 different shapes when $|A_V| = 2$ and $|A_E| = 2$, as in the case of the example network given in Figure 3.1.

where A_1, A_2, A_3 and A_4 are in A_V, $A_1 \neq A_2$ and $A_3 \neq A_4$, although A_1, can equal A_3 or A_4, and/or A_2 can equal A_3 or A_4. Similarly E_1 and E_2 are in A_E and $E_1 \neq E_2$.

The total number of shapes that can exist in a FET dataset D, under normal circumstances, is significantly less than the number of candidates that exist in the same dataset D. The total number of shapes can be calculated using Equation 4.2, where: (i) $|A_V|$ is the size of the attribute set A_V and (ii) $|A_E|$ is the size of the attribute set A_E. Recall that attributes for F and T are drawn from the same domain. Thus if $|A_V| = 2$ and $|A_E| = 2$, as in the case of the movement network given in Figure 3.1, there will be $(2^2 - 1) \times (2^2 - 1) \times (2^2 - 1) = 3 \times 3 \times 3 = 27$ different shapes as listed in Figure 4.4. If we increase $|A_E|$ to 5 there will be $(2^2 - 1) \times (2^5 - 1) \times (2^2 - 1) = 3 \times 31 \times 3 = 279$ different shapes, and so on. The efficiency of the ShaMP algorithm is directly related to the number of different shapes that need to be considered, this will be explored further in Section 4.3 below.

$$(2^{|A_L|} - 1) \times (2^{|A_E|} - 1) \times (2^{|A_L|} - 1) \tag{4.2}$$

The pseudo code for the ShaMP Algorithm is given in Algorithm 5. The input is: (i) a binary valued input data set, represented in terms of a FET dataset D; (ii) a desired support threshold σ; (iii) a set of vertex attributes A_V; and (iv) a set of Edges attributes A_E. The output is set of frequently occurring MPs, stored in a set M, together with their support values, represented as a tuple of the form $\langle MP_i, count_i \rangle$. The algorithm commences (line 7) by generating the available set of shapes, $ShapeSet$, by calling Algorithm 6, and then setting M to the empty set (line 8). The algorithm then loops through the $ShapeSet$ (lines 9 to 16); and, for each shape, loops through D comparing each record $r_j \in D$ with the current shape, $shape_i$. A record r_j matches a $shape_i$ if the attributes featured in the shape also feature in r_j. Where a match is found, the relevant attribute values in r_i form an MP. If the identified MP is already contained in M we simply update the associated support value, otherwise, we add the MP to M with a support value of 1. Once all shapes have been processed we loop through M (lines 17 to 19) and remove all MPs whose support count is less than σ.

The pseudo code for the shape generation is given in Algorithm 6. The algorithm takes as input the attribute sets A_V and A_E (not attribute value sets as used in the case of the AMP approach described earlier) and outputs a set of shapes. The algorithm operates in a similar manner to Algorithm 3 except that we are dealing with attribute sets rather than attribute value sets.

A worked example illustrating the operation of the ShaMP algorithm is given in Figure 4.5 using the same input data as for the AMP worked example given in Figure 4.3; left hand side of the figure. The set of shapes is given in the middle-left of the figure. Recall that each indivual shape consists of three parts with a particular configuration of attributes taken from A_V and A_E. The list of shapes, in this case, is the same as the list given in Figure 4.4. Only some of the shapes exist in the input data. Note also that for each shape there will be a number of alternative MP that fit the shape; some examples are indicated in the figure

Algorithm 5: Shape-based Movement Pattern Algorithm

```
Input:
$D$ = Binary valued input data set
$\sigma$ = Support threshold
$A_V$ = Set of vertex attributes
$A_E$ = Set of Edges attributes
Output:
  $M$ = Set of frequently occurring MPs $\{\langle MP_1, count_1\rangle, \langle MP_2, count_2\rangle, \ldots\}$
Start:
$ShapeSet$ = Generate shapes set $\{shape_1, \ldots, shape_n\}$ (See Algorithm 6)
M = $\emptyset$
forall $shape_i \in ShapeSet$ do
    forall $r_i \in D$ do
        if $r_i$ matches $shape_i$ then
        $MP_k$ = MP extracted from $r_i$
        if $MP_k$ in $M$ then increment support
        else $M = M \bigcup \langle MP_k, 1\rangle$
    end
end
 forall $MP_i \in M$ do
    if count for $MP_i < \sigma$ then remove $MP_i$ from $M$
end
```

Algorithm 6: Generate Shapes

```
    procedure GENERATESHAPES($A_V$, $A_E$)
        Input: $A_V$ Set of vertex attributes; $A_E$ Set of Edges attributes
        Output: $ShapeSet$
      1 Set $ShapeSet = \emptyset$
    forall $a_v \in A_V$ do
        forall $a_e \in A_E$ do
            forall $a_v \in A_V$ do
                $ShapeSet = ShapeSet \bigcup \langle a_v, a_e, a_v\rangle$
        return $ShapeSet$
    end procedure
```

together with their frequency counts. The final stage is to prune the identified MPs that are not frequent according to the predefined σ threshold. Therefore, The complexity of Shape-based Movement Pattern Algorithm is:

1. Shape generation = $\mathcal{O}($ number of shape$)$.
2. MP generation = $\mathcal{O}(n \times$ number of shape$)$; where $n =$ number of records.
3. Overall complexity = $\mathcal{O}$ (Shape generation + (n + number of shape))

In comparison with the AMP algorithm, the ShaMP algorithm offers four advantages:

- The nature of the value of σ has very little (no?) effect on the algorithm's runtime (as will be demonstrated later in this chapter in Section 4.3.1).
- It is faster than AMP when the number of shapes is relatively small, which in turn is dictated by the values for $|A_V|$ and $|A_E|$.
- There are no conflicts between shapes as each individual shape can be considered in isolation, hence the algorithm is well suited to parallelisation. Not the case with respect to the AMP algorithm which operates level by level in a top-down manner and, when parallelised, will feature significant message passing after each level.
- The nature of the shapes can be controlled in that it would be possible to specify only the (application dependent) particular shapes we might be interested in.

Evidence with respect to the first two of the above advantages will be presented in the following section where the evaluation of the two algorithms, AMP and ShaMP, is reported on. The third advantage will be elaborated on in the next chapter, Chapter 5.

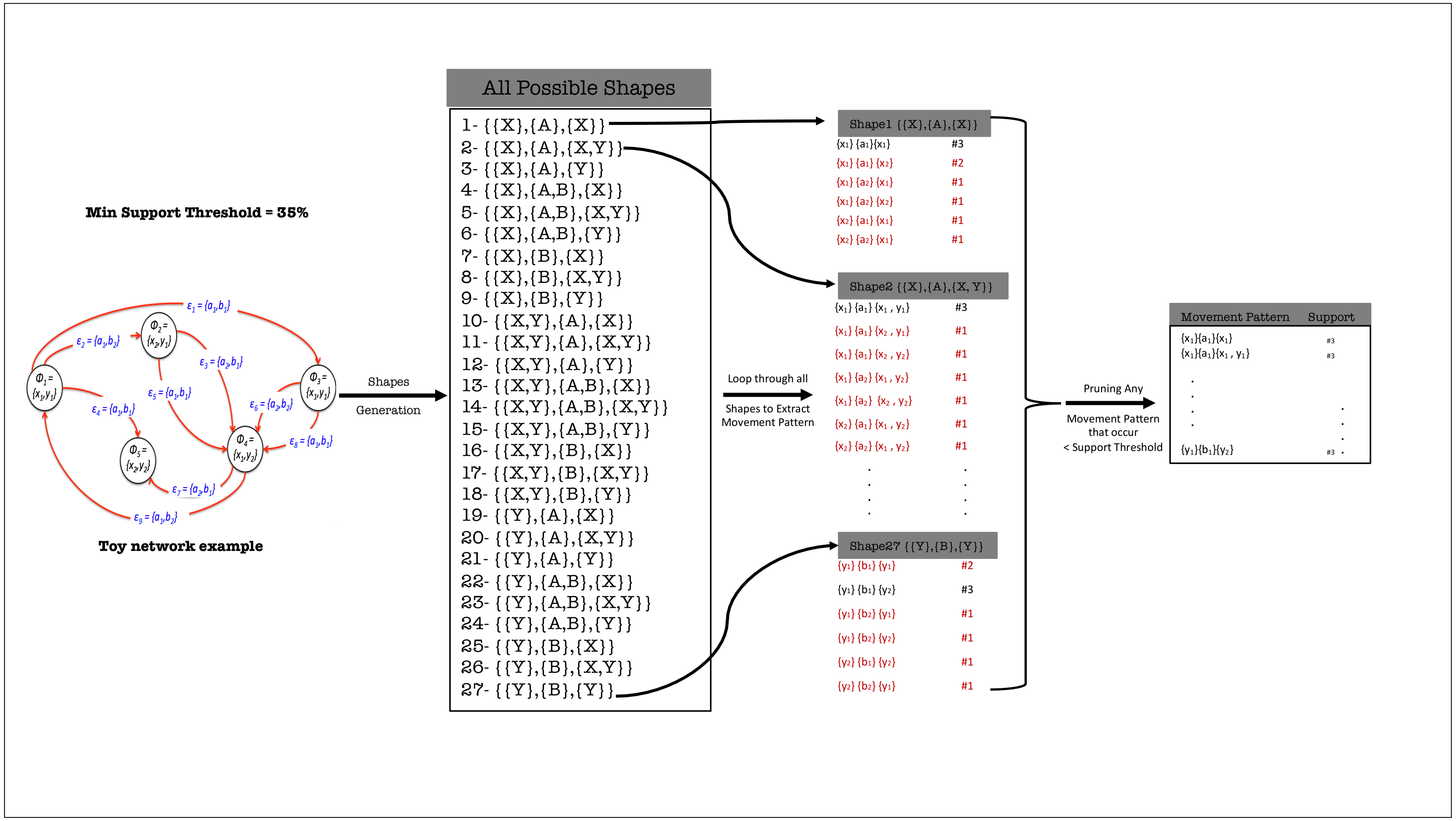


Figure 4.5: Working example illustrating the operation of the ShaMP Algorithm ($\sigma = 35\%$).

4.3 Experiments and Evaluation

This section reports on the experiments conducted to analyse the operation of the proposed ShaMP and AMP algorithms. The evaluation was conducted using two categories of network data: (i) networks extracted from the CTS database introduced in Section 3.2 and (ii) artificial networks, as introduced in Section 3.4, where the parameters can be easily controlled. The objectives of the evaluation were as follows:

1. To determine how the nature of the σ support threshold would affect the operation of the ShaMP and AMP algorithms, and if so what this effect might be.
2. To determine the effect, on the operation of the AMP and ShaMP algorithms, of the size of the networks under consideration, in terms of the size of FETs (the number of attributes featured in FETs).
3. To determine the effect, on the operation of the AMP and ShaMP algorithms of the size of the networks under consideration, in terms of the number of FETs.

The remainder of this section is divided into three Sub-sections, Subsections 4.3.1, 4.3.2 and 4.3.3. All the experiments were conducted using a single machine with a processor 3.5 GHz Intel Core i5 processor with 32 GB, 1600 MHz, DDR3 of RAM and running under the OS operating systems.

4.3.1 Effect of Selected Support Threshold

In the context of Frequent Itemset Mining (FIM) it is clear that the lower the σ threshold value the more frequent itemsets that will be identified; although low σ

Table 4.1: Recorded Runtimes (seconds) using a range of values for σ and the CTS network data

	Runtime for AMP				Runtime for ShaMP			
σ	2003	2004	2005	2006	2003	2004	2005	2006
1.0	57	54	39	62	137	137	139	140
0.75	140	171	107	161	142	143	148	149
0.5	706	600	534	535	126	128	130	137
0.25	2681	2432	2492	2749	155	158	159	162

Table 4.2: Number of frequent MPs discovered using a range of values for σ and the CTS network data

	No. of MPs using AMP				No. of MPs usin ShaMP			
σ	2003	2004	2005	2006	2003	2004	2005	2006
1.0	2299	2273	2382	2279	2299	2273	2382	2279
0.75	3264	3308	3477	3337	3264	3308	3477	3337
0.5	5610	5584	5912	5602	5610	5584	5912	5602
0.25	13153	13226	13844	13170	13153	13226	13844	13170

values are thus seen as desirable because by finding many frequent itemsets there is less chance of missing anything significant. It was anticipated that this would also be true with respect to the AMP algorithm. How this would affect the ShaMP algorithm was unclear, although it was conjectured that changing σ values would have little effect.

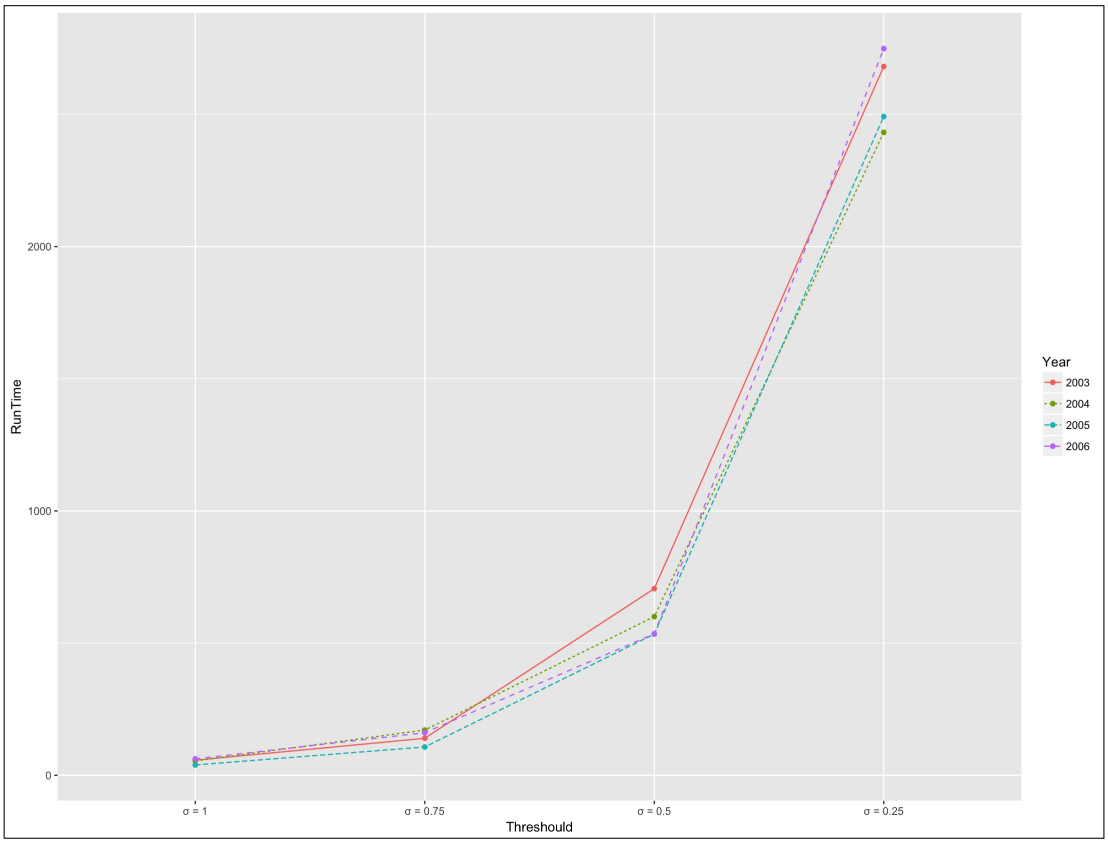
The first set of experiments conducted to analyse the effect of the σ value on the ShaMP and AMP algorithms used CTS data quouted into four movement networks according to year: 2003, 2004, 2005 and 2006. A range of σ values, from 1.0 to 0.25, decreasing in steps of 0.25, was used. The results are presented in Tables 4.1 and 4.2. Tables 4.1 gives the runtimes, in seconds, while Table 4.2 the number of MPs generated in each case.

Considering the runtime results first, these are simplest to interpret by considering them in graph form as shown in Figure 4.6. The figure presents two graphs one for the AMP algorithm runtimes and one for the ShaMP algorithm runtimes. For each graph, the x-axis gives the σ values and the y-axis the runtime values in seconds. Note that the y-axis scales are not the same for both graphs. From the graphs, it can be seen that in both cases the required runtime increases as the value of σ decreases. However, the increase is exponential in the case of the AMP algorithm whilst it is linear in the case of the ShaMP algorithm. Closer inspection of the graphs shows that the increase is much more marked with respect to the AMP algorithm than the ShaMP algorithm. In fact, the largest increase in runtime, between $\sigma = 1.00$ and $\sigma = 0.25$, using the ShaMP algorithm, is 22 seconds for the 2006 CTS network data, whilst the difference with respect to the AMP algorithm, for the same data set, is 2687 seconds. In other words, it would appear that changes in the value of σ have little effect on ShaMP runtime (as conjectured). From the graphs, it is also interesting to note that it is not till σ drops below 1.0 that usage of the ShaMP algorithm becomes more advantageous, in terms of efficiency than usage of the AMP algorithm. This last is significant because we wish to identify as many relevant MPs as possible and to do this we would need to use low σ values ($\sigma \leq 1.0$).

