COMPUTATIONAL METHODS FOR LEARNING AND INFERENCE ON DYNAMIC NETWORKS

by

Kevin S. Xu

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering: Systems)
in The University of Michigan
2012

Doctoral Committee:

Professor Alfred O. Hero III, Chair
Professor George Michailidis
Professor Mark E. J. Newman
Assistant Professor Rajesh Rao Nadakuditi

For Elizabeth

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Professor Alfred Hero, for his guidance and mentorship. I have learned a tremendous amount about statistics, signal processing, and the research process from working as a research assistant in his group. My interactions with him have undoubtedly helped me develop and mature as a researcher. I also extend thanks to my other committee members, Professor George Michailidis, Professor Mark Newman, and Professor Rajesh Rao Nadakuditi, for their valuable input to this dissertation.

I am grateful to have worked alongside such a talented group of graduate students and postdoctoral fellows in Professor Hero's group. I would particularly like to acknowledge Dr. Mark Kliger; I had the pleasure of working with him on what became Chapters II–IV of this dissertation. I also thank Dr. Yilun Chen, Greg Newstadt, Dr. Arnau Tibau Puig, Sung Jin Hwang, Dr. Kumar Sricharan, Ko-Jen Hsiao, Zhaoshi Meng, Tzu-Yu Liu, Dr. Ami Wiesel, and Dr. Dennis Wei for the valuable assistance and discussions they have provided.

I am thankful for all of the encouragement and advice I have received from my professors during my undergraduate and graduate studies. I specifically thank Professor Ravi Mazumdar, Professor Sherman Shen, Professor Ella Atkins, Professor Sandeep Pradhan, and the late Professor Donald Grierson for their mentorship and support.

Chapter II of this dissertation would have not possible without the support of Matthew Prince, Eric Langheinrich, and Lee Holloway of Unspam Technologies Inc., who provided me with the Project Honey Pot data. Thanks also to Tianbao Yang for providing me with the source code for the probabilistic simulated annealing algorithm used in Chapter V.

I am grateful for all the assistance I received from my family over the years. I am blessed to have parents who encouraged me to pursue graduate studies. I am thankful for all the advice they have given me and for pushing me to be the best. I am also thankful to my brother Michael, who somehow seems to know the answer to everything.

Finally I would like to thank my fiancée Elizabeth. Her love and support and her wonderful sense of humor have helped me through difficult times.

My contributions in this dissertation have been supported in part by NSF grant CCF 0830490 and ONR grant N00014-08-1-1065. My work was also supported in part by an award from the Natural Sciences and Engineering Research Council of Canada.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ABSTRACT

Networks are ubiquitous in science, serving as a natural representation for many complex physical, biological, and social phenomena. Significant efforts have been dedicated to analyzing such network representations to reveal their structure and provide some insight towards the phenomena of interest. Computational methods for analyzing networks have typically been designed for static networks, which cannot capture the time-varying nature of many complex phenomena.

In this dissertation, I propose new computational methods for machine learning and statistical inference on *dynamic networks* with time-evolving structures. Specifically, I develop methods for visualization, tracking, and prediction of dynamic networks. The proposed methods take advantage of the dynamic nature of the network by intelligently combining observations at multiple time steps. This involves the development of novel *statistical models* and *state-space representations* of dynamic networks. Using the methods proposed in this dissertation, I identify long-term trends and structural changes in a variety of dynamic network data sets including a social network of spammers and a network of physical proximity among employees and students at a university campus.

CHAPTER I

INTRODUCTION

The study of networks has emerged as a topic of great interest in recent years. Many complex physical, biological, and social phenomena ranging from protein-protein interactions to the formation of social acquaintances can be naturally represented by networks. Much effort has been dedicated to analyzing real-world networks to reveal their often complex structure. Empirical findings from real networks can lead to significant insights towards the phenomena that are being investigated. Typically this is achieved through the creation of a model for the network that can replicate these findings. A well-known example consists of the small-world phenomenon, commonly referred to in the popular media as the "six degrees of separation." The small-world phenomenon was demonstrated experimentally by Milgram (1967), who found that the average number of intermediate acquaintances required to connect two randomly selected individuals is surprisingly small (around five or six in Milgram's experiment). The phenomenon was subsequently observed in other types of networks and can be replicated in synthetic networks generated using the small-world model developed by Watts and Strogatz (1998).

Until recently, most of the research on networks has focused on static networks. A static network can either represent a single time snapshot of the phenomenon being investigated or an aggregate view over time. However, most of the phenomena researchers are interested in investigating are time-varying. A static network representation of such a phenomenon is in many cases an oversimplification that is unable to capture important

features of the phenomenon such as its temporal dynamics. The natural extension is then to consider dynamic, time-evolving networks. There has been recent interest in analyzing dynamic networks, enabled by electronic means of collecting dynamic network data, such as smartphones, email, blogs, etc. Likewise, there is significant interest in modeling dynamic networks and making inferences from these models to help us understand and predict complex time-varying phenomena.

This dissertation contains my contributions to both of these areas, namely analyzing and modeling dynamic networks. I develop computational methods for several machine learning and statistical inference problems including visualization, clustering, tracking, and prediction. The common theme behind all of the proposed methods is the ability to take advantage of the dynamic nature of the network being examined by intelligently combining observations at multiple points in time and by modeling the temporal evolution of the network.

1.1 Scope

Contributions to network science have come from many branches of mathematics, sciences, and engineering, and many different types of networks have been analyzed, including biological, social, and information networks. The methods proposed in this dissertation are not specific to any type of network; however, the majority of the networks I examine are social and information networks. As a result, most of the intuition and explanations are also targeted to social and information networks.

The focus of this dissertation is on *evolving networks*, also sometimes referred to as temporally rewiring networks, rather than growing networks. To understand the difference between evolving and growing networks, consider two notions of *affinity* in a social network. The first notion of affinity between two individuals corresponds to some measure of communication between the individuals, such as the number of messages one sends to the other or the amount of time they spend together on the phone or in physical proximity of

each other. The second notion corresponds to one or both individuals indicating in some manner that they are friends or related in some other manner, whether it be via a survey or an electronic mechanism such as Facebook. Under the first notion, two individuals need to regularly communicate in some form to maintain a high affinity. Under the second notion, two individuals can maintain a high affinity as long as they do not remove their indication that they are friends with each other. The first notion of affinity leads to evolving networks, where affinities fluctuate both upward and downward over time whether or not new nodes join the network over time. The second notion leads to growing networks, where affinities very rarely decrease over time, so that the main dynamic component is the insertion of new nodes and edges over time. Leskovec (2008) provided a comprehensive treatment on analyzing and modeling growing networks; to the best of my knowledge, no such treatment is available for evolving networks. Both evolving and growing networks are referred to as "dynamic networks" in the literature. In the remainder of this dissertation, I use the term "dynamic networks" to refer solely to evolving networks unless otherwise specified.

All of the methods proposed in this dissertation deal with discrete-time dynamic networks; that is, a network is represented by a sequence of *snapshots* of the network topology at discrete *time steps*. In some cases, the snapshots are taken at the actual measurement times when the data were collected, and in others, the snapshots correspond to an aggregation of the measurements over a time step. The latter approach allows us to analyze continuous-time measurements as discrete-time dynamic networks.