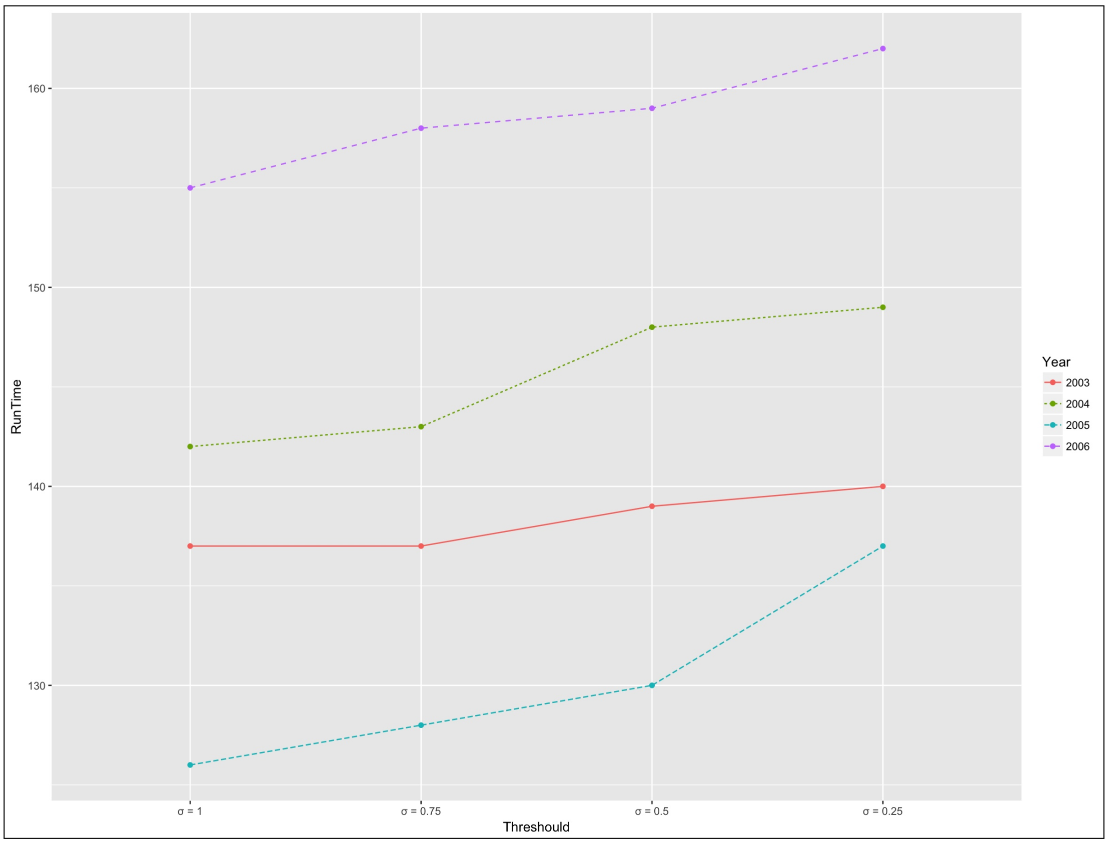
Considering the number of MPs generated (Table 4.2), the first thing to note is that both the AMP and ShaMP algorithms, working entirely independently, produced the same number of MPs, thus indicating that both algorithms are operating correctly. Plotting the ShaMP results in graph form, Figure 4.7, it can clearly be seen that the number of frequent MPs generated increases as the value of σ decreases; this is to be expected. It can also be seen that the four CTS data sets feature different numbers of MPs explaining the differences in the corresponding runtimes given in Figure 4.6.

4.3.2 Effect of Size of FETs

This subsection considers the experiments conducted to determine the effect on the AMP and ShaMP algorithms with respect to the size of the FETs considered



(a) AMP Algorithm



(b) ShaMP Algorithm

Figure 4.6: Runtimes (secs.), using ShaMP and AMP, applied to the CTS network datasets, and a range of σ values.

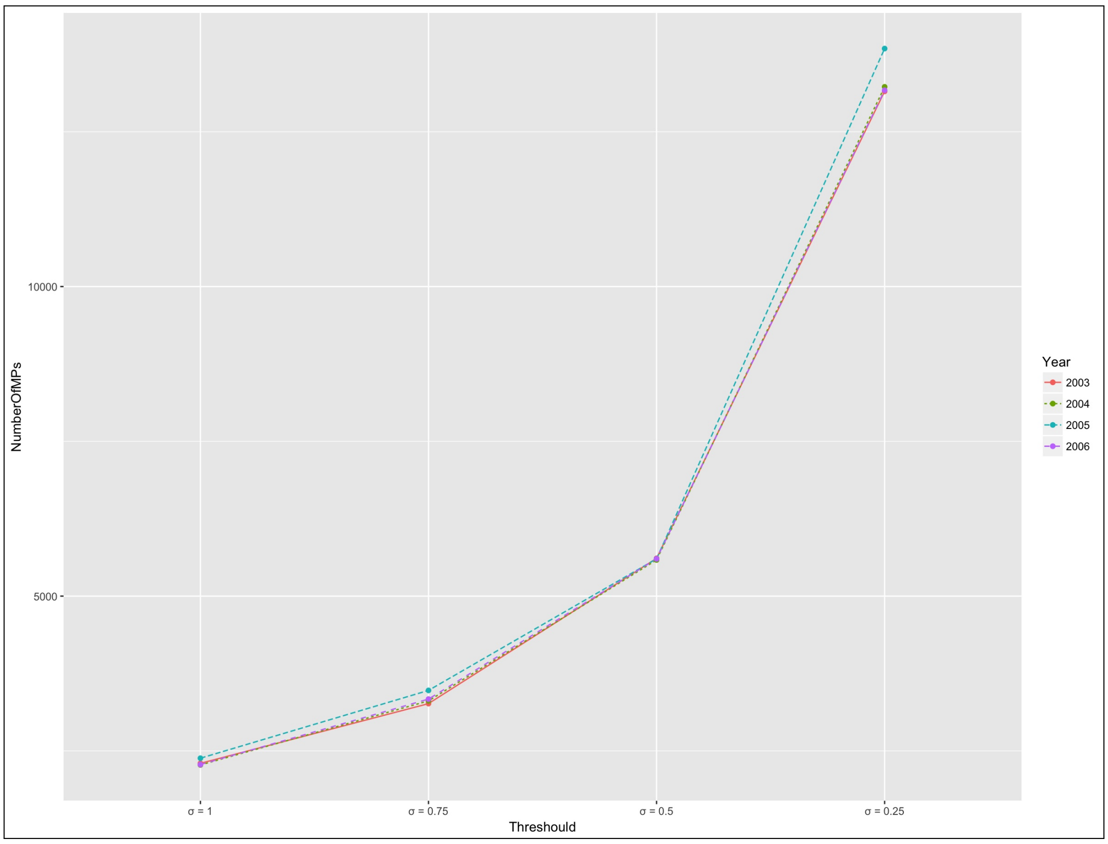


FIGURE 4.7: Number of Movement Pattern, using ShaMP and AMP, applied to the CTS network datasets, and a range of σ values.

TABLE 4.3: Number of ShaMP shapes for a range of different A_V and A_E configurations $|A_E|$ 2 to 10 in steps of 2 and $|A_V|$ from 1 to 5 in steps of 1

A_V	A_E				
	2	4	6	8	10
1	3	15	63	255	1023
2	27	135	567	2295	9207
3	147	735	3087	12495	50127
4	675	3375	14175	57375	230175
5	2883	14415	60543	245055	983103

in terms of the number of attributes.

Before considering the conducted experimentation it should be noted that the time complexity of the ShaMP algorithm can be calculated in terms of the number of shapes to be considered. Table 4.3 shows the number of shapes associate with different configurations of A_V and A_E. From the table, it can be seen that as the size of A_V or A_E is increased the number of shapes increases exponentially.

For the evaluation, a sequence of artificial datasets were generated where the number of shapes increased from 500 to 5000 in steps of 500. The value for $|A_V|$ was maintained at a constant so as to allow the AMP algorithm to perform to its best advantage where each attribute had two possible attribute-values. For the experiments $\sigma = 2$ was used, as discussed in Sub-section 4.3.1 using low σ value can affect the runtime of the AMP algorithm. The results are presented in Figure 4.8. In the figure, the x-axis represents the number of shapes (an amalgamation of $|A_V|$ and $|A_E|$ and not just $|A_E|$). It was thus conjectured that there must be a "cut-off" point where the number of shapes was such that any efficiency benefits that the ShaMP algorithm might have to offer, through avoiding candidate generation, were negated to the extent that it would be more effective to use the AMP algorithm instead (when using a single machine).

Figure 4.8 shows the impact of increasing the number of shapes with respect to A_V and A_E on the runtime of the ShaMP and AMP algorithms. From the Figure, the first observation that can be made, as to be expected, is that the runtime for both algorithms increased as the number of Shapes (attribute values) increased. Inspection of the table also demonstrates that this "cross-over" point ("cut-off" point), when it is no longer effective to use the ShaMP algorithm, is when the number of shapes reaches about $2,600$. In practice, given the applications where the proposed MP mining might be applied, it was anticipated that the traffic of interest could normally be described in terms of $2,600$ attributes or less. Reference to Table 4.3 indicates the kinds of combinations of A_V and A_E that can be considered. It should also be noted that for most applications the number of attribute values can be expected to be considerably more than 2 as used in the reported experiments, in which case the cut off point will be higher.

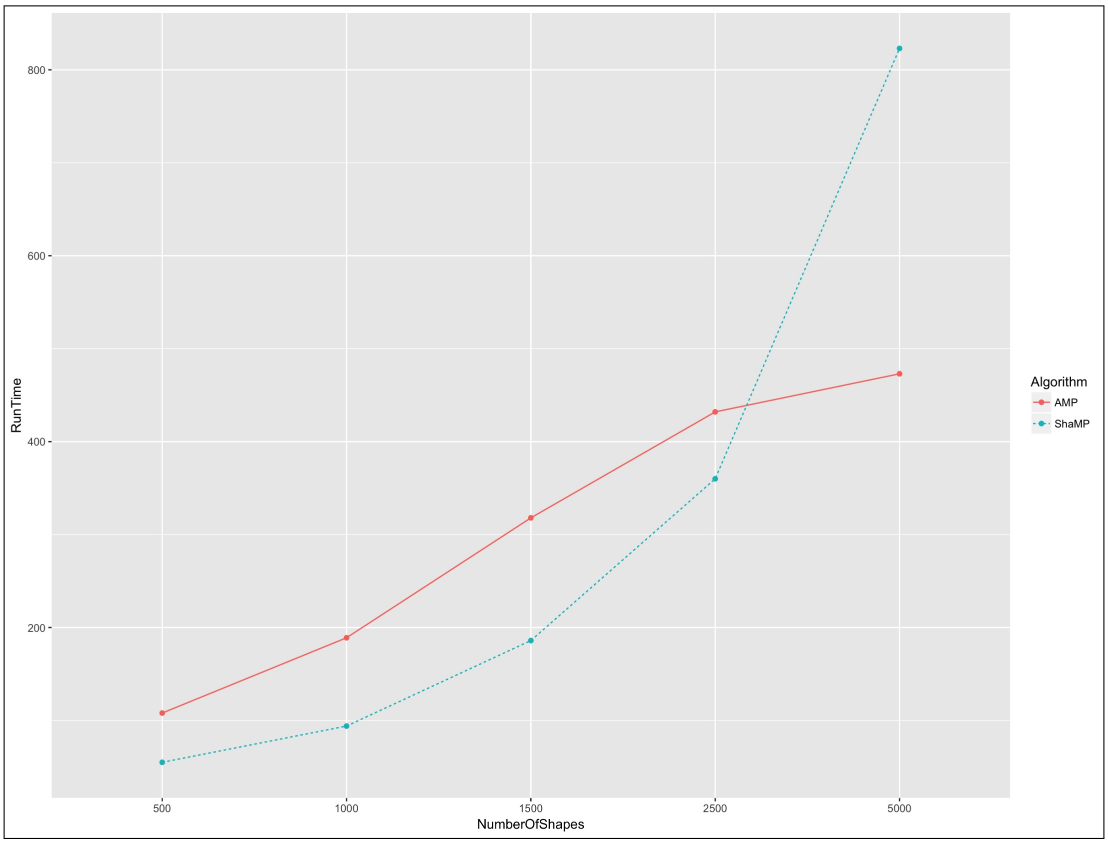


FIGURE 4.8: Runtime versus numbers of shapes ($\sigma = 2$).

4.3.3 Effect of Number of FETs

The third objective of the experimentation reported on here was to determine the effect of the size of the FET datasets considered on the operation of the AMP and ShaMP algorithms in terms of runtime. Two sets of experiments were conducted using artificial datasets. The first considered the effect of increasing the size of the dataset under consideration in terms of the number of FETs in the dataset whilst maintaining A_V and A_E at a constant size (and the corresponding numbers of attribute values). The second considered the effect of increasing the size of A_V and A_E whilst maintaining the number of FETs at a constant value. In both cases, values of σ equating to 0.5 and 1.0 were used.

For the first set of experiments ten artificial datasets, featuring a range of number of FETs from $1M$ to $10M$, increasing in steps of $1M$, were generated.

For each dataset the same attribute sets, and corresponding attribute values were used, $|A_V| = 2$ and $|A_E| = 5$. The AMP and ShaMP algorithms were applied to these datasets using $\sigma = 1.0$ and $\sigma = 0.5$. The results are shown in Tables 4.4 and 4.5. The first table gives the runtime results (seconds) with respect to the AMP algorithm, and the second with respect to the ShaMP algorithm.

Table 4.4: Recorded Runtimes (seconds) using artificial datasets featuring increasing numbers of FETs and the AMP algorithm

Num FETS in Data Set	$\sigma = 1.0$ Runtime (secs)	$\sigma = 0.5$ Runtime (secs)
1 M	349	2255
2 M	623	4318
3 M	OutOfMemoryError	OutOfMemoryError
4 M	OutOfMemoryError	OutOfMemoryError
5 M	OutOfMemoryError	OutOfMemoryError

Table 4.5: Recorded Runtimes (seconds) using artificial datasets featuring increasing numbers of FETs and the ShaMP algorithm

Num FETS in Data Set	$\sigma = 1.0$ Runtime (secs)	$\sigma = 0.5$ Runtime (secs)
1 M	77	77
2 M	149	151
3 M	153	240
4 M	247	315
5 M	387	374
6 M	474	468
7 M	551	559
8 M	607	952
9 M	1150	954
10 M	1102	1236
15 M	1968	2245
20 M	OutOfMemoryError	OutOfMemoryError

Inspection of the tables clearly indicates that, as was to be expected, the AMP algorithm was not able to handle a dataset when the number of FETs increased beyond 2 million FET; the reason for this was because the AMP algorithm operated by generating a large number of candidates that required a significant amount of memory resource not typically available on a single machine. However, the ShaMP algorithm was able to extract movement patterns from artificial datasets with up to 15M FET. It was also shown that the ShaMP runtime was relatively stable using two different σ values. However, when the number of FETs reached the 20 million, the ShaMP algorithm, operating on a single machine, was no longer able to cope. How this issue can be addressed is considered in the next chapter.

4.4 Summary

In this chapter two algorithms, AMP and ShaMP, have been proposed for identifying Movement Pattens (MPs) in network data. The first was founded on a traditional approach to MP mining, while the second adopted the novel approach of considering the "shape" of the MPs to be identified. In other words, the second approach used knowledge of the nature of MPs to realise anticipated efficiency gains. From the reported evaluation it was concluded that both algorithms could successfully identify MPs in large networks, however, the ShaMP algorithm was considerably more efficient at low support threshold (σ) values, provided that the number of shapes was not excessive. The evaluation established that the cut-off point, where the number of shapes was such that it was no longer beneficial to use the ShaMP algorithm, was at $2,600$ shapes (assuming binary valued attributes, the worst case scenario). However, it was conjectured that the real strength of the ShaMP algorithm was that it readily leant itself to parallelisation. This is therefore explored in further detail in the next chapter where three different parallelisation approaches are considered with respect to the ShaMP algorithm. The primary objective of the work presented in this chapter was to provide an answer to the second subsidiary question presented in section 1.3, namely "what are the most appropriate mechanisms whereby movement patterns can be mined (learnt/extracted) from network data". The main finding of this chapter was that the ShaMP algorithm was more efficient in term of the runtime, comparing to the AMP algorithm.

5 Movement Pattern Mining Using Big Data Facilities

The main objective of this thesis, as noted earlier in Chapter 1, was to develop learning mechanisms that are able to extract the behaviour of traffic in large networks. Large in this context was defined as networks that could not easily be processed on a single machine. The aim was then to use the extracted behaviours (Movement Patterns) to predict the nature of traffic movements in the context of networks that were either similar to those from which the patterns were extracted; or, in the case of time-variant networks, future (or past) renditions of the network. As evidenced in Chapter 4, when using a single machine, the proposed ShaMP algorithm was able to extract movement patterns (traffic behaviour) from networks more efficiently than the comparator AMP algorithm, if and only if there was enough memory on the single machine for the given network, otherwise the ShaMP algorithm failed. The work reported on in this chapter was directed at addressing this issue using "big data" mechanisms of various kinds. More specifically, the work presented in this chapter focuses on the third subsidiary question (see Section 1.3): "given mechanisms for mining movement patterns from network data how can these mechanism be scaled up to address movement pattern mining from very large networks" in other words the network presented in this chapter is concerned with the most appropriate mechanisms whereby the mechanisms reported on in earlier chapters, for mining movement patterns from network data, could be scaled up to address movement pattern mining in very large networks (networks that cannot be processed on a single machine).

The particular "big data" mechanisms considered in this chapter is ***Parallelisation***. The idea is to parallelise the operation of the ShaMP algorithm across a number of processes. There has been extensive published research directed at discovering frequent patterns in large datasets using the distributing computing concept of various kinds [2, 5, 11, 23, 39, 70, 88, 100, 102, 103]; however, most of the proposed distributed solution were based on the Apriori concept. The nature of the "shapes" used in the ShaMP algorithm means that each individual shape can be processed independently from other shapes. The ShaMP algorithm is therefore well suited to distribution/parallelisation; this was conjectured to be the main advantage of the ShapMP algorithm. On another hand, the AMP algorithm, which operates level by level in a top-down manner and, when parallelised, will feature significant message passing after each level, makes the AMP algorithm entirely unsuited to distribution/parallelisation.

Three distinct big data mechanisms are considered in this chapter: (i) Shared-Memory, (ii) Distributed-Memory and (iii) Hadoop/MapReduce. The Shared-Memory approach assumes, as the name implies, that tasks will be executed in parallel using n processes and that the available memory is shared between them. For evaluation purposes Omp4j[1] was used to produce a shared-memory variation of the ShaMP algorithm; Omp4j is an open-source OpenMP preprocessor for Java. The Distributed-Memory approach assumes that tasks will be executed in parallel by a number of processes each of which will have its own private memory; there is no shared memory. As such the processes need to communicate interim results between each other by message passing. For the evaluation presented in this

[1]http://www.omp4j.org/home

chapter MPJ Express[2] was used to produce a distributed-memory variation of ShaMP. MPJ Express is an open source Java message passing library that allows application developers to write and execute parallel applications for computer clusters. The third approach for processing large networks using ShaMP adopted Hadoop/MapReduce. Formally speaking, Hadoop is an open source framework for writing and running distributed user applications with respect to large amounts of data [52]. Unlike in the case of the previous two approaches (the Shared- and Distributed-Memory approaches), Hadoop adopts a philosophy to processing large datasets by "moving" the code to the data, rather than the other way round. In summary, the chapter presents three proposed "big data" variations of the ShaMP algorithm presented in the previous chapter: (i) The ShaMP Shared Memory System (ShaMP SMS), (ii) ShaMP Distribute Memory System (ShaMP DMS) and (iii) ShaMP Hadoop/MapReduce System (ShaMP HMS).

The remainder of this chapter is organised as follows. The proposed three "big data" approaches are discussed in Sections 5.1, 5.2 and 5.3 respectively. The performance of the three approaches was evaluated and compared using a set of experiments directed at a number of large networks. The results of this evaluation are presented and discussed in Section 5.4. Finally, the chapter is concluded with a summary and some conclusions in Section 5.5.

5.1 The ShaMP Shared Memory System

This section discusses the implementation of the ShaMP algorithm in the context of the Shared-Memory "big data" mechanism. The approach is referred to as ShaMP SMS (Shared Memory System). ShaMP SMS was realised using the Omp4j implementation of OpenMP in java. The main principle of the share-memory

[2]http://mpj-express.org

approach is that a single memory space is shared by N processes each of which has read/write access to it. The nature of the "shapes" featured in the ShaMP algorithm makes the algorithm well suited to being implemented in the context of SMS. The general principle behind ShaMP SMS was that there will be an administrator node responsible for generating all possible shapes and splitting them into N subsets according to the number of worker nodes available. To this end, all worker nodes share the same memory space to which a given large network D is allocated. As a consequence, the workers can operate in parallel to extract the desired Movement Patterns. The idea is illustrated in the Figure 5.1 which gives an example of ShaMP SMS where four worker nodes share the same memory space.

Algorithm 7: ShaMP SMS Algorithm

```
   Input:
1  $D$ = Binary valued input data set
2  $\sigma$ = Support threshold
3  $A_V$ = Set of vertex attributes
4  $A_E$ = Set of Edges attributes
   Output:
5    $M$ = Set of frequently occurring MPs $\{\langle MP_1, count_1\rangle, \langle MP_2, count_2\rangle, \dots\}$
6  In a given Mater node do:
7  $ShapeSet$ = Generate shapes set $\{shape_1, \dots, shape_n\}$ from the sets ($A_V$, $A_E$)
     (See Algorithm 6 in Chapter 4)
8  M = $\emptyset$
9  $N$ = Number of worker nodes in a cluster
10 Split and Distribute $ShapeSet$ across all worker nodes $P_{1,2,..,N}$
11 $ShapeSet = SubShapeSet_1 \sqcup SubShapeSet_2 \sqcup \dots \sqcup SubShapeSet_N$
12 $SubShapeSet = \{SubShapeSet_1, SubShapeSet_2, \dots, SubShapeSet_N\}$
13 In parallel and for all worker nodes do:
14 forall $P_{1,2,...N}$ work in parallel & on a same memory space do
15     forall $shape_i \in SubShapeSet$ do
16         forall $r_i \in D$ do
17             if $r_i$ matches $shape_i$ then
18
19             $MP_k$
20                 = MP extracted from $r_i$ if $MP_k$ in $M$ then increment support
21             else $M = M \bigcup \langle MP_k, 1\rangle$
22 forall $MP_i \in M$ do
23     if count for $MP_i < \sigma$ then remove $MP_i$ from $M$
```

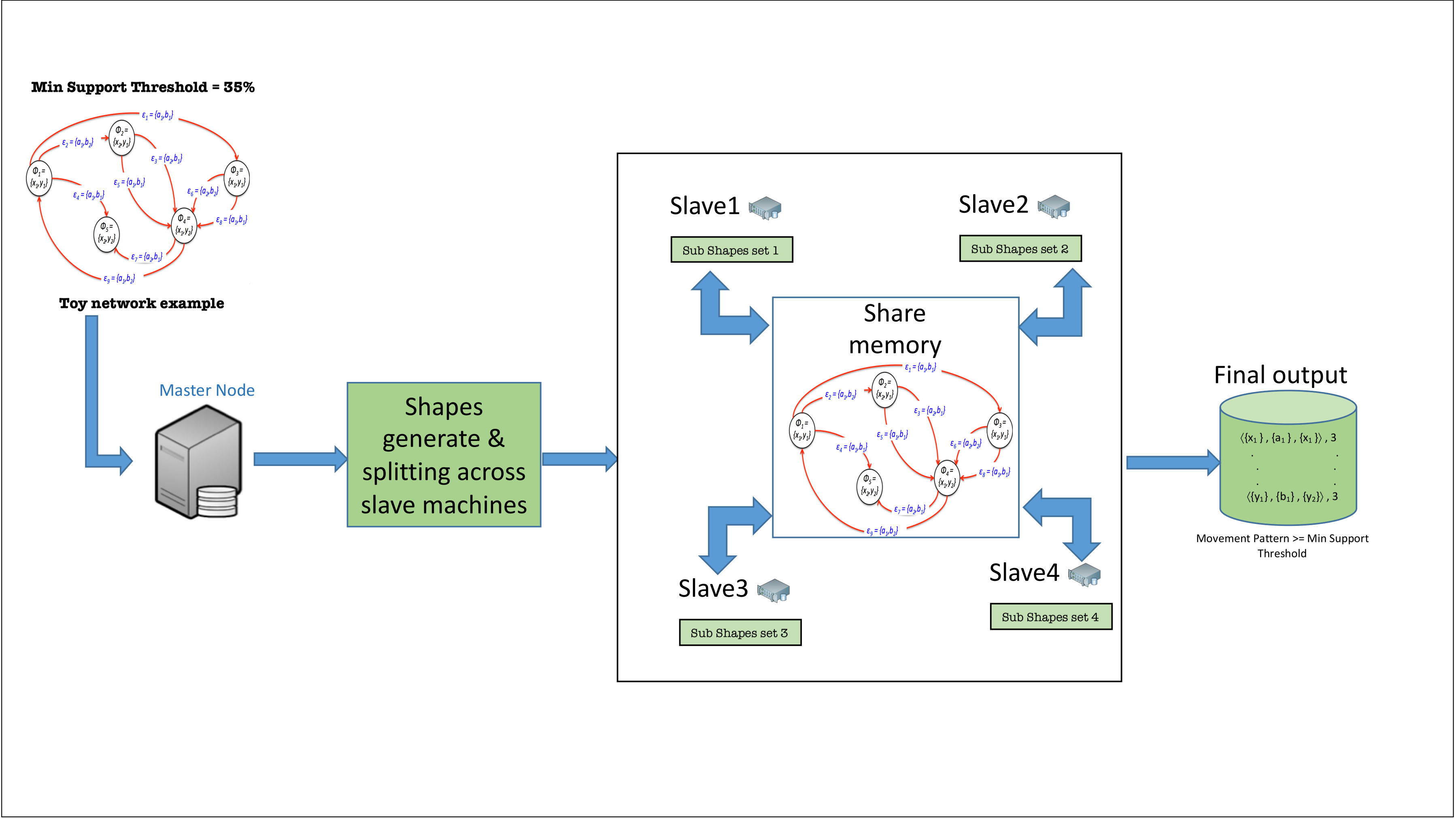


FIGURE 5.1: Working example illustrating the operation of ShaMP DMS as implemented using MPJ express ($\sigma = 35\%$).

The operation of ShaMP SMS is given by the pseudocode presented in Algorithm 7. The input to the algorithm is: (i) a Binary valued FET dataset D; (ii) a desired support threshold σ; (iii) a set of vertex attributes A_V; (iv) a set of edge attributes A_E; and (v) the number of processes to be run in parallel, P_N. The output is set of frequently occurring MPs stored in a set M, together with their support values, each represented as a tuple of the form $\langle MP_i, count_i \rangle$. The algorithm commences with the administrator node (line 7) which first generates the complete set of shapes, *ShapeSet*, by calling Algorithm 6, and then setting M to the empty set (line 8). The administrator node then divides the *ShapeSet* equally into N subsets according to the number of worker nodes available (line 10). Note that all worker nodes work in parallel and they share the same memory space where D is stored. For each worker node, the algorithm loops through the dataset D and compare each record $r_j \in D$ with the current shape $shape_i$. A record r_j matches a $shape_i$ if all the attributes featured in $shape_i$ are also featured in r_j (r_j may include additional attributes not featured in $shape_i$). Where a match is found the relevant attribute values in r_i form an MP. If the identified MP is already contained in M we simply update the support value, otherwise, we add the newly discovered MP to M with a support value of 1. Once all shapes have been processed by the worker nodes, the administrator node will loop through M and remove all MPs whose support count is less than σ.

5.2 The ShaMP Distributed Memory System

This section discusses the distributed memory implementation of the ShaMP algorithm, ShaMP Distributed-Memory System (ShaMP DMS), a parallel approach for extracting Movement Patterns from FET data. ShaMP DMS was realised using the MPJ Express java implementation of the Message Passing Interface (MPI). The main principle of any DMS is that each worker node has its own privet memory space, thus “message passing” is the only way that worker nodes can communicate with one another. This is often seen as a disadvantage of DMS in the context of algorithms which feature a high message passing overhead. However, as noted previously, the advantage offered by the ShaMP algorithm, with respect to the alternative AMP algorithm, is that there is no requirement for any inter-worker communication. The basic ShaMP algorithm was modified so as the benefit from the DMS principle. The general idea was that a single administrator node would control N worker nodes who in turn would work in parallel, but using their private memory space, to extract the desired Movement Patterns. The Administrator node would be responsible for generating a set of shapes that feature in a given FET dataset D representing a network G, and then distribute the shapes across the N worker nodes by splitting the generated shape set into N subsets, one for each worker node. The process is illustrated with a worked example in Figure 5.2. An obvious disadvantage of ShaMP DMS is the large amount of memory, compared to ShaMP SMS, that needs to be available to each worker node so that a full copy of D can be stored at each worker node.

The operation of ShaMP DMS is similar to that of ShpMP SMS, and is given by the pseudocode presented in Algorithm 8 (ShaMP MPJ). The input, as in the case of ShaMP SMS, is: a Binary valued FET dataset D, a desired support threshold σ, a set of vertex attributes A_L, set of Edges attributes A_E and the number of processes to be run in parallel P_N. The output, as before, is a set of frequently occurring MPs stored in a set M, together with their support values, each represented as a tuple of the form $\langle MP_i, count_i \rangle$. The algorithm commences with the administrator node (line 7) first generating the complete set of shapes, *ShapeSet*, by calling Algorithm 6, and then setting M to the empty set (line 8). The administrator node then divides the *ShapeSet* equally into N subsets according to the number of worker nodes available (line 10). Note that all worker nodes work in parallel and they have their private memory space; this is the main

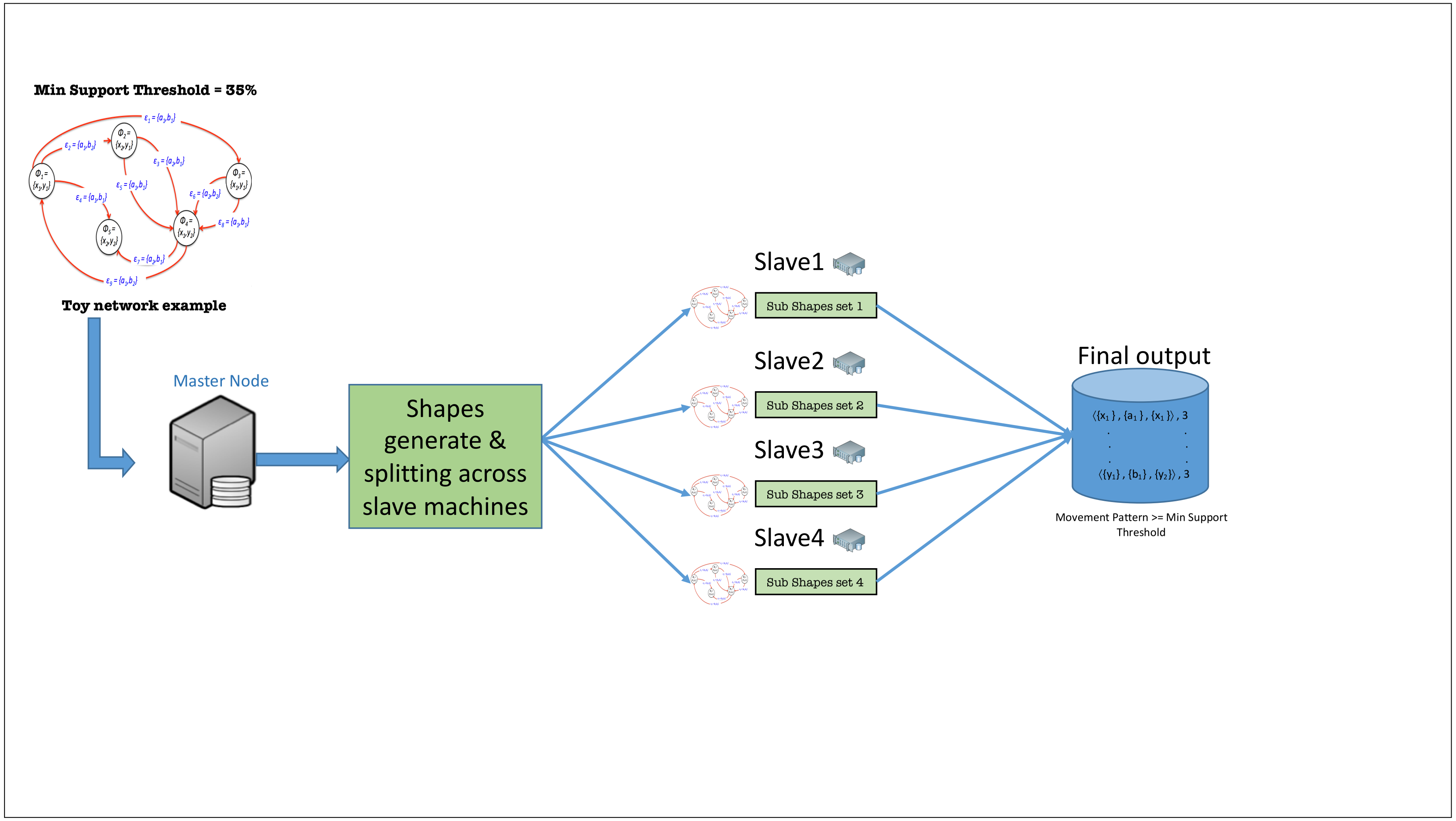


FIGURE 5.2: Worked example illustrating the operation ShaMP DMS as implemented using MPJ express ($\sigma = 35\%$).

difference between ShaMP SMS and ShaMP DMS. The parallelization commences at (line 14) where all P_N work in parallel. For each shape the algorithm loops through the dataset D and compare each record $r_j \in D$ with the current shape, $shape_i$. A record r_j matches a $shape_i$ if all the attributes featured in $shape_i$ are also featured in r_j (r_j may include additional attributes not featured in $shape_i$). Where a match is found the relevant attribute values in r_i form an MP. As in the case of the ShaMP SMS algorithm, if the identified MP is already contained in M we simply update the support value, otherwise we add the newly discovered MP to M with a support value of 1. Once all shapes have been processed by the worker nodes the administrator node loops through M and remove all MPs whose support count is less than σ.