1.2 Outline

In Chapter II I present a motivating application involving dynamic social networks of email spammers. Using data on spam harvesters and servers collected through email traps, I attempt to reveal spammers' social structure by identifying communities of spammers with high behavioral similarity. A *community*, also referred to as a *cluster*, is loosely defined as a group of nodes in a network with stronger ties to other nodes within the group than to nodes

outside the group. In this application, communities could correspond to organizations or gangs of spammers that collaborate for economic or other benefits. The analysis in this chapter is performed using computational methods for static networks, applied to a single graph snapshot at a time. Such an approach is suboptimal because it ignores information contained in other snapshots, which motivates the development of computational methods specific to dynamic networks.

Each of the following chapters addresses a specific problem relevant to learning and inference on dynamic networks. Each chapter is self-contained and includes a survey of existing literature on a particular problem involving dynamic networks along with a method I propose to solve the problem. I then compare the performance of my proposed method with other methods present in the literature on a variety of simulated and real networks.

Chapter III deals with the problem of visualizing dynamic networks. This problem is particularly challenging because it is difficult to represent the entire time sequence of network snapshots in a single 2-D visualization. The widely adopted approach for visualizing dynamic networks is to present an *animated* 2-D layout that evolves over time to reflect the changing topology of the network snapshots. The animation eases the transition between time steps so the viewer can identify the elements of the network that have changed and those that have not changed. In the graph drawing community, this is referred to as preserving the *mental map* (Misue et al., 1995). The mental map can be preserved by creating layouts that are stable over time, because a large number of node movements between layouts may cause a viewer to lose reference of the previous layout. My contribution is the creation of a framework for dynamic network visualization using *regularized graph layouts*, which encourages dynamic stability by adding two penalties to the cost function of a static graph layout algorithm.

Chapter IV discusses methods for tracking the temporal evolution of communities in dynamic networks. Many algorithms for community detection, also known as graph clustering, have been developed for static networks (Fortunato, 2010). In the dynamic network

setting where nodes and edges change over time, it is natural for community memberships to change over time as well. A class of clustering algorithms called *evolutionary clustering* algorithms have been created to target settings with dynamic data, including dynamic networks. The main advantage of evolutionary clustering algorithms over static clustering algorithms is their ability to detect long-term drifts in the data while being robust to short-term variations due to noise. This is typically accomplished by adding a temporal smoothness penalty to the cost function of a static clustering algorithm. The amount of temporal smoothness to enforce is determined a parameter commonly referred to as the *forgetting factor*. My contribution to this area is the creation of an evolutionary clustering framework that *adaptively* estimates the optimal forgetting factor in terms of minimizing mean-squared tracking error. The framework, which I call AFFECT, can be used both for clustering dynamic feature vectors and dynamic networks. A key element of AFFECT is the idea of a time-varying network state matrix that characterizes the long-term drifts of the network.

Chapter V builds on the idea of a network state by extending a parametric model for static networks, namely the stochastic blockmodel (SBM), to the dynamic network setting. The parameters naturally become time-varying states that characterize the evolving structure of the network. I propose a *state-space stochastic blockmodel* for dynamic networks along with an on-line inference procedure using the extended Kalman filter. By exploiting the block structure of the SBM, I show that one can perform near-optimal state tracking without requiring Monte Carlo methods, unlike other methods in the literature.

I conclude in Chapter VI with a summary of my contributions and a discussion of possible extensions and other open problems for future work.

1.3 List of publications

The following publications were produced based on the work presented in this dissertation.

Journal articles

1. K. S. Xu and A. O. Hero III. Revealing social networks of spammers. *The Next Wave*, **18**(3):26–34, 2010.

2. K. S. Xu, M. Kliger, and A. O. Hero III. Adaptive evolutionary clustering. *Submitted*, 2011.

3. K. S. Xu, M. Kliger, and A. O. Hero III. A regularized graph layout framework for dynamic network visualization. *Submitted*, 2011.

Articles in conference proceedings

1. K. S. Xu, M. Kliger, and A. O. Hero III. Visualizing the temporal evolution of dynamic networks. In *Proc. 9th Workshop on Mining and Learning with Graphs*, 2011.

2. K. S. Xu, M. Kliger, and A. O. Hero III. Tracking communities in dynamic social networks. In *Proc. 4th International Conference on Social Computing, Behavioral-Cultural Modeling, and Prediction*, 2011.

3. K. S. Xu, M. Kliger, and A. O. Hero III. A shrinkage approach to tracking dynamic networks. In *Proc. IEEE Workshop on Statistical Signal Processing*, 2011.

4. K. S. Xu, M. Kliger, and A. O. Hero III. Identifying spammers by their resource usage patterns. In *Proc. 7th Collaboration, Electronic Messaging, Anti-Abuse, and Spam Conference*, 2010.

5. K. S. Xu, M. Kliger, and A. O. Hero III. Evolutionary spectral clustering with adaptive forgetting factor. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2010.

6. K. S. Xu, M. Kliger, Y. Chen, P. J. Woolf, and A. O. Hero III. Revealing social networks of spammers through spectral clustering. In *Proc. IEEE International Conference on Communications*, 2009.

CHAPTER II

REVEALING SOCIAL NETWORKS OF SPAMMERS

I begin with a motivating application involving social networks of email spammers. Spam doesn't really need an introduction—anyone who owns an email address likely receives spam emails every day. Spammers face many challenges in order to deliver spam emails to their intended recipients. One such challenge is to evade spam filters, which are provided by many email services to separate spam from other emails. Furthermore, spam falls under a legal grey area, so spammers may try to hide their identities to make it difficult to identify and thus take legal action against them. Due to these challenges, one would expect spammers' behavior to be highly dynamic and constantly evolving to evade spam filters and avoid detection. This chapter presents an analysis of spammer behavior from a dynamic network perspective. However, the analysis in this chapter utilizes methods designed for static networks rather than dynamic networks. The limitations of these methods motivate the development of methods specifically designed for dynamic networks, which I present in the following chapters.

Previous studies on spam have mostly focused on studying its content or its source. Likewise, currently used anti-spam methods mostly involve filtering emails based on their content or by their email server IP address. More recently, there have been studies on the network-level behavior of spammers (Ramachandran and Feamster, 2006; Duan et al., 2007). Less attention has been devoted to studying how spammers acquire the email addresses that they send spam to, a process commonly referred to as harvesting. In this chap-

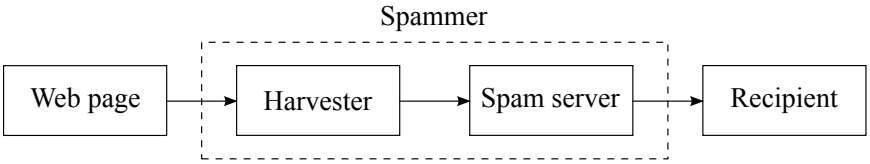


Figure 2.1: The path of spam: from an email address on a web page to a recipient's inbox.

ter I present a network-level study of spammers that incorporates data on both spamming and harvesting. Harvesting is the first phase of the spam cycle; sending the spam emails to the acquired addresses is the second phase.

Spammers send spam emails using spam servers, which are typically compromised computers or open proxies, both of which allow spammers to hide their identities. On the other hand, it has been observed that spammers do not make the same effort to conceal their identities during the harvesting phase (Prince et al., 2005), indicating that harvesters, which are individuals or bots that collect email addresses, are closely related to the spammers who are sending the spam emails. The harvester and spam server are the two intermediaries in the path of spam, illustrated in Figure 2.1.

In this chapter I try to reveal social networks of spammers by identifying communities of harvesters, which are closely related to communities of actual spammers, using data from both phases of the spam cycle. Such communities could correspond to spam organizations or gangs that operate in a collaborative manner for economic or other benefits. The source of the data analyzed in this chapter is Project Honey Pot (Unspam Technologies, Inc., 2012), a web-based network for monitoring harvesting and spamming activity by using trap email addresses. For every spam email received at a trap email address, the Project Honey Pot data set provides us with the IP address of the harvester that acquired the recipient's email address in addition to the IP address of the spam server, which is contained in the header of the email. Spammers make use of both harvesters and spam servers in order to distribute emails to recipients, but the IP address of the harvester that acquired the recipient's email address is typically unknown; it is only through Project Honey Pot that it is uncovered.

Project Honey Pot is a great source for studying phishing, a term commonly used to denote the process of attempting to fraudulently acquire sensitive information from a user by appearing to represent a trustworthy entity. I discuss Project Honey Pot and phishing in detail and summarize some related work in Section 2.1.

I look for community structure within the network of harvesters by partitioning harvesters into groups such that the harvesters in each group exhibit high similarity. This is a clustering problem, and I adopt a method commonly referred to as spectral clustering. Identifying community structure not only reveals groups of harvesters that have high similarity but also groups of spammers who may be socially connected, due to the close relation between harvesters and spammers. I provide an overview of spectral clustering in Section 2.2, and I discuss choices of similarity measures in Section 2.3.

My main findings are as follows:

1. *Most harvesters are either phishers or non-phishers (Section 2.1.2).* I find that most harvesters are either associated only with phishing emails or only with non-phishing emails.

2. *Phishers and non-phishers tend to separate into different communities when clustering based on similarity in spam server usage (Section 2.4.1).* That is, phishers tend to associate with other phishers, and non-phishers tend to associate with other non-phishers. In particular, phishers appear in small communities with strong ties, which suggests that they are sharing resources (spam servers) with other members of their community.

3. *Several groups of harvesters have coherent temporal behavior and similar IP addresses (Section 2.4.2).* In particular, I identify a group of ten harvesters associated with extremely large amounts of spam that have the same /24 IP address prefix, which happens to be owned by a rogue Internet service provider. This indicates that these harvesters are either the same spammer or a group of spammers operating from the same physical location.

These findings suggest that spammers do indeed form social networks, and I am able to identify meaningful communities.

2.1 Preliminaries

2.1.1 Project Honey Pot

Project Honey Pot is a system for monitoring harvesting and spamming activity via a network of decoy web pages with trap email addresses, known as *honey pots*, distributed all over the world by volunteers participating in the project. These trap addresses are embedded within the HTML source code of web pages and are invisible to human visitors. Spammers typically acquire email addresses by running harvesting bots that scan the HTML source of web pages and collect email addresses automatically. Spammers could also acquire addresses by manually browsing web sites and looking for them, although this is more time-consuming. Since the trap email addresses in the honey pots are invisible to human visitors, Project Honey Pot is trapping only the harvesting bots, and as a result, this is the only type of harvester investigated in this chapter.

Each time a harvester visits a honey pot, the centralized Project Honey Pot server generates a unique trap email address. The harvester's IP address is recorded and sent to the Project Honey Pot server. The email address embedded into each honey pot is unique, so a particular email address could only have been collected by the visitor to that particular honey pot. Thus, when an email is received at one of the trap addresses, exactly which harvester acquired the address is known. These email addresses are not published anywhere besides the honey pot, so it can safely be assumed that all emails received at these addresses are spam.

As of May 2012, over 83 million trap email addresses, 92 million spam servers, and $130,000$ harvesters have been identified by Project Honey Pot (Unspam Technologies, Inc., 2012). The number of emails received at the trap email addresses monitored by Project Honey Pot in 2006 and 2007 is shown by month in Figure 2.2. The number of emails have been normalized by the number of addresses collected to distinguish between growth of Project Honey Pot and an increase in spam volume. October 2006 is a month of particular

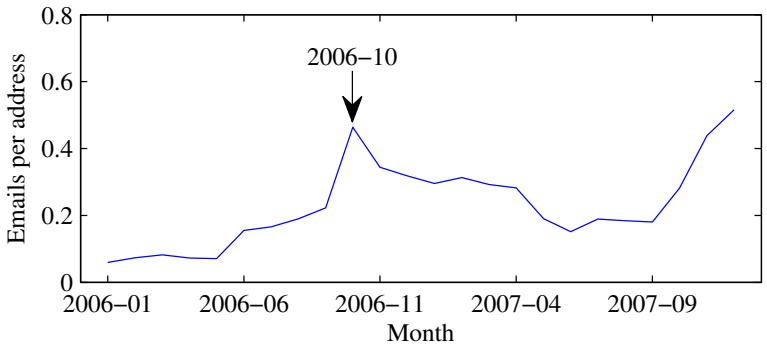


Figure 2.2: Number of emails received (per address collected) at trap addresses monitored by Project Honey Pot.

interest. Notice that the number of emails received in October 2006 increased significantly from September 2006 then came back down in November 2006. This observation agrees with media reports of a spam outbreak in October 2006 (Austin, 2006); thus I highlight results for this month in the remainder of the chapter. I refer readers to (Prince et al., 2005; Unspam Technologies, Inc., 2012) for additional details on Project Honey Pot.

In order to discover social networks of spammers, I need to associate emails to the spammers who sent them. Since I do not know the identity of the spammer who sent a particular email, I can associate the email either to the spam server that was used to send it or to the harvester that acquired the recipient's email address. A previous study using the Project Honey Pot data set has suggested that the harvester is more likely to be associated with the spammer than the spam server (Prince et al., 2005), so *I associate each email with the harvester that acquired the recipient's email address*. In particular, this is different from studies by Ramachandran and Feamster (2006) and Duan et al. (2007), which did not involve harvesters and implicitly associated emails with the spam servers that were used to send them. Note that I am not assuming that the harvesters are the spammers themselves. A harvester may collect email addresses for multiple spammers, or a spammer may use multiple harvesters to collect email addresses. To summarize, I am associating emails with harvesters and trying to discover communities of harvesters, which are closely related to communities of actual spammers.

2.1.2 Phishing

Project Honey Pot happens to be an ideal data source for studying phishing emails. It is impossible for a trap email address to, for example, sign up for a PayPal account, so all emails supposedly received from financial institutions can immediately be classified as phishing emails. Note that this is not possible with legitimate email addresses, which may receive legitimate emails from these sources. Since I know that any email mentioning such a source is a phishing email, I can classify each email as phishing or non-phishing based on its content. I classify an email as phishing if its subject contains a commonly used phishing word. The list of such words was built using common phishing words such as "password" and "account" and includes those found in a study on phishing (Chandrasekaran et al., 2006) and names of large financial institutions that do business on-line such as PayPal.