Algorithm 8: ShaMP DMS Algorithm

```
   Input:
1  $D$ = Binary valued input data set
2  $\sigma$ = Support threshold
3  $A_V$ = Set of vertex attributes
4  $A_E$ = Set of Edges attributes
   Output:
5    $M$ = Set of frequently occurring MPs $\{\langle MP_1, count_1\rangle, \langle MP_2, count_2\rangle, \dots\}$
6  In a given Mater node do:
7  $ShapeSet$ = Generate shapes set $\{shape_1, \dots, shape_n\}$ from the sets ($A_V$, $A_E$)
     (See Algorithm 6 in Chapter 4)
8  M = $\emptyset$
9  $N$ = Number of woker nodes in a cluster
10 Split and Distribute $ShapeSet$ across all worker nodes $P_{1,2,\dots,N}$:
11 $ShapeSet = SubShapeSet_1 \sqcup SubShapeSet_2 \sqcup \dots \sqcup SubShapeSet_N$
12 $SubShapeSet = \{SubShapeSet_1, SubShapeSet_2, \dots, SubShapeSet_N\}$
13 In parallel and for all worker nodes do:
14 $\forall$ $P_N$ has $SubShapeSet_i$ and $D$
15 forall $shape_i \in SubShapeSet$ do
16     forall $r_i \in D$ do
17         if $r_i$ matches $shape_i$ then
18
19         $MP_k$
20             = MP extracted from $r_i$ if $MP_k$ in $M$ then increment support
21         else $M = M \bigcup \langle MP_k, 1\rangle$
22 forall $MP_i \in M$ do
23     if count for $MP_i < \sigma$ then remove $MP_i$ from $M$
```

5.3 The ShaMP Hadoop/MapReduce System

This section presents the proposed coupling ShaMP system with Hadoop and MapReduce (ShaMP HMS). The utilisation of the Hadoop framework for any distributed computing process is not straightforward; there are number of stages that need to be addressed. With respect to the ShaMP algorithm, movement patterns can be extracted from a large network, using Hadoop and MapReduce, using a five stage process: (i) uploading of the given data set D to the Hadoop Distributed File System (HDFS), (ii) shape generation, (iii) map function execution, (iv) shuffle and (v) reduce function execution. Each of these stages is discussed in further detail in the remainder of this section. A worked example illustrating the operation of ShaMP MRS, with $\sigma = 35\%$, is given in Figures 5.3.

The first ShaMP HMS stage is the loading of a given FET dataset D into the Hadoop Distributed File System (HDFS)[3]. For the Hadoop framework, HDFS is written in Java. The input data should be uploaded into the same directory where Hadoop is located so that it can be used by MapReduce nodes. After the given data has been uploaded successfully HDFS will divide/split it into small chunks with a typical size of several dozen megabytes (i.e 46MB or 128MB) and store them at DataNodes as indicated in Figure 5.3 by the three blue cylinders; ***DataNodes*** represent worker nodes in a computer cluster, each with its own storage, they are controlled by a single administrator node called the ***NameNode***.

The second stage is shape generation, indicated in Figure 5.3 by the green rectangle. In this stage, all possible shapes are generating in a similar manner to that described for ShaMP SMS and ShaMP DMP above. Every DataNode generates the same set of shpes to make sure each individual DataNode can work independently without a need for shape transfer between DataNodes.

[3]https://hadoop.apache.org/

Like any other MapReduce programme, two functions need to be defined, the ***Map*** and ***Redcue*** functions. The third stage is then the execution of the given map function whereby the identified data chunks from stage two are processed. The input for this stage is a defined set of shapes. The mapping task, conducted at each DataNode, is done by looping through the given shapes and for each shape will loop through the dataset D to extract an intermediate set of movement patterns S which is temporarily held in the form of a set of $\langle Key, Value\rangle$ pairs; where in this case Key is an MP and $Value$ is the occurrence count for the MP which at this stage will equate to 1. This stage is shown by orange rectangles in Figure 5.3.

The fourth stage is the shuffle stage where Hadoop groups together all intermediate $\langle Key, Value\rangle$ pairs produced by the Map function which have the same key so as to produce a list of n values for each key $\{Key, value_1, value_2, \dots\}$ as indicated by blue rectangles in Figure 5.3.

The final stage is the execution of the reduce function. This is the function responsible for outputting the result to the HDFS. This stage is indicated by the blue cylinder on the right-hand side of Figure 5.3.

The operation of ShaMP HMS is given by the pseudocode presented in Algorithm 9 (ShaMP Hadoop). Note that all of the above five stages are executed in parallel at each DataNode. The algorithm comprises both Map and Reduce functions. The input for the Map function is again a network G represented in terms of a FET dataset D, and a desired support threshold σ; while the output is a set S comprised of a set of tuples of the form $\langle Key, Value\rangle$ where, as already noted, Key is an MP and $Value$ is the corresponding occurrence count value. The input to the Reduce function is the set S, whilst the output is the desired set of MPs, M. The Map task commences by generating the available set of shapes. For each shape, the algorithm loops through D comparing each record $r_j \in D$ with

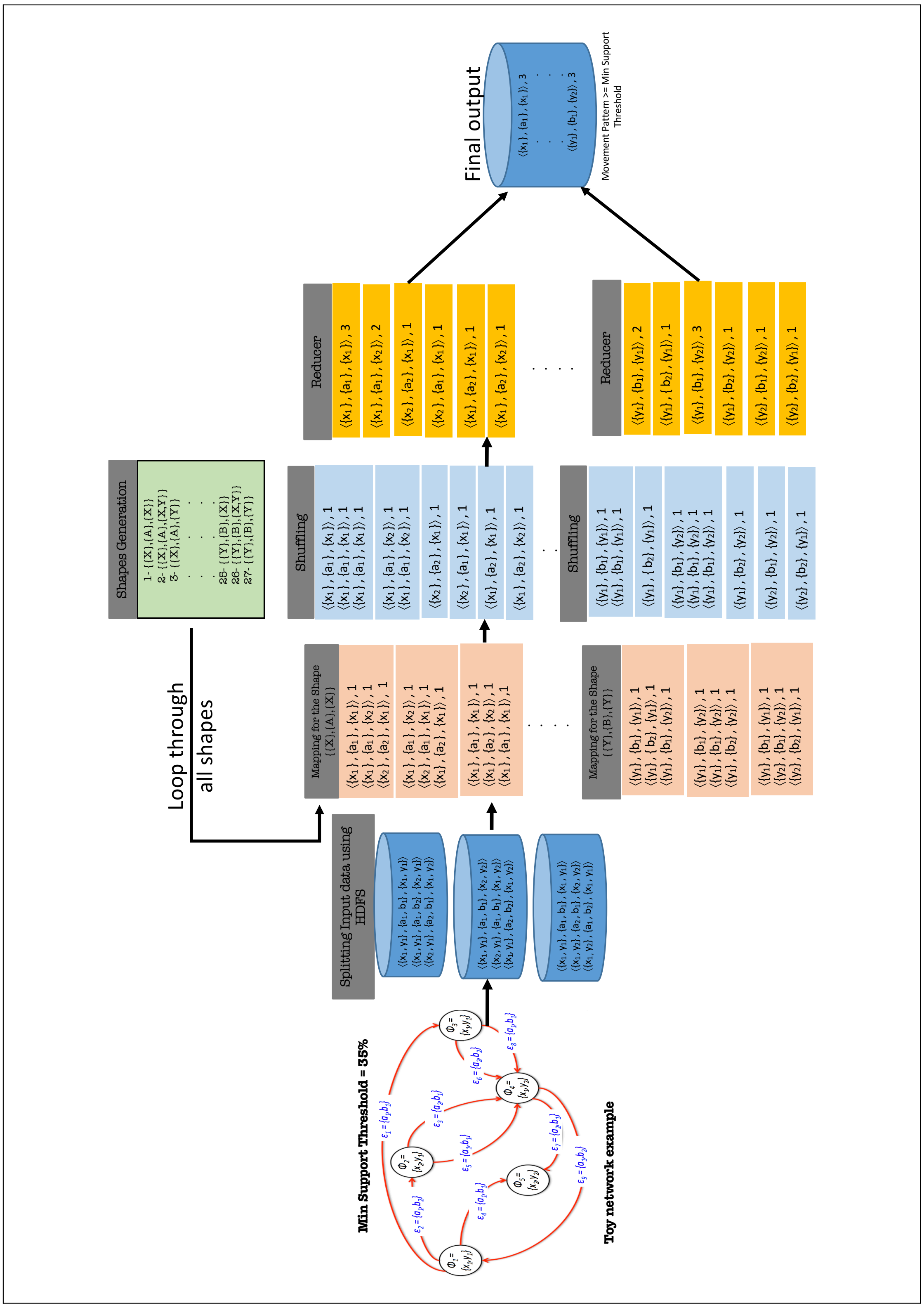

Min Support Threshold = 35%
Toy network example
Splitting Input data using HDFS
Loop through all shapes
Shapes Generation
Mapping for the Shape {[X],(A),[X]}
Mapping for the Shape {[Y],[B],[Y]}
Shuffling
Reducer
Final output
Movement Pattern >= Min Support Threshold

FIGURE 5.3: Working example illustrating the operation of the ShaMP HMS implemented on the top of Hadoop with ($\sigma = 35\%$).

the current shape $shape_i$, where a match is found, a key-value $\langle MP, 1\rangle$ is created. The Reduce function aggregates all the MPs with the same key (MP) so as to produce frequency counts for each key (MP). After that, the set S is processed and any MPs that have a frequency count greater than σ are appended to the set M.

Algorithm 9: ShaMP Haddop Algorithm

```
   Input:
1  $D$ = Binary valued input data set
2  $\sigma$ = Support threshold
3  $A_V$ = Set of vertex attributes
4  $A_E$ = Set of Edges attributes
   Output:
5    $M$ = Set of frequently occurring MPs $\{\langle MP_1, count_1\rangle, \langle MP_2, count_2\rangle, \dots\}$
6  Start:
7  $ShapeSet$ = Generate shapes set $\{shape_1, \dots, shape_n\}$ from the sets ($A_V$,
     $A_E$) (See Algorithm 6 in Chapter 4)
8  Map Function:
9  Begin:
10 S = $\emptyset$
11 $ShapeSet$ = The set of possible shapes
12       $\{shape_1, shape_2, \dots\}$
13 forall $shape_i \in ShapeSet$ do
14     forall $r_i \in D$ do
15         $MP_k$ = MP extracted from $r_i$
16         $MP_k$SupportCount = Count ($MP_k$,1)
17 $S = S \bigcup \langle MP_k, 1\rangle$
18
19 Reduce Function:
20 Begin:
   Input:
21   $S$ set from Map Function
   Output:
22   $M$ = Set of frequently occurring MPs $M$ =
23       $\{\langle MP_1, count_1\rangle, \langle MP_2, count_2\rangle, \dots\}$
24 forall $\langle MPs, 1\rangle$ $pairs \in S$ do
25     $\langle MPs, count\rangle$ = $Aggregation$ all $\langle MPs, counter\rangle$
26             pairs that have same $MP$
27     if $count\ for\ MP > \sigma$ then $M = M \cup \langle MPs, count\rangle$
```

A perceived disadvantage of the Hadoop framework, in the context of ShaMP, is that it requires significant interchange with the HDFS. It was conjectured that this would slow down the operation of ShaMP HMS compared with ShaMP SMS and SgaMP DMS.

5.4 Experiments and Evaluation

This section reports on the experiments conducted to analyse the operation of the proposed ShaMP parallel and distributed variations considered in this chapter: (i) ShaMP SMS, (ii) ShaMP DMS and (iii) ShaMP HMS. The evaluation was conducted using two categories of network data: (i) networks extracted from the CTS database introduced in Section 3.2 and (ii) artificial networks, as introduced in Section 3.4. The first was used to investigate the operation of the variations with respect to real data, and the second to investigate the operation of the variations under changing parametric conditions (the parameters can be easily controlled when using artificial data). The objective of the evaluation was to compare the operation of the ShaMP algorithm, using three different parallel variations, so as to determined which variation was most suitable with respect to large networks; the metric used was run time. In the context of the experiments conducted to determine the effect of distributing the ShaMP algorithm the desired parallelisation was achieved using a 4 Xeon Phi node cluster where each node had 40 cores. In the remainder of this section the experiments with regard to the CTS dataset are reported on in Sub-section 5.4.1, while those with respect to the artificial datasets are reported on in Sub-section 5.4.2.

5.4.1 Evaluation Results Using CTS Data

This section presents the results obtained with respect to experiments conducted to compare the efficiency of the operation of the three proposed distribution ShaMP variations. For this purpose the four CTS network datasets used with respect to

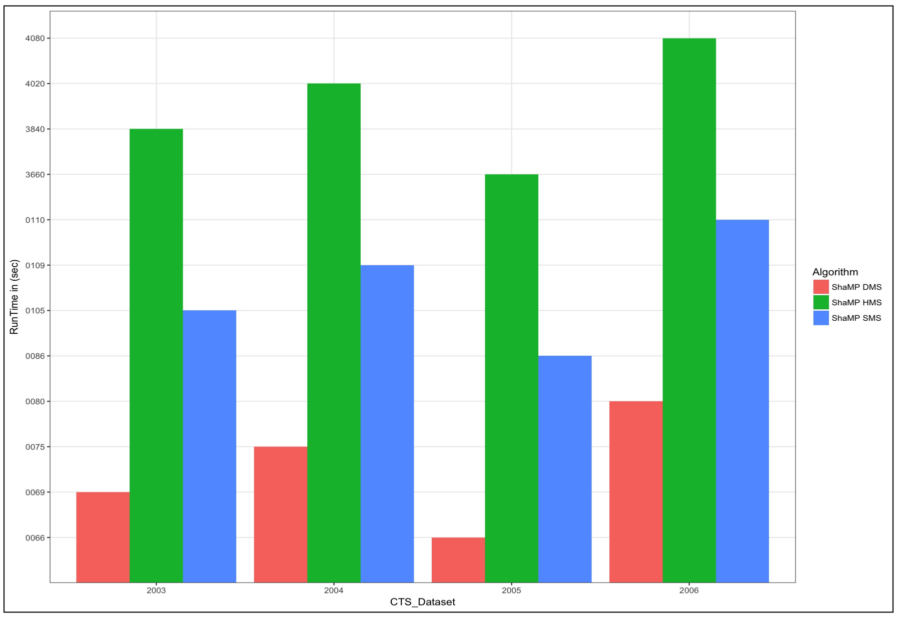


Figure 5.4: Runtime comparison between ShaMP SMS, ShaMP DMS and ShaMP HMS using CTS network data.

the evaluation presented in chapter 4 "were again used": 2003, 2004, 2005, and 2006. The efficiency results obtained are presented in Figure 5.4. From the figure, it can clearly be seen that the runtime for ShaMP DMS is much faster than that for ShaMP SMS and ShaMP HMS. The reason for this was because the nature of the MPI, implement using MPJ Express in the case of the evaluation of ShaMP DMS, was more beneficial when the amount of messages passing between processes is not too large (a number of $FETs$ of less than 3 million). With respect to ShaMP HMS, usage of the Hadoop framework involved many saves to the Hadoop Distributed File System (HDFS), this made the Hadoop framework slower in comparison to ShaMP SMS and ShaMP DMS.

5.4.2 Evaluation Results Using Artificial Data

It was clear that, whatever ShaMP variation used, as the number of edges increased, the runtimes for ShaMP SMS, ShaMP DMS, and ShaMP HMS would also increase. However, there would be a point where one of these algorithms could

Table 5.1: Runtime comparison between ShaMP SMS, ShaMP DMS and ShaMP HMS using artificial data sets.

#	RunTime		
Dataset Size in million	Using DMS	Using SMS	Using Hadoop
5	5	5	148
10	10	10	320
15	15	15	370
20	22	20	505
25	30	26	615
30	35	30	Unable to process
35	43	36	Unable to process
40	48	43	Unable to process
45	56	51	Unable to process
50	66	64	Unable to process

no longer process the number of edges to be considered. To determine where this point might be a sequence of five artificial datasets was generated that featured increasing numbers of edges ranging from $5,000,000$ to $50,000,000$ in steps of $5,000,000$. In addition, for the artificial datasets $|A_v| = 2$ and $|A_E| = 5$ were used because these were the values featured in the CTS datasets. For the experiments, $\sigma = 1.0$ was used. Table 5.1 presents the results obtained using the artificial data sets. From the Table, it can be seen that there are no significant differences in operation between ShaMP DMS and ShaMP SMS with respect to the artificial data sets. However, as the number of edges increased, the ShaMP HMS algorithm was eventually no longer able to process the data as the extent of the message passing between machines became restrictive. This occurred when the numbers of edges were more than 25 million. The reason for this relates to the Hadoop Distributed File System (HDFS) and the MapReduce process. Hadoop splits files into blocks and distributes them across nodes in a cluster and when the size of the dataset becomes large some machines will no longer have enough disk space to store the temporary files generated. However, with respect to ShaMp DMS and ShaMP SMS, the two algorithms were able to process data in an efficient manner even when 50 million edges were considered (less than $\sim$ 65 minutes). To give a better feel for the ShaMP SMS and ShaMP DMS runtimes these are presented in the form of a plot in Figure 5.5.