In general I find that a small percentage of the spam received through Project Honey Pot consists of phishing emails. In 2006 and 2007, 3.9% of the spam received was phishing spam. I define a phishing level for each harvester as the ratio of the number of phishing emails it is associated with to the total number of emails (both phishing and non-phishing) it is associated with. An interesting finding is that *most harvesters are associated only with phishing emails or only with non-phishing emails*. This can be seen in the histogram of harvester phishing levels in Figure 2.3. Specifically, 18% of harvesters have a phishing level of 0.95 or higher while 69% have a phishing level of 0.05 or lower. I label all harvesters as phishers or non-phishers based on their phishing level. I label a harvester as a phisher if its phishing level exceeds 0.5. In total, about 23% of harvesters are labeled as phishers. Note that phishers send less emails on a per-harvester basis than non-phishers, as only 3.9% of emails received were phishing emails as mentioned earlier. The labeling of harvesters as phishers or non-phishers will be used later when interpreting the clustering results.

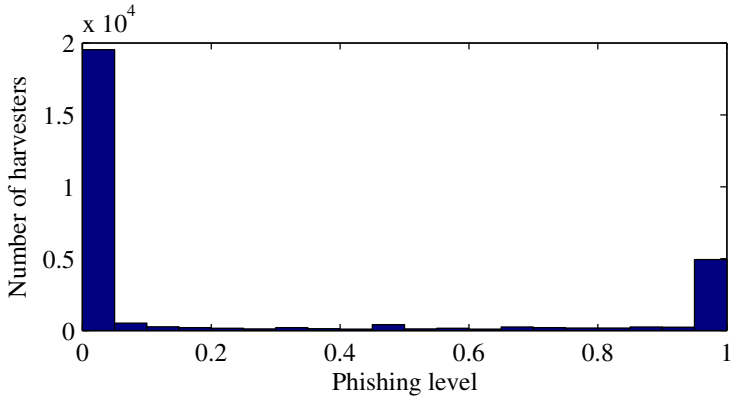


Figure 2.3: Histogram of harvester phishing levels. Most harvesters are associated only with phishing emails or only with non-phishing emails.

2.1.3 Related work

The majority of studies on spam have focused on studying its content in order to develop classification rules for separating spam from legitimate emails (Drucker et al., 1999; Androutsopoulos et al., 2000; Carreras and Marquez, 2001). Several other studies have examined spam sources and the effectiveness of source-based filtering approaches such as black listing (Jung and Sit, 2004; Ramachandran et al., 2006). More recently, there have been studies on behavioral characteristics of spammers. Ramachandran and Feamster (2006) and Duan et al. (2007) studied the network-level behavior of spammers but did not study harvesting. As a result, they implicitly associated emails with the spam servers that sent them. On the other hand, I associate emails with the harvesters that acquired the email addresses, which are closer entities to the spammers themselves than the spam servers.

Harvesting is an area of spam that has not been very well studied, to the best of our knowledge, likely due to the difficulty in obtaining harvesting data. There has been a previous study (Prince et al., 2005) on the Project Honey Pot data set. One of the main findings of that study was that spammers conceal their identity to a lesser degree when harvesting, which is one of the main assumptions made in this chapter. However, that study did not examine the behavioral similarities or social interactions between spammers. To the best of our knowledge, this is the first study that attempts to reveal the social structure

of spammers.

Finally, the problem of identifying communities in networks has been studied by researchers in many disciplines. Many methods for community detection have been proposed; a survey of these methods can be found in Fortunato (2010). Identifying communities in networks is intimately related to the problem of graph partitioning. Spectral clustering (Ng et al., 2001; Yu and Shi, 2003; von Luxburg, 2007) has emerged as a computationally efficient method for finding high quality graph partitions.

2.2 Overview of spectral clustering

In this study, I employ spectral clustering to identify groups of harvesters with high similarity. Spectral clustering is used over other clustering techniques because of its close relation to the graph partitioning problem of minimizing the normalized cut between clusters, which is a natural choice of objective function for community detection as discussed in (Leskovec et al., 2008) where it is referred to as conductance.

2.2.1 The graph partitioning problem

I represent the network of harvesters by a weighted undirected graph $G = (V, E, W)$ where V is the set of vertices, representing harvesters; E is the set of edges between vertices; and $W = [w_{ij}]_{i,j=1}^{M}$ is the matrix of edge weights with w_{ij} indicating the similarity between harvesters i and j. The choice of similarity measures is discussed in Section 2.3. W is known as the adjacency matrix of the graph and is also referred to in the literature as the similarity matrix or affinity matrix. $M = |V|$ is the total number of harvesters. The total weights of edges between two sets of vertices $A, B \subset V$ is defined by

$$\text{links}(A, B) = \sum_{i \in A} \sum_{j \in B} w_{ij}, \tag{2.1}$$

also referred to in the literature as the association between A and B. When A and B are disjoint, $\mathrm{links}(A, B)$ is usually referred to as the cut between A and B. The degree of a set A is defined by

$$\deg(A) = \mathrm{links}(A, V) \tag{2.2}$$

and is also referred to in the literature as the volume of A.

Our objective is to find highly similar groups of vertices in the graph, which represent harvesters that behave in a similar manner. This is a graph partitioning problem, and our objective translates into minimizing similarity between groups, maximizing similarity within groups, or preferably both. Let the groups be denoted by $V_1, V_2, \ldots, V_K$ where K denotes the number of groups to partition the graph into. I represent the graph partition by an M-by-K partition matrix X. Let $X = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K]$ where $x_{ij} = 1$ if harvester i is in cluster j and $x_{ij} = 0$ otherwise.

I adopt the normalized cut disassociation measure proposed in (Shi and Malik, 2000). The K-way normalized cut and normalized association are defined by

$$\mathrm{KNcut}(X) = \frac{1}{K} \sum_{i=1}^{K} \frac{\mathrm{links}(V_i, V \backslash V_i)}{\deg(V_i)}$$

$$\mathrm{KNassoc}(X) = \frac{1}{K} \sum_{i=1}^{K} \frac{\mathrm{links}(V_i, V_i)}{\deg(V_i)},$$

where the backslash operator denotes set difference. Minimizing KNcut also maximizes $\mathrm{KNassoc}$ because $\mathrm{KNcut}(X) + \mathrm{KNassoc}(X) = 1$, so one can minimize similarity between groups and maximize similarity within groups simultaneously. Thus the normalized cut appears to be a good choice of disassociation measure for our objective, which is to minimize KNcut, or equivalently, to maximize $\mathrm{KNassoc}$.

2.2.2 Finding a near-optimal solution

Unfortunately, maximizing $\mathrm{KNassoc}$ is an NP-complete problem even for $K = 2$ as noted in (Shi and Malik, 2000) so I turn to an approximate method. Define the degree matrix $D = \mathrm{diag}(W1_M)$ where $\mathrm{diag}(\cdot)$ creates a diagonal matrix from its vector argument, and 1_M is a vector of M ones. Rewrite links and deg as

$$\mathrm{links}(V_i, V_i) = \mathbf{x}_i^T W \mathbf{x}_i$$
$$\mathrm{deg}(V_i) = \mathbf{x}_i^T D \mathbf{x}_i.$$

The $\mathrm{KNassoc}$ maximization problem can then be written as follows:

$$\text{maximize} \quad \mathrm{KNassoc}(X) = \frac{1}{K} \sum_{i=1}^{K} \frac{\mathbf{x}_i^T W \mathbf{x}_i}{\mathbf{x}_i^T D \mathbf{x}_i} \tag{2.3}$$

$$\text{subject to} \quad X \in \{0,1\}^{M \times K} \tag{2.4}$$

$$X1_K = 1_M, \tag{2.5}$$

where constraints (2.4) and (2.5) force each vertex to be assigned to exactly one cluster. Let C be a diagonal matrix with $c_{ii} = \sum_{j=1}^{M} x_{ji}$, the number of vertices in cluster i. Let $Z = XC^{-1/2}$. (2.3)–(2.5) can be re-formulated as a trace maximization problem over Z (Yu and Shi, 2003). The re-formulated problem is given by

$$\text{maximize} \quad \frac{1}{K} \mathrm{tr}(Z^T \tilde{W} Z) \tag{2.6}$$

$$\text{subject to} \quad Z^T Z = I_K$$

$$ZC^{1/2} \in \{0,1\}^{M \times K}, \tag{2.7}$$

where $\tilde{W} = D^{-1/2} W D^{-1/2}$ is a normalized adjacency matrix, and $\mathrm{tr}(\cdot)$ denotes the trace of a matrix.