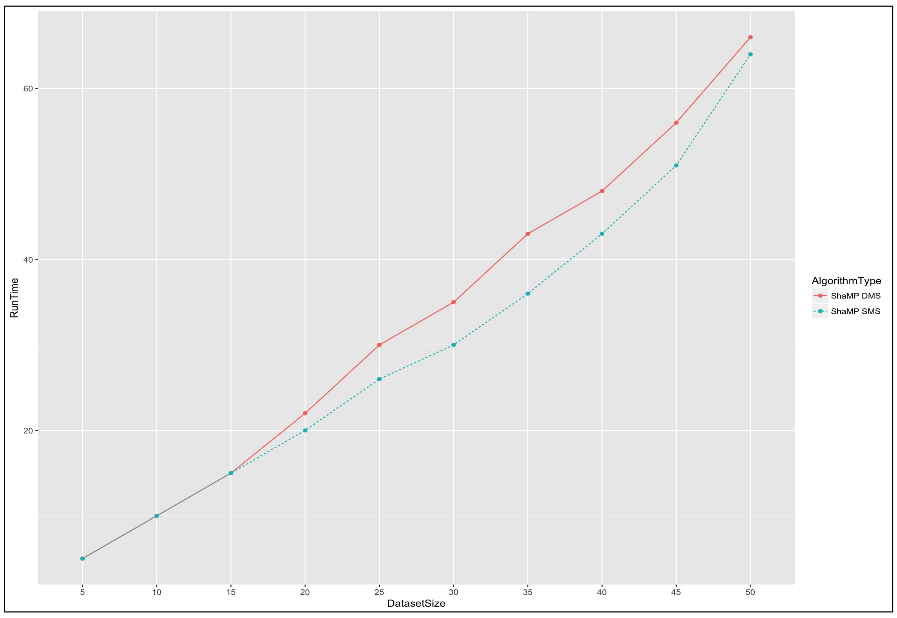


FIGURE 5.5: Runtime (minutes) comparison between ShaMP DMS and ShaMP SMS using artificial data sets.

5.5 Summary and Conclusions

In this chapter, a number of variations of the ShaMP algorithm from the previous chapter have been proposed that incorporate various "big data" mechanisms so as to be able to apply the MP mining concept to large datasets. Three ShaMP variations were proposed: The ShaMP Shared Memory System (ShaMP SMS), the ShaMP Distribute Memory System (ShaMP DMS) and the ShaMP Hadoop MapReduce System (ShaMP HMS). The first was founded on the Shared Memory System (SMS) and implemented using Omp4j. The second was founded on the Distributed Memory System (DMS) and implemented using MPJ Express. The third was founded on Hadoop/MapReduce. From the reported evaluation designed to compare the efficiency of the proposed variations it was concluded that both ShaMP SMS and ShaMP DMD could successfully be applied in reasonable runtime; however, ShaMP HMS was found to be slow and was unable to process large networks (25 million edges and above). The primary objective of the work presented in this chapter was to provide an answer to the third subsidiary question presented in section 1.3, namely "given mechanisms for mining movement patterns from network data how can these mechanisms be scaled up to address movement pattern mining from very large networks". The work and evaluation presented in this chapter have established that ShaMP algorithm is well suitable to handle very large networks by utility big data facilities, like Share/Distribute Memory Systems and Hadoop/MapReduce, because of the nature the shapes concept used that makes the algorithm well suitable to be implemented in a parallized manner. The next chapter will focus on how to utilize the extracted traffic behaviour (Movement Patterns) to predict the nature of traffic movement.

6 The Application Of Movement Patterns

The fourth subsidiary research question (see Section) 1.3 was "*Once we have a collection of movement patterns how can they best be applied to previously unseen network data and how do we know whether the manner in which movement patterns are applied produces the correct results?*. The work presented in this chapter was directed at providing an answer to this subsidiary research question. The fundamental idea of the Movement Pattern (MP) concept, the main motivation, was that they can be used to predict traffic movement. This prediction can be considered in two contexts:

1. **The Global Context**: The prediction of the nature of traffic movement (the network connectivity) in the context of entire networks that were either similar to those from which the MPs were extracted; or, in the case of time-variant networks, future (or past) renditions of the network.
2. **The Local Context**: The prediction of the nature of traffic movement (connectivity) between a newly introduced network node and the remainder of a given network.

The first scenario was considered in the context of the CTS network, thus predicting cattle movement in future or related networks. The idea was to investigate a mechanism that might be used to provide answers to questions of the form:

"given the nature and volume of 'traffic' in a given network N at time t what will the nature and volume of traffic be at time $t + \delta$?", or alternatively "given the nature and volume of 'traffic' in a given large network N what will be the nature and volume of traffic in a closely related network M?". Note that this form of prediction is distinct from other applications that involve movements such as trajectory or link prediction.

The second scenario was considered in the context of the DSN network; thus predicting the nature of the message flow with respect to a newly introduced node. Predictions of this form could then be used to make "dating" recommendations. The aim was to illustrate the idea that the MP concept also has application in the context of recommender systems, more specifically recommender systems embedded into Social Networks.

The remainder of this chapter is organised as follows. Mechanisms for achieving Global MP Prediction are discussed in Sections 6.1; the section also reports on the results of experiments directed at measuring the accuracy of the proposed mechanism with respect to the CTS network data. Mechanisms for Local MP prediction are then discussed in Section 6.2 with a focus on local MP prediction in the context of recommender systems. Finally, the chapter is concluded with a summary and some conclusions in Section 6.3.

6.1 Global Movement Pattern Prediction

The fundamental idea of Global MP prediction, as presented in this section, is to use frequently occurring MPs extracted from a given network, where the traffic is known, and apply these patterns to a related network where the traffic is unknown. In other words, to apply extracted MPs to a network work where we only have descriptions of the vertices and not the edges; it is the edges and their nature that we wish to predict. More formally the idea is to use the frequently occurring MPs that have been mined from a network $G = \{V, E\}$ to predict movement (traffic) in a previously unseen but related network $G' = \{V', E'$ where $E' = \emptyset$ (the previously unseen network is thus comprised solely of vertices).

In the case of the DSN application, we might wish to apply global MP prediction to establish what the traffic in the network might be in the future. Alternatively, we might wish to predict the traffic in a parallel DSN. In the case of the CTS network, we might wish to apply global MP prediction to determine what the traffic in the CTS will be in the future. Alternatively, we might wish to apply global MP prediction to determine the cattle movement in a related network; for example, a CTS set up in a neighboring country. The work presented in this section focuses on the later although the proposed mechanism could equally well be applied to the first.

The remainder of this section is divided into two subsections. Sub-section 6.1.1 presents the proposed global movement pattern prediction mechanism. The evaluation of this mechanism is then presented in Sub-section 6.1.2.

6.1.1 The Global Movement Pattern Prediction Mechanism

The basic challenge, given a previously unseen network comprised only of vertices, is to predict the edges that will exist. To do this, every pair of vertices in G' needs to be compared with the vertices in the collected set of movement patterns M. The matching can be conducted in various ways that can be categorised as either: (i) exact matching or (ii) partial matching. The intuition was that exact matching would be too restrictive and thus partial matching was adopted. More specifically, given two vertices $v_i \in MP_i$ (either the $From$ or To vertex) and $v_j \in V'$ a match exists if $v_i \subseteq v_j$, in other words v_j can include additional attribute values but must include the attribute values making up v_i. The novel mechanism proposed for populating E', using partial matching as described above, was to first predict the MPs that exist in G' and to then use these to populate E'.

The pseudocode for predicting the MPs that exist in G' is given in Algorithm 10. The algorithm takes as input a set of previously derived MPs M and a network $G' = \{V', E'\}$ where $E' = \emptyset$. Note that there is no requirement for any direct correspondence between V and V'. The output is a set of predicted MPs I; on startup $I = \emptyset$. The algorithm commences by looping through all MPs in M. For each MP the "from" and "to" vertex attribute-value sets ($From_i$ and To_i) are extracted. The algorithm then loops through V' to check if $From_i$ is a subset of any $v_j \in V'$. If so the algorithm loops through V' again to check if To_i is a subset of any $v_k \in V'$. If so the edge attribute-value set is extracted from MP_i and used to form the tuple $\langle From_i, Edge_i, To_i \rangle$ which is added to the set i so far (the set of predicted MPs). The set of predicted MPs is then used to populate E'.

6.1.2 The Accuracy Of Movement Pattern Prediction

The accuracy of the proposed prediction mechanism was tested using the CTS network training and test datasets (20003, 2004, 2005 and 2006). The idea was to compare the set of predicted MPs, M', with the set of MPs known to exist in the test data. More specifically the evaluation was conducted by applying the MPs extracted with respect to one year to the vertices of all other years and determining the frequent MPs that resulted (using the same σ value). In each case, the resulting set M' of predicted MPs was compared with the set of known "ground truth" MPs M_T.

As noted above, the performance measures used were Precision, Recall and

Algorithm 10: Prediction Mechanism

```
Input:
M = Set of frequently occurring MPs $M =$
      $\{\langle MP_1, count_1\rangle, \langle MP_2, count_2\rangle, \dots\}$
      describing trafic a network $G$
$G'$ = A previously unseen net work $G' = \{V', E'\}$
      where $E' = \emptyset$
Output:
$I$ = Set of predicted MPs in $G'$, $I =$
      $\{\langle MP'_1, count'_1\rangle, langleMP'_2, count'_2\rangle, \dots\}$
Start:
for all $MP_i \in M$ do
    $From_i$ = From attribute value set extracted
        from $MP_i$
    $To_i$ = To attribute value set extracted
        from $MP_i$
    for all $v_j \in V'$ do
        if $From_i \subseteq v_j$ then
            for all $v_k \in V'$ do
                if $To_i \subseteq v_k$ and $v_j \neq v_k$ then
                    $Edge_i$ = Edge attribute values
                        set extracted from $MP_i$
                    I = I $\cup \langle From_i, Edge_i, To_i\rangle$
```

the F1 measure [65]. Note that high precision relates to a low false positive rate, and high recall relates to a low false negative rate. High scores for precision and recall show that the predictor is performing well. The F1 measure combines both precision and recall and is thus the most significant measure. The results are presented in Tables 6.1, 6.2, 6.3 and 6.4. Each table shows the test results from using the MPs extracted from one year of data (2003, 2004, 2005 and 2006 respectively) to the remaining three years. Concentrating on the F1 measure, average values of 0.945, 0.951, 0.935 and 0.932 were generated, an overall average of 0.941, a very good result. The overall average values for precision and recall were 0.965 and 0.919 respectively, again a very good result.

In summary, from the main findings, it may be concluded that the proposed

Table 6.1: Accuracy of proposed prediction mechanism using MPs generated from the 2003 CTS data set.

Network from which MPs were extracted	G' network = 2003		
	Percision	**Recall**	**F1**
2004	0.978	0.967	0.973
2005	0.967	0.89	0.927
2006	0.961	0.929	0.945
average	0.961	0.929	0.945

Table 6.2: Accuracy of proposed prediction mechanism using MPs generated from the 2004 CTS data set.

Network from which MPs were extracted	G' network = 2004		
	Percision	**Recall**	**F1**
2003	0.971	0.971	0.971
2005	0.979	0.895	0.935
2006	0.946	0.948	0.947
average	0.965	0.938	0.951

Table 6.3: Accuracy of proposed prediction mechanism using MPs generated from the 2005 CTS data set.

Network from which MPs were extracted	G' network = 2005		
	Percision	**Recall**	**F1**
2003	0.95	0.917	0.933
2004	0.962	0.918	0.939
2006	0.954	0.913	0.933
average	0.955	0.916	0.935

Table 6.4: Accuracy of proposed prediction mechanism using MPs generated from the 2006 CTS data set.

Network from which MPs were extracted	G'network = 2006		
	Percision	**Recall**	**F1**
2003	0.957	0 .863	0.907
2004	0.968	0.889	0.927
2005	0.991	0.817	0.896
Average	0.979	0.892	0.932

prediction mechanism can be effectively used to accurately predict traffic (movement) in previously unseen networks.

6.2 Local Movement Pattern Prediction

This section discusses the usage of the concept of Movement Patterns (MPs) in the local context where a new node is added to a network and we wish to predict the likely traffic to and from this new node with respect to the rest of the network. Recall that the distinctions between global movement pattern prediction and local movement pattern prediction are:

1. For global movement pattern prediction $|G'| > 1$ while for local movement pattern prediction $|G'| = 1$.
2. For global movement pattern prediction we are interested in predicting the MPs within G', while for local movement pattern prediction we are interested in predicting the MPs that link G' to G.

In the case of the CTS network this would be the situation where a new holding area is added to the network and we wish to establish the cattle to and from movements between this new holding area and the rest of the network. In the case of the DSN application, this is where a new participant joins the network and we wish to predict the traffic to and from this new participant. As noted above, in the case of the DSN application, movement pattern prediction is equivalent to making "recommendations". Thus, in the remainder of this section, we consider local MP prediction in terms of recommendation.

There are two types of interaction between users of social networks: (i) reciprocal interaction and (ii) non-reciprocal interaction. The first is exemplified by the interaction between friends on facebook (the traffic is two-way), the second by followers on Twitter (the traffic is one way). In recommender systems the interaction is typically non-reciprocal, a recommendation is made to a user, the traffic is one-way. This is not the case in DSNs where the recommendation is two way, users are recommended to each other.

The fundamental motivation for recommendation system is to help overcome information overload. Recommender systems have also been found to provide a significant impact with respect to improving user satisfaction in online retail settings [82, 90]. Many recommendation algorithms and systems have been proposed, however, improving recommendation accuracy, data sparsity, and "cold-start" issues remain challenges for any form of automated recommendation. This is equally true in the context of DSNs which have become an impotent platform for people looking for potential partners online. According to a recent survey[1], conducted in the USA, more than 49 million single people (out of 54 million) have used DSNs such as eHarmony and Match.com. Moreover, according to the same survey, 20% of current committed relationships began online. In a large dating network finding potential partners is time consuming, therefore many DSNs give compatible partner suggestions; in the same manner as more general recommender systems, see for example [76]. A further challenge for DSN recommendation is that the recommender system must satisfy the preferences of pairs of users [73] as opposed to single users. The basic operation of DSNs, regardless of the adopted recommendation system used, is as follows.

1. ***Joining the network***. When a new user joins a DSN a new user profile is created using information provided by the new user; information such as: age, gender, location, job, education, income, smoking, drinking, hobbies, and so on.

[1]see http://www.statisticbrain.com/online-dating-statistics/

2. ***B*rowsing**. After the creation of the profile, the new user can browse the profiles of existing users (as can existing users).

3. ***O*ne sides match**. While browsing, users may send messages to other users.

4. ***R*eciprocal match**. On receipt of a message a user can return a message (reciprocate). Where this happens an edge is established in the DSN. The strength of an edge can be defined in terms of the quantity and/or duration of the messages. A degradation factor can also be applied to take into account the temporal nature of the network.

Given the large number of users, browsing is unlikely to be successful, hence DSN systems also provide recommendations. Recommendations can be made when a new user joins the network and periodically for existing users. The most commonly adopted techniques for automated recommendation generation are founded on some form of Collaborative Filtering [49, 86]. A recommendation system based on the concept of frequently occurring Movement Patterns (MPs), as proposed in this section, represents a novel and alternative approach to automated recommendation. More specifically the next section proposes the RecoMP system which uses the MP concept to provide a set of "recommendations". The remainder of this section is organised as follows. The proposed RecoMP algorithm is presented in Sub-section 6.2.1. To evaluate the proposed algorithm the idea was to compare its operation with that of a system that adopted the established Collaborative Filtering approach. To this end a second recommendation system was developed, the RecoCF system, which is presented in Sub-section 6.2.2. The evaluation outcomes are then presented in Sub-section 6.2.3.

6.2.1 The Recommendation Based on Movement Patterns Algorithm (RecoMP)

In this section, the proposed RecoMP algorithm is presented. Recall that the idea is to use knowledge of existing frequently occurring MPs in the DSN to make recommendations. A particular challenge of finding frequently occurring MPs in DSNs is the size of the networks to be considered. The exemplar dataset used for the evaluation reported on in the following sub-section comprised 548,395 vertices and some 3.5 million edges. In other words, it was not possible to mine and maintain all the MPs that might feature in the data set using a single machine. Note that, although the number of MPs generated can be reduced by using a high σ threshold, this is undesirable as we need to use a low σ threshold so as to ensure no significant MPs are missed (the most appropriate value for σ will be considered in Section 4.3). The proposed solution is to mine MPs as required with respect to a specific user and to consequently generate recommendations with respect to that specific user. Users would be considered in turn, but recommendations would be made periodically. It would therefore not be necessary to consider all DSN users in one processing run. In addition, by mining MPs on a required basis, the continuously evolving (dynamic) nature of DSNs can be taken into account.

The pseudo code for the RecoMP process is presented in Algorithm 11. The inputs are: (i) a given user profile u_{new}, (ii) the set of all user profiles U, (iii) the DSN represented as a FET dataset D and (iv) a desired support threshold σ. Note that for illustrative purposes, in Algorithm 11, we have assumed a new user, but this could equally well be an existing user for whom a new set of recommendations is to be generated. The output is a set R of recommendations (matches). The algorithm comprises two sub-processes: (i) ***M*ining** (lines 7 to 21) and (ii) ***R*ecommendation** (lines 22 to 28).