As mentioned earlier, maximizing (2.3), and thus (2.6), is NP-complete. A common

method for finding a near global-optimal solution known as *spectral clustering* first relaxes the problem into the continuous domain by dropping constraint (2.7), then discretizes the optimal continuous solution to obtain a near global-optimal discrete solution. By a generalized version of the Rayleigh-Ritz theorem (Lütkepohl, 1997), the optimal continuous solution is $Z = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_K]$, where $\mathbf{v}_i$ denotes the eigenvector corresponding to the ith largest eigenvalue of $\tilde{W}$. To obtain a feasible discrete solution, a common heuristic approach is to cluster the rows of a scaled version of Z using the K-means algorithm (Ng et al., 2001; von Luxburg, 2007). Alternatively, Yu and Shi (2003) propose an approach using orthogonal transformations and thresholding that is robust to random initialization and guarantees a near global-optimal solution. I opt for this approach and refer interested readers to Yu and Shi (2003) for more details.

2.2.3 Choosing the number of clusters

As with most clustering algorithms, the proper choice of K, the number of clusters, is unknown in spectral clustering. A useful heuristic particularly well-suited for choosing K in spectral clustering problems is the eigengap heuristic (von Luxburg, 2007). The goal is to choose K such that the highest eigenvalues $\lambda_1, \ldots, \lambda_K$ of the normalized adjacency matrix $\tilde{W}$ are very close to 1 but λ_{K+1} is relatively far away from 1.

One explanation for this heuristic is based on perturbation theory. Define a connected component to be a set of vertices A such that any two vertices in A can be joined by a path where all intermediate vertices on the path are in A, and there are no edges between between vertices in A and $V \backslash A$. Multiple connected components are disconnected from each other, meaning that there is no path between them. If a graph consists of K connected components, then the eigenvalue 1 has multiplicity K, and there is a gap to the $(K+1)$th eigenvalue (Chung, 1997). If the entries of the adjacency matrix of a graph with K connected components are slightly perturbed by introducing an edge between two connected components, then the eigenvalues of the normalized adjacency matrix are similarly per-

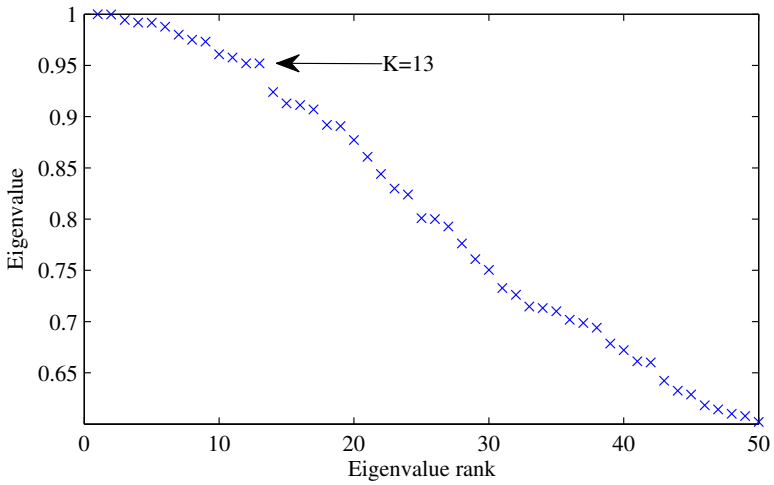


Figure 2.4: Top 50 eigenvalues of normalized adjacency matrix.

turbed by a small amount. Instead of seeing K eigenvalues with value 1, one would expect to see $K-1$ such eigenvalues, with the Kth eigenvalue close to 1, followed by a gap to the $(K+1)$th eigenvalue as before. Hence the eigengap heuristic assumes that the clusters in the graph correspond to slightly perturbed connected components and selects the number of clusters that best fits this criterion.

The top 50 eigenvalues of a normalized adjacency matrix $\tilde{W}$ are shown in Figure 2.4. Notice that there is a gap between the 13th and 14th eigenvalues; hence I would choose 13 clusters for this graph based on the eigengap heuristic.

2.3 Analysis methodology

A social network is a social structure composed of actors and ties, which indicate the relationships between actors. Direct relationships between harvesters (the actors) cannot be observed, so I use indirect relationships as the ties. I explore two types of ties; each type of tie corresponds to a similarity measure for choosing the edge weights w_{ij} between a pair of actors corresponding to vertices i and j. I often refer to these edge weights as behavioral similarities between harvesters, because the similarities are obtained by observations of harvesters' behavior over space or time.

Note that the network may evolve over time so I need to choose a time frame for analysis that is short enough so that I should be able to see this evolution if it is present yet long enough so that I have a large enough sample for the clustering results to be meaningful. There is no clear-cut method for choosing the time frame. As a starting point, I split the data set by month and analyze each month independently. This is not the optimal approach, but it allows the network to be analyzed using static clustering methods. I develop a better approach for clustering dynamic networks in Chapter IV.

2.3.1 Similarity measures

I examine two measures of behavioral similarity: similarity in spam server usage and temporal similarity. For both of these similarity measures, I create a coincidence matrix H, also referred to in the literature as a co-occurrence matrix, as an intermediate step to the creation of the adjacency matrix W, which is discussed in Section 2.3.2. The choice of similarity measure is crucial because it determines the topology of the graph. Each similarity measure provides a different view of the social network, so a poor choice of similarity measure may lead to detecting no community structure if harvesters are too similar or too dissimilar.

Similarity in spam server usage

Spammers typically send emails through multiple spam servers. Spam servers can be viewed as resources for spammers to deliver spam emails to their intended recipients. Thus common usage of resources, namely spam servers, is one way to establish similarity between harvesters. Consider a bipartite network of harvesters and spam servers described by the $M \times N$ coincidence matrix $H = [h_{ij}]_{i,j=1}^{M,N}$, where M is the number of harvesters and N is the number of spam servers. I choose $h_{ij} = p_{ij}/(d_j e_i) \in [0, 1]$ where

- p_{ij} denotes the number of emails associated with harvester i and spam server j;
- d_j denotes the total number of emails sent using spam server j;
- e_i denotes the total number of email addresses harvester i has acquired.

d_j is a normalization term that is included to account for the variation in the total number of emails sent through each spam server. For example, a harvester associated with 4 emails sent using a particular spam server that only sent 4 emails total should indicate a much stronger connection to that spam server than a harvester associated with 4 emails sent using a spam server that sent $1,000$ total. e_i is a normalization term to account for the variation in the number of email addresses each harvester has acquired, based on the assumption that harvesters send an equal amount of spam to each address they have acquired. One can interpret h_{ij} as *harvester i's percentage of usage of spam server j per address it has acquired*. The similarity between two harvesters i_1 and i_2 is taken to be the dot product between rows i_1 and i_2 of H.

Temporal similarity

Harvesters that exhibit high similarity in their temporal behavior may also indicate a social connection between spammers, so another possibility for linking harvesters is by their temporal spamming behavior. I look at the timestamps of all emails associated with a particular harvester and bin them into 1-hour intervals, resulting in a vector indicating how many emails a harvester is associated with in each interval. Doing this for all of the harvesters, I get another coincidence matrix H but with the columns representing time (in 1-hour intervals) rather than spam servers. The entries of H are $h_{ij} = s_{ij}/e_i$ where s_{ij} denotes the number of emails associated with harvester i in the jth time interval, and e_i is defined as before. Again I normalize by the number of email addresses acquired but no other normalizations are necessary because the columns represent time, which does not vary for different harvesters. The similarity between two harvesters i_1 and i_2 is the dot product between rows i_1 and i_2 of H as before.

2.3.2 Creating the adjacency matrix

From the coincidence matrix H I create an unnormalized matrix of pairwise similarities $S = HH^T$. I normalize S to form a normalized matrix of pairwise similarities $S' =$

$D_S^{-1/2} S D_S^{-1/2}$, where D_S is a diagonal matrix consisting of the diagonal elements of S. I can interpret this final normalization as a scaling of the edge weights between harvesters such that each harvester's self-edge has unit weight. This ensures that each harvester is equally important because there is no prior information on the importance of a particular harvester in the network.

I create an adjacency matrix W describing the graph by connecting the harvesters together according to their similarities in S'. There are several methods of connecting the graph, including k-nearest neighbors, ϵ-neighborhood, and the fully-connected graph. I opt for the k-nearest neighbor method, which translates into connecting vertices i and j if i is among the vertices with the k highest similarities to j or if j is among the vertices with the k highest similarities to i. This is the recommended choice in von Luxburg (2007) and is less vulnerable to poor choices of the connection parameters (in this case, the value of k). It also results in a sparse adjacency matrix, which speeds up computations and makes the graph easier to visualize. Unfortunately, there are not many guidelines on how to choose k. A heuristic suggested in von Luxburg (2007), motivated by asymptotic results, is to choose $k = \log M$. I use this choice of k as a starting point and increase k as necessary to avoid artificially disconnecting the graph.

2.4 Results

I present visualizations of the clustering results from October 2006, which is a month of particular interest as noted in Section 2.1.1. The visualizations are created using force-directed layout in Cytoscape (Shannon et al., 2003). Key statistics of the clustering results over a period of one year starting in July 2006 are presented in 3-month intervals in tables.

2.4.1 Similarity in spam server usage

The graph created using similarity in spam server usage usually consists of a giant connected component and many small connected components. The small components are eas-

ily recognized as clusters, while the large component is divided into multiple clusters. In Figure 2.5 I show the network of harvesters, connected using similarity in spam server usage, from October 2006. The shape and color of a harvester indicates the cluster it belongs to. The heuristics discussed in Section 2.2.3 suggest that the large connected component should be divided into 64 clusters, but to make the figure easier to interpret, I present a clustering result that divides the large component into only 7 clusters. I also remove connected components of less than ten harvesters. These modifications were made for visualization purposes only. In my analysis, and in particular when calculating the validation indices I present later, I use the number of clusters suggested by the heuristics discussed in Section 2.2.3 and include all small connected components.

Notice that the majority of harvesters belong in a large cluster with weak ties, which is a subset of the large component. This agrees with the observation in Leskovec et al. (2008) of a large network "core" that does not divide into clusters, which is found in many different networks from different domains. On the other hand, there exist several smaller clusters with strong ties, some of which are connected to the large cluster. Each cluster represents a community of harvesters associated with the same resources (spam servers), indicating that there is a strong likelihood that these harvesters correspond to spammers who are working together.

As with any clustering problem, the results need to be validated. If common usage of spam servers indeed indicates social connections between spammers, perhaps I can find some other property that is consistent within clusters. Recall from Section 2.1.2 that harvesters can be classified as either phishers or non-phishers. In Figure 2.6 I show the same network as in 2.5, except that it is colored by phishing level, as defined in Section 2.1.2, rather than cluster. Note that each of the clusters consists almost entirely of phishers or almost entirely of non-phishers. In particular, phishers appear to concentrate in small clusters with strong ties. This observation is further enhanced when clustering using 64 clusters as suggested by the eigengap heuristic discussed in Section 2.2.3. However, it is difficult to

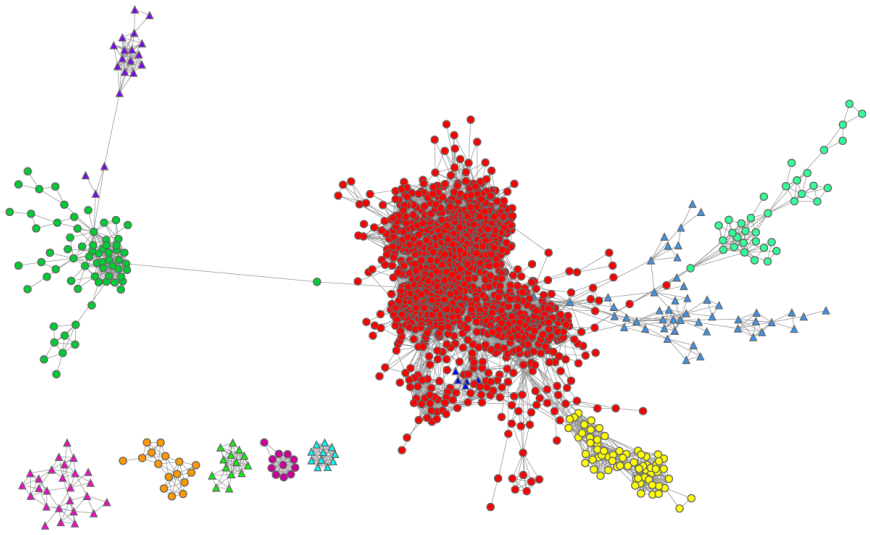


Figure 2.5: Network of harvesters formed by similarity in spam server usage in October 2006. The color and shape of a harvester indicate the cluster it belongs to.