The mining sub-process is where the relevant MPs are generated. MPs are

Algorithm 11: The RecoMPA Algorithm

```
Input:
1   $u_{new}$ = new joined user profile vector
2   $U$ = Collection of all user profile vectors
3   $D$ = Collection of FETs $\{r1, r2, ...\}$ describing network $G$
4   $\sigma$ = Support threshold
Output:
5   $R$ = Set of recommended users
6 Start:
7 Mining Part:
8 M = $\emptyset$
9 $D_{new}$ = Pruning $D$ by looping through $D$ and considering only $FET_i$ where
    $F$ or $T$ similar to $u_{new}$
10 $ShapeSet$ = the set of possible shapes $\{shape_1, shape_2, \dots\}$
11 forall $shape_i \in ShapeSet$ do
12     forall $r_j \in D_{new}$ do
13         if $r_j$ matches $shape_i$ then
14         $MP_k$ = MP extracted from $r_j$
15         if $MP_k$ in $M$ then increment support
16         else $M = M \bigcup \langle MP_k, 1\rangle$
17 forall $MP_i \in M$ do
18     if count for $MP_i < \sigma$ then remove $MP_i$ from $M$
19 Recommendation Part:
20 forall $u_i \in U$ do
21     forall $MP_j \in M$ do
22         if $u_i \subseteq MP_j$ and $u_i \nsubseteq R$ then
23         $R = R \bigcup u_i$
```

stored in a set $M = \{\langle MP_1, count_1\rangle, \langle MP_1, count_1\rangle, \dots\}$. On startup (line 8) M is set to the empty set $\emptyset$. The sub-process commences (line 8) by pruning D to create D_{new} ($D_{new} \subset D$) so that we are left with a set of FETs where either the From and/or the To part correspond (are similar) to u_{new}. The benefit of this pruning is that it results in a significantly reduced search space. Similarity measurement was conducted using the well known Cosine similarity metric calculated as shown in Equation 6.1 where A and B are the set of attribute values of a newly joined user, and a selected user in the network, respectively.

$$cos(A,B) = \frac{\sum_{i=1}^{n} A.B}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{6.1}$$

Next (line 9) a "shape set" is generated to support MP extraction. A shape, as in the case of the ShaMP algorithm described in Chapter 4, is an MP template (prototype) with a particular configuration of attributes taking from the attribute sets A_V and A_E without considering the associated attribute values. Once generated shapes can be populated with attribute values to give candidate MPs. As demonstrated in the context of the ShaMP algorithm presented earlier, the shape concept enhances the efficiency of the MPM in comparison with the more standard Apriori approach. This is because the size of the set of attributes will be less than the size of the set of attribute values. Recall, from Chapter 4 that the maximum number of shapes that can exist in a FET dataset can be determined using Equation 6.2. To calculate the maximum number of MPs Equation 6.2 can be used again but with $|A_v|$ and $|A_E|$ replaced with the size of the set of vertex and edge attribute value sets respectively. Thus if $|A_v| = 2$ and $|A_E| = 2$ the maximum number of shapes will be $(2^2-1)\times(2^2-1)\times(2^2-1) = 27$; if each attribute can have two values the maximum number of MPs will be $2^4-1)\times(2^4-1)\times(2^4-1) = 3375$; a considerable difference. Note also that of the 3375 potential MPs in this example many of them will not feature in the given dataset. Thus the shape concept was used in the context of RecoMP because, although both increase exponentially with the size of the attribute sets, the maximum number of shapes increases at a lower rate than the maximum number of MPs.

$$(2^{|A_v|}-1)\times(2^{|A_E|}-1)\times(2^{|A_v|}-1) \tag{6.2}$$

Returning to algorithm 11 the next step is to populate the set of generated shapes (lines 11 to 18). For each shape $shape_i$ in the shape set, and for each

FET (record) r_j in D_{new}, if r_j matches $shape_i$ then r_j is temporarily stored in a variable MP_k. Note that a record r_j matches a $shape_i$ if the attributes featured in the shape also feature in r_j. If MP_k is already contained in M we increment the associated count (line 15), otherwise we add MP_k to M with a count of 1. Once all shapes have been processed we loop through M (lines 19 to 20) and remove all MPs whose support count is less than σ.

When the set of frequently occurring MPs has been generated the recommender sub-process is commenced (line 22). For each MP MP_j in M, and each user profile (vertex) u_i in U, if u_i is a subset of either the From or To part of MP_j, and has not previously been recorded in R, u_i is appended to R (line 23). In this manner, a set of recommended users is generated.

6.2.2 The Recommendation Based on Collaborative Filtering Algorithm (RecoCF)

To evaluate the proposed RecoMP algorithm described above its operation was compared with a Collaborative Filtering based approach. As noted above, a benchmark Collaborative Filtering system, the RecoCF system, was developed for this purpose. Details concerning this system are presented in this section.

The general methodology for Collaborative Filtering based recommendation, for any system, can be described in terms of the following two steps:

1. Identify users who share the same vector pattern with the service user (the user for whom the prediction is for).
2. Use the preferences of those users founded in step 1 to create a prediction (recommendation) for the service user.

The same methodology was adopted with respect to the RecoCF system. The pseudo code for this system is presented in Algorithm 12. As in the case of the RecoMP algorithm, the RecoCF algorithm takes the same input except there is no need for a σ threshold. The output, as before, is a set of recommended users R. The algorithm commences (line 6), as in the case of the RecoMP algorithm, by pruning the dataset D to give D_{new}. Then for each record (FETs) in D_{new} the From and To attribute value sets are extracted (lines 8 and 9), the sets $From_i$ and To_i. If $From_i$ is a subset of u_{new} (the new user profile) the user profile associated with $From_i$ is added to R if it has not already been included. Similarly, if To_i is a subset of u_{new} the user profile associated with To_i is added to R, again provided if has not already been included. The result is a set R of recommended users (matches).

Algorithm 12: The RecoCF Algorithm

```
   Input:
1    $u_{new}$ = new joined user profile vector
2    $U$ = Collection of all user profile vectors
3    $D$ = Collection of FETs $\{r1, r2, ...\}$ describing network $G$
   Output:
4    $R$ = Set of recommended users
5  Start:
6  $D_{new}$ = Pruning $D$ by looping through $D$ and considering only $FET_i$ where
     $F$ or $T$ similar to $u_{new}$
7  forall $D_i \in D_{new}$ do
8      $From_i$ = return From part from $D_i$
9      $To_i$ = return To part from $D_i$
10     if $From_i \subseteq u_{new}$ and $From_i \nsubseteq R$ then
11     $R = R \bigcup To_i$
12     else if $To_i \subseteq u_{new}$ and $To_i \nsubseteq R$ then
13     $R = R \bigcup From_i$
```

6.2.3 Evaluation

This section reports on the comparison of the operation of the RecoMP system with the RecoCF system. The evaluation was conducted using the DSN dataset presented earlier. The metrics used for the evaluation were: (i) Recall (R), (ii) Precision and (iii) F-score (F) calculated as follows:

$$R = \frac{TP}{TP + FN} \tag{6.3}$$

$$P = \frac{TP}{TP + FP} \tag{6.4}$$

$$F = 2 \times \frac{recall \times precision}{recall + precision} \tag{6.5}$$

where: TP (True Positives) is the number of recommendation correctly predicted, FN (False Negatives) is the number of recommendations we should have predicted but did not predict and FP (False Positives) is the number of incorrect recommendations made. Thus precision is the ratio of the number of correct recommendations over the total number of recommendations, if this is 1.000 we have all the correct recommendations and no wrong ones (we have got all the right recommendations and made no incorrect recommendations). Recall in turn is the ratio of the number of correct recommendations over the total number of recommendations we should have made, if this is 1.000 we have not missed anything. Thus if we have good recall this does not necessarily mean we will have good precision and vice-versa. The F-score, the (harmonic) mean of recall and precision, is thus a good overall measure.

Two sets of experiments were conducted. The first focussed on a random sample of 50 users (without repetition), the aim being to conduct a detailed analysis of the operation of RecoMP compared to RecoCF in the context of individual users. The second used a variation of Ten Cross Validation (TCV) whereby the entire Jiayuan.com FET database was divided into tenths and the process run ten times with a different tenth used for testing. More specifically for each run a random sample of ten users was extracted from the testing tenth and used for the evaluation. In this manner, the process of TCV could be conducted without processing all $548,395$ vertices represented in the database. For both sets of experiments, a threshold value of $\sigma = 1.0$ was used.

The results with respect to the first set of experiments are presented in Table 6.5. From the table it can be seen that recommendations made using the RecoMP algorithm were generally better (more accurate) than those generated using the RecoCF approach, although for some users (#3, #12, #25, #29, #44, #50) the collaborative filtering approach produced recall, precision and F-score values of 1.000. The average recall, precision and F-score using RecoMP were 0.904, 1.000 and 0.941; compared to average recall, precision and F-score values of 0.435, 0.888 and 0.503 using RecoCF. Note also that SD values produced using RecoMP are small compared to when RecoCF is used. It is also interesting to note that the precision using RecoMP was frequently 1.000.

The results with respect to the second set of experiments are given in Tables 6.6 and 6.7; Table 6.6 gives the results using the RecoMP algorithm while Table 6.7 gives the results using the RecoCF algorithm. The tables give the average Precision (P), Recall (R) and F-score (F) for each tenth, and a total average and Standard Deviation (SD). Inspection of the tables clearly indicates that the recommendations made using the RecoMP are better than those generated using the RecoCF. The total average recall, precision and F-score using RecoMP were 0.928, 1.000 and 0.961; compared to total average recall, precision and F-score values of 0.322, 0.744 and 0.392 using RecoCF with small SD values were recorded. Again it is interesting to note that the total average precision using RecoMP, as before, was frequently 1.000; meaning we often make all the correct recommendations and no incorrect recommendations.

Table 6.5: Comparative performance of RecoMP and RecoCF over 50 randomly selected users

#User	RecoMP			RecoCF		
ID	Percision	Recall	F1	Percision	Recall	F1
#1	1.000	1.000	1.000	0.21	0.12	0.355
#2	1.000	1.000	1.000	0.229	0.833	0.359
#3	1.000	1.000	1.000	1.000	1.000	1.000
#4	0.8 00	1.000	0.888	0.072	0.958	0.135
#5	0.966	1.000	0.982	0.986	0.986	0.986
#6	0.857	1.000	0.923	0.651	0.978	0.782
#7	0.764	1.000	0.866	0.344	0.917	0.501
#8	0.915	1.000	0.955	0.976	0.991	0.983
#9	0.980	1.000	0.990	0.892	0.992	0.939
#10	1.000	1.000	1.000	0.047	0.096	0.063
#11	0.887	1.000	0.940	0.356	0.884	0.507
#12	1.000	1.000	1.000	1.000	1.000	1.000
#13	0.997	1.000	0.998	0.382	0.949	0.545
#14	1.000	1.000	1.000	0.647	0.952	0.770
#15	0.932	1.000	0.964	0.991	0.991	0.991
#16	0.822	1.000	0.902	0.994	0.994	0.994
#17	1.000	1.000	1.000	0.148	0.967	0.257
#18	1.000	1.000	1.000	0.158	0.983	0.273
#19	0.971	1.000	0.985	0.676	0.872	0.761
#20	0.949	1.000	0.973	0.042	0.9 00	0.080
#21	0.838	1.000	0.912	0.988	0.988	0.988
#22	1.000	1.000	1.000	0.013	0.794	0.026
#23	0.920	1.000	0.958	0.574	1.000	0.729
#24	1.000	1.000	1.000	0.110	1.000	0.198
#25	0.993	1.000	0.996	1.000	1.000	1.000
#26	0.937	1.000	0.967	0.231	0.941	0.372
#27	0.585	1.000	0.738	0.180	0.9 00	0.300
#28	0.631	1.000	0.774	0.958	0.958	0.958
#29	1.000	1.000	1.000	1.000	1.000	1.000
#30	0.799	1.000	0.888	0.006	1.000	0.012
#31	0.953	1.000	0.976	0.186	0.805	0.303
#32	0.352	1.000	0.521	0.305	0.916	0.458
#33	1.000	1.000	1.000	0.034	0.739	0.065
#34	0.878	1.000	0.935	0.986	0.99	0.988
#35	0.931	1.000	0.964	0.144	0.405	0.212
#36	0.96	1.000	0.979	0.082	0.815	0.149
#37	0.839	1.000	0.912	0.788	0.976	0.872
#38	0.872	1.000	0.931	0.261	0.941	0.409
#39	0.966	1.000	0.983	0.067	0.853	0.125
#40	1.000	1.000	1.000	0.125	0.941	0.222
#41	0.886	1.000	0.94	0.056	0.885	0.107
#42	0.327	1.000	0.493	0.021	0.950	0.042
#43	0.932	1.000	0.965	0.447	0.985	0.615
#44	1.000	1.000	1.000	1.000	1.000	1.000
#45	0.895	1.000	0.945	0.129	0.838	0.224
#46	1.000	1.000	1.000	0.034	0.931	0.065
#47	1.000	1.000	1.000	0.010	0.900	0.021
#48	1.000	1.000	1.000	0.017	0.681	0.033
#49	0.915	1.000	0.956	0.238	0.922	0.378
#50	1.000	1.000	1.000	1.000	1.000	1.000
Average	**0.905**	**1.000**	**0.942**	**0.436**	**0.888**	**0.503**
SD	**0.149**	**0.00**	**0.105**	**0.391**	**0.192**	**0.372**

Table 6.6: TCV results using the RecoMP algorithm

#	**RecoMP**		
Tenth	**Percision**	**Recall**	**F1**
# 1	0.938	1.000	0.978
# 2	0.908	1.000	0.949
# 3	0.917	1.000	0.956
# 4	0.948	1.000	0.972
# 5	0.948	1.000	0.972
# 6	0.948	1.000	0.972
# 7	0.928	1.000	0.961
# 8	0.928	1.000	0.961
# 9	0.952	1.000	0.974
# 10	0.867	1.000	0.917
Avarage	**0.928**	**1.000**	**0.961**
SD	**0.02**	**0.00**	**0.02**

Table 6.7: TCV results using the RecoCF algorithm

#	**RecoCF**		
Tenth	**Percision**	**Recall**	**F1**
# 1	0.217	0.764	0.298
# 2	0.369	0.831	0.470
# 3	0.325	0.760	0.416
# 4	0.305	0.722	0.364
# 5	0.305	0.722	0.364
# 6	0.305	0.722	0.364
# 7	0.333	0.756	0.424
# 8	0.354	0.763	0.416
# 9	0.265	0.683	0.361
# 10	0.446	0.717	0.439
Avarage	**0.322**	**0.744**	**0.392**
SD	**0.058**	**0.038**	**0.048**

TABLE 6.8: The accuracy of RecoMPA with respect to a range of σ values (ranging from 1.0 to 7.5 increasing in steps of 2.5) over 50 randomly selected users.

#	Sigma = 1.0			sigma = 2.5			sigma = 5.0			sigma = 7.5		
Users	**Percision**	**Recall**	**F1**	**Percision**	**Recall**	**F1**	**Percision**	**Recall**	**F1**	**Percision**	**Recall**	**F1**
#1	1.000	1.000	1.000	Null	Null	Null	Null	Null	Null	Null	Null	Nul l
#2	1.000	1.000	1.000	Null	Null	Null	Null	Null	Null	Null	Null	Null
#3	1.000	1.000	1.000	Null	Null	Null	Null	Null	Null	Null	Null	Null
#4	0.8 00	1.000	0.888	0.8 00	1.000	0.888	Null	Null	Null	Null	Null	Null
#5	0.966	1.000	0.982	0.232	1.000	0.376	0.232	1.000	0.376	Null	Null	Null
#6	0.857	1.000	0.923	0.857	0.998	0.922	Null	Null	Null	Null	Null	Null
#7	0.764	1.000	0.866	0.764	1.000	0.866	0.764	1.000	0.866	Null	Null	Null
#8	0.915	1.000	0.955	0.914	0.997	0.954	Null	Null	Null	Null	Null	Null
#9	0.980	1.000	0.990	Null	Null	Null	Null	Null	Null	Null	Null	Null
#10	1.000	1.000	1.000	1.000	1.000	1.000	Null	Null	Null	Null	Null	Null
#11	0.887	1.000	0.940	Null	Null	Null	Null	Null	Null	Null	Null	Null

Continue on the next page

TABLE 6.8: The accuracy of RecoMPA with respect to a range of σ values (ranging from 1.0 to 7.5 increasing in steps of 2.5) over 50 randomly selected users (cont.).

#	Sigma = 1.0			sigma = 2.5			sigma = 5.0			sigma = 7.5		
Users	**Percision**	**Recall**	**F1**	**Percision**	**Recall**	**F1**	**Percision**	**Recall**	**F1**	**Percision**	**Recall**	**F1**
#12	1.000	1.000	1.000	1.000	1.000	1.000	Null	Null	Null	Null	Null	Null
#13	0.997	1.000	0.998	0.997	0.997	0.997	Null	Null	Null	Null	Null	Null
#14	1.000	1.000	1.000	1.000	0.996	0.998	Null	Null	Null	Null	Null	Null
#15	0.932	1.000	0.964	0.932	1.000	0.964	Null	Null	Null	Null	Null	Null
#16	0.822	1.000	0.902	0.822	1.000	0.902	Null	Null	Null	Null	Null	Null
#17	1.000	1.000	1.000	Null	Null	Null	Null	Null	Null	Null	Null	Null
#18	1.000	1.000	1.000	1.000	1.000	1.000	Null	Null	Null	Null	Null	Null
#19	0.971	1.000	0.985	0.971	1.000	0.985	Null	Null	Null	Null	Null	Null
#20	0.949	1.000	0.973	0.949	1.000	0.973	Null	Null	Null	Null	Null	Null
#21	0.838	1.000	0.912	Null	Null	Null	Null	Null	Null	Null	Null	Null
#22	1.000	1.000	1.000	Null	Null	Null	Null	Null	Null	Null	Null	Null

Continue on the next page

TABLE 6.8: The accuracy of RecoMPA with respect to a range of σ values (ranging from 1.0 to 7.5 increasing in steps of 2.5) over 50 randomly selected users (cont.).