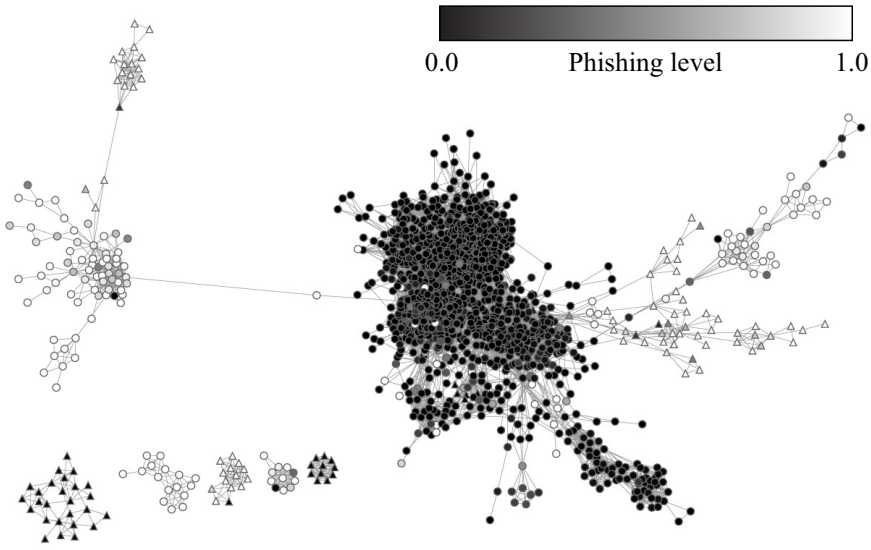


Figure 2.6: Alternate view of network pictured in Figure 2.5, where the color of a harvester corresponds to its phishing level.

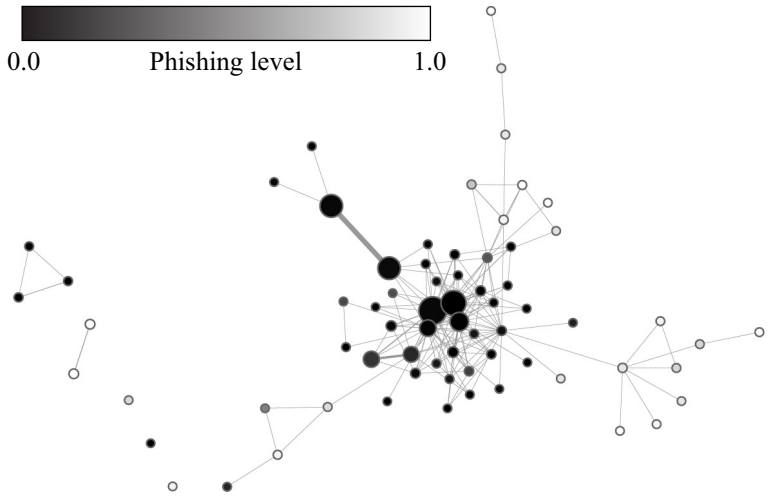


Figure 2.7: Network of clusters of harvesters formed by similarity in spam server usage in October 2006. The color and size of a cluster correspond to its phishing level and degree, respectively, and the width of an edge corresponds to the cut size between the two clusters it connects.

visualize the cluster membership of each vertex when the number of clusters is so large.

In Figure 2.7 I present a visualization of the network of clusters when the large connected component is divided into 64 clusters. Each node denotes a cluster rather than a harvester. The clusters are colored by their phishing level, which is the ratio of the number of phishing emails associated with all harvesters in the cluster to the total number of emails associated with all harvesters. The size of a node corresponds to its degree as defined in (2.2). Finally, the width of an edge corresponds to the cut size between the two clusters it connects, which is the links function between two clusters defined in (2.1). Notice that most clusters have either very high or very low phishing levels, much like I found with harvesters. In particular, the clusters with high phishing levels tend to be small and scattered far from the dense core of clusters with low phishing levels.

The five most common subject lines from a phishing and a non-phishing cluster are shown in Table 2.1. Notice the distinct separation between phishing subject lines and non-phishing subject lines. The subject headings were not provided to the clustering algorithm and are used only for validation and interpretation of the clustering results.

Next I quantify the observation that clusters tend to divide into phishing and non-

Phishing cluster	Non-phishing cluster
Password Change Required	Make Money by Sharing Your Life with Friends and F
Question from eBay Member	Premiere Professional & Executive Registries Invit
Credit Union Online (R) $50 Reward Survey	Texas Land/Golf is the Buzz
PayPal Account	Keys to Stock Market Success
PayPal Account - Suspicious Activity	An Entire Case of Fine Wine plus Exclusive Gift to

Table 2.1: Most common subject lines from a phishing and non-phishing cluster (truncated to 50 characters by Project Honey Pot database).

phishing clusters by examining the distribution of phishers and non-phishers in clusters. I consider a cluster as a phishing cluster if it contains more phishers than non-phishers and as a non-phishing cluster otherwise. Using phisher or non-phisher as a label for each harvester, I compute the Rand index (Rand, 1971) and adjusted Rand index (Hubert and Arabie, 1985), both commonly used indices used for clustering validation. The Rand index is a measure of agreement between clustering results and a set of class labels and is given by

$$\text{Rand index} = \frac{a+d}{a+b+c+d} \tag{2.8}$$

where

- a is the number of pairs of vertices with the same label and in the same cluster;
- b is the number of pairs with the same label but in different clusters;
- c is the number of pairs with different labels but in the same cluster;
- d is the number of pairs with different labels and in different clusters.

A Rand index of 0 indicates complete disagreement between clusters and labels, and a Rand index of 1 indicates perfect agreement. The adjusted Rand index is corrected for chance so that the expected adjusted Rand index for a random clustering result is 0.

In this clustering problem, the Rand index indicates how well phishers and non-phishers divide into phishing and non-phishing clusters, respectively. The adjusted Rand index in-

Year	2006		2007		
Month	July	October	January	April	July
Rand index	0.908	0.952	0.941	0.962	0.895
Adj. Rand index	0.782	0.865	0.791	0.799	0.636

Table 2.2: Validation indices for clustering results by month.

dicates how well phishers and non-phishers divide compared to the expected division that a random clustering algorithm would produce. Both indices are shown in Table 2.2 for five months. Note that the clustering results have excellent agreement with the labels, and the agreement is much higher than expected by chance. The division between phishers and non-phishers is not perfect, as there are some phishers belonging in non-phishing clusters and vice-versa, but the high adjusted Rand index indicates that this split is highly unlikely to be due to chance alone. Hence I have found empirical evidence that phishers tend to form small communities with strong ties, suggesting that they share resources (spam servers) between members of their community.

2.4.2 Temporal similarity

Unlike the graph created by similarity in spam server usage, the graph created by temporal similarity usually consists of just a single connected component. In Figure 2.8 I show the network of harvesters, connected using temporal similarity, from October 2006, where again the shape and color of a harvester indicates the cluster it belongs to. Again, I present a clustering result with fewer clusters than suggested by the heuristics discussed in Section 2.2.3 for ease of visualization. Any similarity in color with Figure 2.5 is coincidental; Figure 2.8 represents a completely different view of the social network and provides different insights.