#	Sigma = 1.0			sigma = 2.5			sigma = 5.0			sigma = 7.5		
Users	**Percision**	**Recall**	**F1**	**Percision**	**Recall**	**F1**	**Percision**	**Recall**	**F1**	**Percision**	**Recall**	**F1**
#23	0.920	1.000	0.958	0.920	1.000	0.958	Null	Null	Null	Null	Null	Null
#24	1.000	1.000	1.000	1.000	1.000	1.000	Null	Null	Null	Null	Null	Null
#25	0.993	1.000	0.996	0.993	1.000	0.996	Null	Null	Null	Null	Null	Null
#26	0.937	1.000	0.967	0.937	1.000	0.967	Null	Null	Null	Null	Null	Null
#27	0.585	1.000	0.738	0.585	1.000	0.738	Null	Null	Null	Null	Null	Null
#28	0.631	1.000	0.774	0.631	1.000	0.774	Null	Null	Null	Null	Null	Null
#29	1.000	1.000	1.000	1.000	1.000	1.000	Null	Null	Null	Null	Null	Null
#30	0.799	1.000	0.888	0.799	0.998	0.887	Null	Null	Null	Null	Null	Null
#31	0.953	1.000	0.976	0.953	1.000	0.976	Null	Null	Null	Null	Null	Null
#32	0.352	1.000	0.521	0.352	1.000	0.521	Null	Null	Null	Null	Null	Null
#33	1.000	1.000	1.000	1.000	1.000	1.000	Null	Null	Null	Null	Null	Null

Continue on the next page

TABLE 6.8: The accuracy of RecoMPA with respect to a range of σ values (ranging from 1.0 to 7.5 increasing in steps of 2.5) over 50 randomly selected users (cont.).

#	Sigma = 1.0			sigma = 2.5			sigma = 5.0			sigma = 7.5		
Users	**Percision**	**Recall**	**F1**	**Percision**	**Recall**	**F1**	**Percision**	**Recall**	**F1**	**Percision**	**Recall**	**F1**
#34	0.931	1.000	0.964	Null	Null	Null	Null	Null	Null	Null	Null	Null
#35	0.960	1.000	0.979	0.960	1.000	0.979	Null	Null	Null	Null	Null	Null
#36	0.839	1.000	0.912	0.839	1.000	0.912	Null	Null	Null	Null	Null	Null
#37	0.872	1.000	0.931	0.872	1.000	0.931	Null	Null	Null	Null	Null	Null
#38	0.966	1.000	0.983	0.966	0.982	0.974	Null	Null	Null	Null	Null	Null
#39	1.000	1.000	1.000	1.000	1.000	1.000	Null	Null	Null	Null	Null	Null
#40	0.886	1.000	0.940	0.886	1.000	0.940	Null	Null	Null	Null	Null	Null
#41	0.327	1.000	0.493	0.327	1.000	0.493	Null	Null	Null	Null	Null	Null
#42	0.932	1.000	0.965	0.932	1.000	0.965	Null	Null	Null	Null	Null	Null
#43	1.000	1.000	1.000	Null	Null	Null	Null	Null	Null	Null	Null	Null
#44	1.000	1.000	1.000	1.000	1.000	1.000	Null	Null	Null	Null	Null	Null

Continue on the next page

Table 6.8: The accuracy of RecoMPA with respect to a range of σ values (ranging from 1.0 to 7.5 increasing in steps of 2.5) over 50 randomly selected users (cont.).

#	Sigma = 1.0			sigma = 2.5			sigma = 5.0			sigma = 7.5		
Users	**Percision**	**Recall**	**F1**	**Percision**	**Recall**	**F1**	**Percision**	**Recall**	**F1**	**Percision**	**Recall**	**F1**
#45	1.000	1.000	1.000	1.000	1.000	1.000	Null	Null	Null	Null	Null	Null
#46	1.000	1.000	1.000	1.000	1.000	1.000	Null	Null	Null	Null	Null	Null
#47	0.915	1.000	0.956	0.915	1.000	0.956	Null	Null	Null	Null	Null	Null
#48	1.000	1.000	1.000	1.000	1.000	1.000	Null	Null	Null	Null	Null	Null
#49	1.000	1.000	1.000	1.000	1.000	1.000	Null	Null	Null	Null	Null	Null
#50	1.000	1.000	1.000	1.000	1.000	1.000	Null	Null	Null	Null	Null	Null
Average	0.910	1.000	0.944	0.878	0.999	0.920	0.498	1.000	0.621	Null	Null	Null
SD	0.150	0.000	0.106	0.193	0.003	0.145	0.376	0.000	0.346	Null	Null	Null

6.3 Summary and Conclusions

This chapter has explored the utility if extracted MPs in the global and local contexts. In the global context to predict traffic within a previously unseen network, either a future rendition of the network from which the MPs were mined or a separate but closely related network. In the local context to predict the traffic to and from a given network to a newly added node. In the context of the latter, it was noted that local prediction could be viewed as a form of recommendation and this local MP prediction was discussed in this context. From the reported evaluation it was firstly concluded, in the context of global prediction, that the proposed mechanism could successfully predict MPs in large networks. Secondly, it was concluded, in the context of local prediction, that the usage of movement patterns for recommendation purposes provide for a significant improvement in accuracy compared to the traditional collaborative filtering based approach. The work presented in this chapter thus provided a satisfactory answer to the fourth subsidiary research question considered in this thesis. In the following chapter, this thesis is concluded with a summary of the main findings and some suggestions for future work.

7 Conclusion And Future Vision

Large networks are very powerful representations of data and they are everywhere in real life, The Web, social networks and traffic flow are only a few examples of large networks, which often involved hundreds of millions or even billions of nodes and edges. The main challenge within this abundance of nodes and edges is how to efficiently and effectively extracted knowledge from such large networks. Therefore, the overarching theme of this thesis has been knowledge extraction from the large network. The idea was to develop learning mechanisms able to extract the behaviour of traffic in large networks. More specifically, the idea was to extract this behaviour and express this in the form of Movement Pattern (MPs) which could then can be used to predict the nature of traffic movements in the context of networks that were either similar to those from which the patterns were extracted; or, in the case of time-variant networks, future (or past) renditions of the network. To this end, the work in this thesis was divided into two parts, the first part was based on investigating and developing an efficient mechanism for Movement Pattern Mining (MPM), while the second part was based on the utilization of the extracted MPs in term of prediction. This chapter presents a summary of the research presented in this thesis in terms of the research issues addressed, the research contributions and how the research can be progressed further.

7.1 Summary And Main Findings

As stated in Chapter 1, the key aim of the work described in this thesis was to provide an answer to the research question: *"How can the concept of movement patterns, as envisaged in this thesis, be efficiently and effectively extracted from large networks, and how can those movement patterns best be utilised"*. Five further subsidiary questions were identified which needed to be answered before an answer to the overriding research question could be postulated. This section presents a summary of the work presented in this thesis. The summary commences by considering the subsidiary questions first and then goes on to consider the main research question, as follow:

Q1: What is the most appropriate mechanism for preprocessing and representing large network data so that movement patterns can be extracted?

After considering various potential models it was decided that the most appropriate mechanism was to process the networks so that they were represented in the form of a three-part tuple $\langle From, Edge, To \rangle$, where each part had a number of attributes associated with it. This representation was termed the FET representation.

Q2: Given a solution to (1) what are the most appropriate mechanisms whereby movement patterns can be mined (learnt/extracted) from network data?

This question was answered in Chapter 4 where two different Movement Pattern Mining approaches were investigated, the Apriori Movement Pattern (AMP) approach and the ShaMP Movement Pattern (ShaMP) approach. The two approaches are similar in that they take FET data as input and output collections of MPs, the only distinguishing feature between them was the mechanism for extracting the MPs. The first was founded on a traditional Apriori approach to pattern mining, while the second adopted the novel approach of considering the "shape" of the MPs to be identified. In other words, the second approach used knowledge of the nature of MPs to realise anticipated efficiency gains.

Q3: Following on from (2), given mechanisms for mining movement patterns from network data how can these mechanisms be scaled up to address movement pattern mining from very large networks?

This question was answered in Chapter 5, where it was noted that the nature of the ShaMP algorithm was well suitable to scaling up, given a very large network, because shapes could be processed independently. Three different distributed approaches were investigated, namely: (i) Hadoop/MapReduce, (ii) Shared Memory Systems and (iii) Distributed Memory System. Consequently, three ShaMP variations were proposed: (i) The ShaMP Shared Memory System (ShaMP SMS), (ii) the ShaMP Distribute Memory System (ShaMP DMS) and (iii) the ShaMP Hadoop/MapReduce System (ShaMP HMS). The main finding was, as anticipated, that the ShaMP algorithm was well suited to handle very large networks by utilizing big data facilities.

Q4: Once we have a collection of movement patterns how can they best be applied to previously unseen network data and how do we know whether the manner in which movement patterns are applied produces the correct results?

This question was answered in Chapter 6. Two prediction contexts were considered: the Global Context and Local Context. In the Global Context MPs were used to predict the nature of traffic movement (network connectivity) in the context of entire networks that were either similar to those from which the MPs were extracted; or, in the case of time-variant networks, future (or past) renditions of the network. In the Local Context, MPs were utilized to predict the nature of traffic movement (connectivity) between a newly introduced network node and the remainder of a given network. The main findings were that MP concept could be successfully employed for prediction purpose. The prediction accuracy in the

Global Context was tested using CTS network training and test datasets (2003, 2004, 2005 and 2006). The idea was to compare the set of predicted MPs, M', with the set of MPs known to exist in the test data. More specifically the evaluation was conducted by applying the MPs extracted with respect to one year to the vertices of all other years and extracting MPs. In each case, the resulting set M' of predicted MPs was compared with the set of known "ground truth" MPs M_T. The prediction accuracy in the Local Context was tested by considering a recommendation system that might be embedded in a DSN. The main finding was that MPs can be successfully employed for the purpose of recommendation with a high degree of accuracy.

Referring to Section 1.4 in Chapter 1, A number of additional criteria were also devised for measuring the overall success of the research work conducted and presented in this thesis, as follows:

- **Generality:** The proposed MPM is generic. The proposed algorithms were applied to two different datasets, the CTS dataset, and the DSN dataset. However, clearly, MPM has wider application.
- **Efficiency:** For each conducted experiment, when using the ShaMP algorithm reasonable runtimes were recorded comparing to a more traditional Apriori approach. The memory resource required to process and store the MPs was also found not to be excessive.
- **Effectiveness:** Clearly the proposed MPM algorithms and prediction mechanisms were also effective in that the correct movement patterns were extracted and high prediction accuracy was achieved when applying the MPs.

7.2 Research Contributions

The main contributions of the research work considered in this thesis were presented in section 1.5. For convenience these are listed again here as follows:

1. The Apriori-based Movement Pattern (AMP) algorithm
2. The Shape-based Movement Pattern (ShaMP) algorithm
3. Three variations of the ShaMP algorithm based on distributed/shared memory systems and Hadoop/MapReduce, with a comparison of their operation.
4. A mechanism for applying MPs to predict traffic in networks.
5. The RecoMP recommender system RecoMPA which utilize MPs so as to provide recommendations in the context of DSN.
6. A mechanism for evaluating the effectiveness of the use of movement patterns in the context of prediction.
7. A mechanism for evaluating the effectiveness of the use of movement patterns in the context of recommendation.

7.3 Future Vision

Although the contributions of the thesis are significant with respect to MPM and traffic prediction, there are a number of areas which merit further investigation and future consideration so as to enhance the functionality and increase the overall quality of the work. The research described in this thesis has indicated a number of future interesting research direction to enhance the operation of the MPM and prediction mechanisms as follows:

1. **ShaMP algorithm improvements:** The efficiency of the ShaMP algorithm can be improved by making a hybrid model comprised of both the ShaMP and AMP algorithm. The possible scenario can benefit from the advantages of both algorithms. By first pruning any attribute values that occurred below a specified support threshold, and then generate a shape set for those who pass the support threshold.

2. **Using Movement Pattern to build a technique for DDoS attack detection:** The concept of MPs could be used to detect Denial-of-service attack (DoS attacks) over a network, by implementing in-time movement pattern mining over the traffic network and comparing the extracted MPs with MPs for the normal behaviour so as to check for DoS attacks in real time.

3. **Traffic prediction Using GAN:** The current traffic prediction mechanisms can be further investigated in terms of traffic prediction modeling based on Generative Adversarial Networks (GAN) [37]. This would also be of interest with respect to the wider research vision.

Bibliography

[1] Lada A Adamic and Eytan Adar, *Friends and neighbors on the web*, Social networks **25** (2003), no. 3, 211–230.

[2] Ramesh C Agarwal, Charu C Aggarwal, and VVV Prasad, *A tree projection algorithm for generation of frequent item sets*, Journal of parallel and Distributed Computing **61** (2001), no. 3, 350–371.

[3] Charu C Aggarwal, *Applications of frequent pattern mining*, Frequent Pattern Mining, Springer, 2014, pp. 443–467.

[4] Charu C Aggarwal and Jiawei Han, *Frequent pattern mining*, Springer, 2014.

[5] Rakesh Agrawal and John C Shafer, *Parallel mining of association rules*, IEEE Transactions on knowledge and Data Engineering **8** (1996), no. 6, 962–969.

[6] Rakesh Agrawal, Ramakrishnan Srikant, et al., *Fast algorithms for mining association rules*, Proc. 20th int. conf. very large data bases, VLDB, vol. 1215, 1994, pp. 487–499.

[7] Xavier Amatriain, *Big & personal: data and models behind netflix recommendations*, Proceedings of the 2nd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, ACM, 2013, pp. 1–6.

[8] Gregory M Ames, Dylan B George, Christian P Hampson, Andrew R Kanarek, Cayla D McBee, Dale R Lockwood, Jeffrey D Achter, and Colleen T Webb, *Using network properties to predict disease dynamics on*

human contact networks, Proceedings of the Royal Society of London B: Biological Sciences **278** (2011), no. 1724, 3544–3550.

[9] David C Anastasiu, Jeremy Iverson, Shaden Smith, and George Karypis, *Big data frequent pattern mining*, Frequent Pattern Mining, Springer, 2014, pp. 225–259.

[10] Paul Anderson, *Web 2.0 and beyond: Principles and technologies*, Chapman and Hall/CRC, 2016.

[11] E Ansari, GH Dastghaibifard, M Keshtkaran, and H Kaabi, *Distributed frequent itemset mining using trie data structure.*, IAENG International Journal of Computer Science **35** (2008), no. 3.

[12] Mark Baker, Bryan Carpenter, and A Shaft, *Mpj express: towards thread safe java hpc*, Cluster Computing, 2006 IEEE International Conference on, IEEE, 2006, pp. 1–10.

[13] Elena Baralis, Tania Cerquitelli, Silvia Chiusano, and Alberto Grand, *P-mine: Parallel itemset mining on large datasets*, Data Engineering Workshops (ICDEW), 2013 IEEE 29th International Conference on, IEEE, 2013, pp. 266–271.

[14] Blaise Barney et al., *Introduction to parallel computing*, Lawrence Livermore National Laboratory **6** (2010), no. 13, 10.

[15] Mathieu Bastian, Sebastien Heymann, Mathieu Jacomy, et al., *Gephi: an open source software for exploring and manipulating networks.*, Icwsm **8** (2009), 361–362.

[16] Michael Batty, Jake DeSyllas, and Elspeth Duxbury, *The discrete dynamics of small-scale spatial events: agent-based models of mobility in carnivals and street parades*, International Journal of Geographical Information Science **17** (2003), no. 7, 673–697.

[17] Catherine A Bliss, Morgan R Frank, Christopher M Danforth, and Peter Sheridan Dodds, *An evolutionary algorithm approach to link prediction in dynamic social networks*, Journal of Computational Science **5** (2014), no. 5, 750–764.

[18] Steven Brawer, *Introduction to parallel programming*, Academic Press, 2014.

[19] Xiongcai Cai, Michael Bain, Alfred Krzywicki, Wayne Wobcke, Yang Sok Kim, Paul Compton, and Ashesh Mahidadia, *Learning collaborative filtering and its application to people to people recommendation in social networks*, Data Mining (ICDM), 2010 IEEE 10th International Conference on, IEEE, 2010, pp. 743–748.

[20] Balakrishnan Chandrasekaran, *Survey of network traffic models*, Wasching-ton University in St. Louis CSE **567** (2009).

[21] Pavel Chebotarev and Elena Shamis, *The matrix-forest theorem and measuring relations in small social groups*, arXiv preprint math/0602070 (2006).

[22] Lin Chen, CholLi Wang, and Francis CM Lau, *A grid middleware for distributed java computing with mpi binding and process migration supports*, Journal of Computer Science and Technology **18** (2003), no. 4, 505–514.

[23] David W Cheung, Jiawei Han, Vincent T Ng, Ada W Fu, and Yongjian Fu, *A fast distributed algorithm for mining association rules*, Parallel and Distributed Information Systems, 1996., Fourth International Conference on, IEEE, 1996, pp. 31–42.

[24] Gobinda G Chowdhury, *Introduction to modern information retrieval*, Facet publishing, 2010.

[25] Aaron Clauset, Cristopher Moore, and Mark EJ Newman, *Hierarchical structure and the prediction of missing links in networks*, arXiv preprint arXiv:0811.0484 (2008).

[26] Souptik Datta, Kanishka Bhaduri, Chris Giannella, Ran Wolff, and Hillol Kargupta, *Distributed data mining in peer-to-peer networks*, Internet Computing, IEEE **10** (2006), no. 4, 18–26.

[27] Jeffrey Dean and Sanjay Ghemawat, *Mapreduce: simplified data processing on large clusters*, Communications of the ACM **51** (2008), no. 1, 107–113.

[28] B Dumont, A Boissy, C Achard, AM Sibbald, and HW Erhard, *Consistency of animal order in spontaneous group movements allows the measurement of leadership in a group of grazing heifers*, Applied Animal Behaviour Science **95** (2005), no. 1, 55–66.

[29] Zahra Farzanyar and Nick Cercone, *Efficient mining of frequent itemsets in social network data based on mapreduce framework*, Proceedings of the

2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ACM, 2013, pp. 1183–1188.

[30] Michael Fleischman, Phillip Decamp, and Deb Roy, *Mining temporal patterns of movement for video content classification*, Proceedings of the 8th ACM international workshop on Multimedia information retrieval, ACM, 2006, pp. 183–192.

[31] Francois Fouss, Alain Pirotte, Jean-Michel Renders, and Marco Saerens, *Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation*, IEEE Transactions on knowledge and data engineering **19** (2007), no. 3, 355–369.

[32] Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer, *Learning probabilistic relational models*, IJCAI, vol. 99, 1999, pp. 1300–1309.

[33] John Galloway and Simeon J Simoff, *Network data mining: methods and techniques for discovering deep linkage between attributes*, Proceedings of the 3rd Asia-Pacific conference on Conceptual modelling-Volume 53, Australian Computer Society, Inc., 2006, pp. 21–32.

[34] Győző Gidófalvi and Ehsan Saqib, *From trajectories of moving objects to route-based traffic prediction and management*, Proc. of GIScience, From trajectories of moving objects to route-based traffic prediction and management, 2010.

[35] Bart Goethals, *Survey on frequent pattern mining*, Univ. of Helsinki **19** (2003), 840–852.

[36] Hector Gonzalez, Jiawei Han, Xiaolei Li, Margaret Myslinska, and John Paul Sondag, *Adaptive fastest path computation on a road network: a traffic mining approach*, Proceedings of the 33rd international conference on Very large data bases, VLDB Endowment, 2007, pp. 794–805.

[37] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, *Generative adversarial nets*, Advances in neural information processing systems, 2014, pp. 2672–2680.

[38] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum, *A high-performance, portable implementation of the mpi message passing interface standard*, Parallel computing **22** (1996), no. 6, 789–828.

[39] Eui-Hong Han, George Karypis, and Vipin Kumar, *Scalable parallel data mining for association rules*, IEEE Transactions on Knowledge and Data Engineering **12** (2000), no. 3, 337–352.

[40] David Heckerman, Chris Meek, and Daphne Koller, *Probabilistic entity-relationship models, prms, and plate models*, Introduction to statistical relational learning (2007), 201–238.

[41] G Jeh and J Widom, *A measure of structural-context similarity [c]*, Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2002, pp. 538–543.

[42] Huzefa Kagdi, Michael L Collard, and Jonathan I Maletic, *A survey and taxonomy of approaches for mining software repositories in the context of software evolution*, Journal of Software: Evolution and Process **19** (2007), no. 2, 77–131.

[43] Nicholas T Karonis, Brian Toonen, and Ian Foster, *Mpich-g2: A grid-enabled implementation of the message passing interface*, Journal of Parallel and Distributed Computing **63** (2003), no. 5, 551–563.

[44] Leo Katz, *A new status index derived from sociometric analysis*, Psychometrika **18** (1953), no. 1, 39–43.

[45] David Kempe, Jon M Kleinberg, and Éva Tardos, *Maximizing the spread of influence through a social network.*, Theory of Computing **11** (2015), no. 4, 105–147.

[46] Myunghwan Kim and Jure Leskovec, *The network completion problem: Inferring missing nodes and edges in networks.*, SDM, vol. 11, SIAM, 2011, pp. 47–58.

[47] Christine Kiss and Martin Bichler, *Leveraging network effects for predictive modelling in customer relationship management*, 15th Annual Workshop on Information Technolgies & Systems (WITS) Paper, 2005.

[48] Valdis E Krebs, *Mapping networks of terrorist cells*, Connections **24** (2002), no. 3, 43–52.

[49] Alfred Krzywicki, Wayne Wobcke, Yang Sok Kim, Xiongcai Cai, Michael Bain, Paul Compton, and Ashesh Mahidadia, *Evaluation and deployment of a people-to-people recommender in online dating*, 26th Innovative Applications of Artificial Intelligence Conference 2014, vol. 4, 2014, pp. 2914–2921.

[50] Jérôme Kunegis, Gerd Gröner, and Thomas Gottron, *Online dating recommender systems: The split-complex number approach*, Proceedings of the 4th ACM RecSys workshop on Recommender systems and the social web, ACM, 2012, pp. 37–44.

[51] Sangeetha Kutty, Richi Nayak, and Lin Chen, *A people-to-people matching system using graph mining techniques*, World Wide Web **17** (2014), no. 3, 311–349.

[52] Chuck Lam, *Hadoop in action*, Manning Publications Co., 2010.

[53] Elizabeth A Leicht, Petter Holme, and Mark EJ Newman, *Vertex similarity in networks*, Physical Review E **73** (2006), no. 2, 026120.

[54] Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, and Edward Y Chang, *Pfp: parallel fp-growth for query recommendation*, Proceedings of the 2008 ACM conference on Recommender systems, ACM, 2008, pp. 107–114.

[55] Yen-hui Liang and Shiow-yang Wu, *Sequence-growth: A scalable and effective frequent itemset mining algorithm for big data based on mapreduce framework*, Big Data (BigData Congress), 2015 IEEE International Congress on, IEEE, 2015, pp. 393–400.

[56] David Liben-Nowell and Jon Kleinberg, *The link-prediction problem for social networks*, journal of the Association for Information Science and Technology **58** (2007), no. 7, 1019–1031.

[57] Ming-Yen Lin, Pei-Yu Lee, and Sue-Chen Hsueh, *Apriori-based frequent itemset mining algorithms on mapreduce*, Proceedings of the 6th international conference on ubiquitous information management and communication, ACM, 2012, p. 76.

[58] Greg Linden, Brent Smith, and Jeremy York, *Amazon. com recommendations: Item-to-item collaborative filtering*, IEEE Internet computing **7** (2003), no. 1, 76–80.

[59] Jian-Guo Liu, Lei Hou, Xue Pan, Qiang Guo, and Tao Zhou, *Stability of similarity measurements for bipartite networks*, Scientific reports **6** (2016), 18653.

[60] Weiping Liu and Linyuan Lü, *Link prediction based on local random walk*, EPL (Europhysics Letters) **89** (2010), no. 5, 58007.

[61] Linyuan Lü, Ci-Hang Jin, and Tao Zhou, *Similarity index based on local paths for link prediction of complex networks*, Physical Review E **80** (2009), no. 4, 046122.

[62] Linyuan Lü and Tao Zhou, *Link prediction in weighted networks: The role of weak ties*, EPL (Europhysics Letters) **89** (2010), no. 1, 18001.

[63] Linyuan Lu and Tao Zhou, *Link prediction in complex networks: A survey*, Physica A: statistical mechanics and its applications **390** (2011), no. 6, 1150–1170.

[64] Yiding Lu, Yufan Guo, and Anna Korhonen, *Link prediction in drug-target interactions network using similarity indices*, BMC bioinformatics **18** (2017), no. 1, 39.

[65] John Makhoul, Francis Kubala, Richard Schwartz, Ralph Weischedel, et al., *Performance measures for information extraction*, Proceedings of DARPA broadcast news workshop, Herndon, VA, 1999, pp. 249–252.

[66] Zeev Maoz, *Preferential attachment, homophily, and the structure of international networks, 1816–2003*, Conflict Management and Peace Science **29** (2012), no. 3, 341–369.

[67] Naohiro Matsumura, David E Goldberg, and Xavier Llorà, *Mining directed social network from message board*, Special interest tracks and posters of the 14th international conference on World Wide Web, ACM, 2005, pp. 1092–1093.

[68] Sandy Moens, Emin Aksehirli, and Bart Goethals, *Frequent itemset mining for big data*, Big Data, 2013 IEEE international conference on, IEEE, 2013, pp. 111–118.

[69] Joshua O'Madadhain, Danyel Fisher, Padhraic Smyth, Scott White, and Yan-Biao Boey, *Analysis and visualization of network data using jung*, Journal of Statistical Software **10** (2005), no. 2, 1–35.

[70] J Hernández Palancar, O Fraxedas Tormo, J Festón Cárdenas, and R Hernández León, *Distributed and shared memory algorithm for parallel mining of association rules*, International Workshop on Machine Learning and Data Mining in Pattern Recognition, Springer, 2007, pp. 349–363.

[71] Jian Pei and Jiawei Han, *Constrained frequent pattern mining: a pattern-growth view*, ACM SIGKDD Explorations Newsletter **4** (2002), no. 1, 31–39.

[72] Luiz Pizzato, Tomasz Rej, Joshua Akehurst, Irena Koprinska, Kalina Yacef, and Judy Kay, *Recommending people to people: the nature of reciprocal recommenders with a case study in online dating*, User Modeling and User-Adapted Interaction **23** (2013), no. 5, 447–488.

[73] Luiz Pizzato, Tomek Rej, Thomas Chung, Irena Koprinska, and Judy Kay, *Recon: a reciprocal recommender for online dating*, Proceedings of the fourth ACM conference on Recommender systems, ACM, 2010, pp. 207–214.

[74] Jelica Protic, Milo Tomasevic, and Veljko Milutinovic, *Distributed shared memory: Concepts and systems*, IEEE Parallel & Distributed Technology: Systems & Applications **4** (1996), no. 2, 63–71.

[75] Abhijit Raorane and RV Kulkarni, *Data mining techniques: A source for consumer behavior analysis*, arXiv preprint arXiv:1109.1202 (2011).

[76] Paul Resnick and Hal R Varian, *Recommender systems*, Communications of the ACM **40** (1997), no. 3, 56–58.

[77] Matthew Richardson and Pedro Domingos, *Mining knowledge-sharing sites for viral marketing*, Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2002, pp. 61–70.

[78] Ramesh R Sarukkai, *Link prediction and path analysis using markov chains*, Computer Networks **33** (2000), no. 1, 377–386.

[79] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker, *Cytoscape: a software environment for integrated models of biomolecular interaction networks*, Genome research **13** (2003), no. 11, 2498–2504.

[80] Oliver Sharp, David Wortendyke, Scot Gellock, Robert Wahbe, and Paul Viola, *Interest graph-powered feed*, August 8 2017, US Patent 9,727,618.

[81] Sudhakar Singh, Rakhi Garg, and PK Mishra, *Review of apriori based algorithms on mapreduce framework*, arXiv preprint arXiv:1702.06284 (2017).

[82] Shahab Saquib Sohail, Jamshed Siddiqui, and Rashid Ali, *Book recommendation system using opinion mining technique*, Advances in Computing, Communications and Informatics (ICACCI), 2013 International Conference on, IEEE, 2013, pp. 1609–1614.

[83] Lei Tang and Huan Liu, *Community detection and mining in social media*, Synthesis Lectures on Data Mining and Knowledge Discovery **2** (2010), no. 1, 1–137.

[84] Akhilesh Tiwari, Rajendra K Gupta, and Dharma P Agrawal, *A survey on frequent pattern mining: Current status and challenging issues*, Information Technology Journal **9** (2010), no. 7, 1278–1293.

[85] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan, *Fast random walk with restart and its applications*, Data Mining, 2006. ICDM'06. Sixth International Conference on, IEEE, 2006, pp. 613–622.

[86] Kun Tu, Bruno Ribeiro, David Jensen, Don Towsley, Benyuan Liu, Hua Jiang, and Xiaodong Wang, *Online dating recommendations: matching markets and learning preferences*, Proceedings of the 23rd International Conference on World Wide Web, ACM, 2014, pp. 787–792.

[87] Muhammad Tufail, Frans Coenen, Tintin Mu, and Saqib Jamshid Rind, *Mining movement patterns from video data to inform multi-agent based simulation*, International Workshop on Agents and Data Mining Interaction, Springer, 2014, pp. 38–51.

[88] Abhinav Vishnu and Khushbu Agarwal, *Large scale frequent pattern mining using mpi one-sided model*, Cluster Computing (CLUSTER), 2015 IEEE International Conference on, IEEE, 2015, pp. 138–147.

[89] Lan Vu and Gita Alaghband, *Novel parallel method for mining frequent patterns on multi-core shared memory systems*, Proceedings of the 2013 International Workshop on Data-Intensive Scalable Computing Systems, ACM, 2013, pp. 49–54.

[90] Xinxi Wang and Ye Wang, *Improving content-based and hybrid music recommendation using deep learning*, Proceedings of the 22nd ACM international conference on Multimedia, ACM, 2014, pp. 627–636.

[91] Tom White, *Hadoop: The definitive guide*, " O'Reilly Media, Inc.", 2012.

[92] Dawen Xia, Yanhui Zhou, Zhuobo Rong, and Zili Zhang, *Ipfp: An improved parallel fp-growth algorithm for frequent itemsets mining*, School of Computer and Information Science, Southwest University, Chongqing, China, Institute of Statistics, Southwest University, Chongqing, China, School of Information Technology, Deakin University, Victoria, Australia

Guizhou Minzu University, Guiyang, China, Corresponding author: Zili Zhang (2013), 4034–4039.

[93] J Xia, Vic Ciesielski, and Colin Arrowsmith, *Data mining of tourists spatio-temporal movement patterns: A case study on phillip island*, Proceedings of the Eighth d International Conference on GeoComputation, 2005, pp. 1–15.

[94] Peng Xia, Shuangfei Zhai, Benyuan Liu, Yizhou Sun, and Cindy Chen, *Design of reciprocal recommendation systems for online dating*, Social Network Analysis and Mining **6** (2016), no. 1, 1–16.

[95] Jennifer J Xu and Hsinchun Chen, *Fighting organized crimes: using shortest-path algorithms to identify associations in criminal networks*, Decision Support Systems **38** (2004), no. 3, 473–487.

[96] Yaling Xun, Jifu Zhang, Xiao Qin, and Xujun Zhao, *Fidoop-dp: data partitioning in frequent itemset mining on hadoop clusters*, IEEE Transactions on Parallel and Distributed Systems **28** (2017), no. 1, 101–114.

[97] Othman Yahya, Osman Hegazy, and Ehab Ezat, *An efficient implementation of a-priori algorithm based on hadoop-mapreduce model*, International sjournal of Reviews in Computing **12** (2012).

[98] Xin Yue Yang, Zhen Liu, and Yan Fu, *Mapreduce as a programming model for association rules algorithm on hadoop*, Information Sciences and Interaction Sciences (ICIS), 2010 3rd International Conference on, IEEE, 2010, pp. 99–102.

[99] Kai Yu, Wei Chu, Shipeng Yu, Volker Tresp, and Zhao Xu, *Stochastic relational models for discriminative link prediction*, Advances in neural information processing systems, 2007, pp. 1553–1560.

[100] Kun-Ming Yu, Jiayi Zhou, Tzung-Pei Hong, and Jia-Ling Zhou, *A load-balanced distributed parallel mining algorithm*, Expert Systems with Applications **37** (2010), no. 3, 2459–2464.

[101] Mohammed J Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, Wei Li, Paul Stolorz, and Ron Musick, *Parallel algorithms for discovery of association rules*, Scalable High Performance Computing for Knowledge Discovery and Data Mining, Springer, 1997, pp. 5–35.

[102] Jiayi Zh and Kun Ming Yu, *Tidset-based parallel fp-tree algorithm for the frequent pattern mining problem on pc clusters*, International Conference on Grid and Pervasive Computing, Springer, 2008, pp. 18–28.

[103] Jiayi Zhou and Kun-Ming Yu, *Balanced tidset-based parallel fp-tree algorithm for the frequent pattern mining on grid system*, Semantics, Knowledge and Grid, 2008. SKG'08. Fourth International Conference on, IEEE, 2008, pp. 103–108.

[104] Tao Zhou, L-L Jiang, R-Q Su, and Y-C Zhang, *Effect of initial configuration on network-based recommendation*, EPL (Europhysics Letters) **81** (2008), no. 5, 58004.

[105] Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang, *Predicting missing links via local information*, The European Physical Journal B-Condensed Matter and Complex Systems **71** (2009), no. 4, 623–630.