Unfortunately I do not have validation for this clustering result on a global scale like I did with phishing level for similarity in spam server usage. However by looking at temporal spamming plots of the small clusters, I find some local validation. Namely, I see groups of harvesters in the same cluster with extremely coherent temporal spamming behavior. I

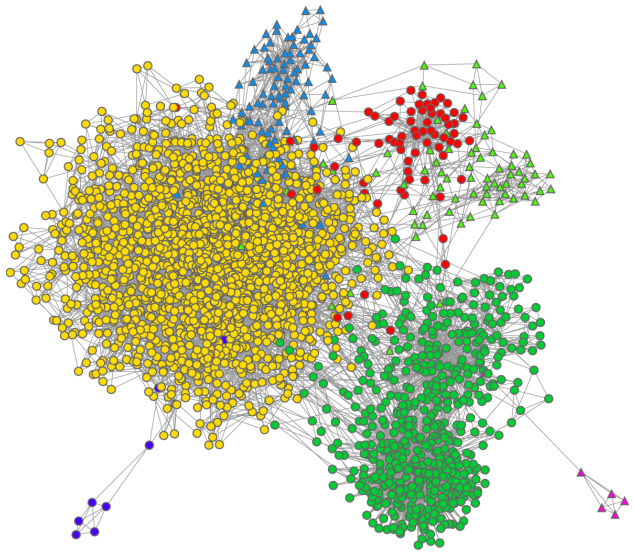


Figure 2.8: Network of harvesters formed by temporal similarity in October 2006. The color and shape of a harvester indicate the cluster it belongs to.

notice that in several of these groups, the harvesters also have similar IP addresses. In particular, I notice a group of ten harvesters that have extremely coherent temporal spamming patterns and have the same /24 IP address prefix, namely 208.66.195/24, indicating that they are also in the same physical location. In Figure 2.8 they can be found in the light green cluster of triangular nodes at the top right of the network. Their temporal spamming plots from October 2006 are shown in Figure 2.9, where the horizontal axis corresponds to time (in days), and vertical axis corresponds to the number of emails associated with each harvester.

Upon further investigation, I find that their IP addresses are in the 208.66.192/22 prefix owned by McColo Corp., a known rogue Internet service provider that acted as a gateway to spammers and was finally removed from the Internet in November 2008 (Nazario, 2008). This serves as further confirmation that these harvesters correspond to spammers that are likely to be socially connected. They first appeared at the end of May 2006 and have been among the heaviest harvesters, in terms of the number of emails sent, in every month since then. The average correlation coefficients ρ_{avg} between two harvesters in this group are listed in Table 2.3 for five months. Notice that their average correlation coefficients

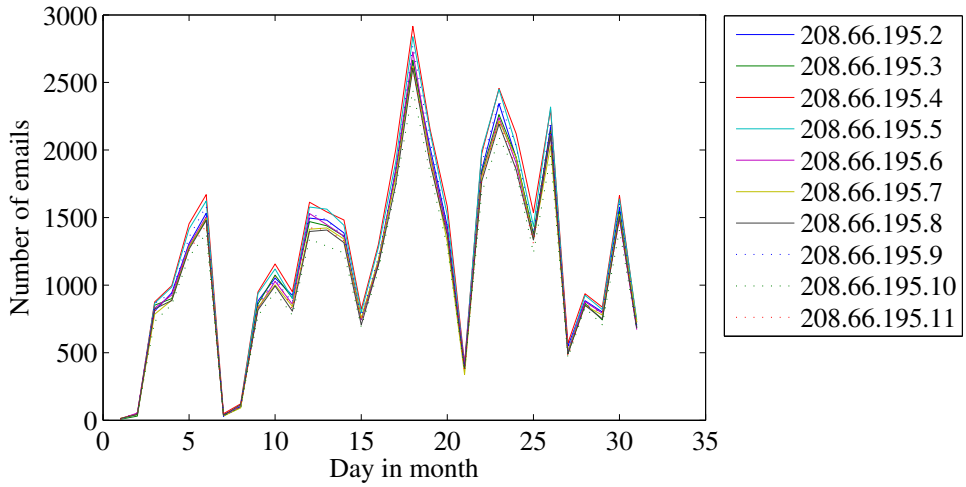


Figure 2.9: Temporal spamming plots of 208.66.195/24 group of ten harvesters.

Year	2006		2007		
Month	July	October	January	April	July
ρ_{avg}	0.980	0.988	0.950	0.949	0.935

Table 2.3: Average temporal correlation coefficients of 208.66.195/24 group of ten harvesters by month.

are extremely high and strongly suggest that these spammers are working together in a coordinated matter. Also note that their behavior is still highly correlated more than a year after they first appeared. Furthermore, I discover that they have high temporal correlation in the harvesting phase; that is, they collect email addresses in a very similar manner as well. I would certainly expect them to belong to the same cluster, which agrees with the clustering results. Hence I believe that this group is either the same spammer or a community of spammers operating from the same physical location.

2.5 Summary

In this chapter, I revealed social networks of spammers by discovering communities of harvesters from the data collected through Project Honey Pot. Specifically, I clustered harvesters using two similarity measures reflecting their behavioral correlations. In addition, I studied the distribution of phishing emails among harvesters and among clusters. I found

that harvesters are typically either phishers or non-phishers; that is, they are either associated only with phishing emails or only with non-phishing emails. Moreover, I discovered that communities of harvesters divide into communities of mostly phishers and mostly non-phishers when clustering according to similarity in spam server usage. In particular, I observed that phishers tend to form small communities with strong ties. I also discovered several groups of harvesters with extremely coherent temporal behavior and very similar IP addresses, indicating that these groups are also geographically close.

Note that the two similarity measures I studied provided us with different views of the social networks of spammers, and I gained useful insights from both of them. All of my findings are empirical; however, I believe that they reveal some previously unknown behavior of spammers, namely that spammers do indeed form social networks. Since harvesters are closely related to spammers, the discovered communities of harvesters are closely related to communities of spammers. Identifying communities of spammers could allow us to fight spam from a new perspective by incorporating the social structure of spammers.

A limitation of the analysis presented in this chapter is that it utilizes methodology designed for static networks, namely static spectral clustering. To get a better idea for the temporal evolution in this dynamic network, it would be better to apply methods that are specifically designed for dynamic networks. In the following chapters, I develop such methods, targeted at the problems of visualizing, clustering, tracking, and predicting dynamic networks.

CHAPTER III

VISUALIZING DYNAMIC NETWORKS

How to visually represent a network is one of the earliest questions posed by network analysts. Visualization is an important tool that can provide insights and intuition about networks that cannot be conveyed by summary statistics alone. The task of visualizing dynamic networks has been an open problem that has attracted attention from sociologists (Moody et al., 2005; Bender-deMoll and McFarland, 2006; Leydesdorff and Schank, 2008) and the information visualization community (Brandes and Corman, 2003; Erten et al., 2004; Frishman and Tal, 2008) among others.

Visualizing static networks is a challenge in itself. Static networks are typically represented by graphs, which have no natural representation in a Euclidean space. Many graph layout algorithms have been developed to create aesthetically pleasing 2-D representations of graphs (Di Battista et al., 1999; Herman et al., 2000). Common layout methods for general graphs include force-directed layout (Kamada and Kawai, 1989; Fruchterman and Reingold, 1991), multidimensional scaling (MDS) (Gansner et al., 2004; Borg and Groenen, 2005) and graph Laplacian layout (GLL), also known as spectral layout (Hall, 1970; Koren, 2003).

Dynamic networks are typically represented by a time-indexed sequence of graph snapshots; thus visualizing dynamic networks in 2-D presents an additional challenge due to the temporal aspect. If one axis is used to represent time, then only a single axis remains to convey the topology of the network. While it is possible to identify certain trends from a

1-D time plot created in this manner, it is a poor representation of the network topology. Furthermore, it is difficult to display edges between nodes in a 1-D layout without a significant amount of overlap. Brandes and Corman (2003) presented a possible solution to this problem by creating a pseudo-3-D visualization that treats 2-D layouts of each snapshot as layers in a stack. Unfortunately, the resulting visualization is difficult to interpret. The more conventional approach is to present an animated 2-D layout that evolves over time to reflect the current snapshot (Erten et al., 2004; Moody et al., 2005; Bender-deMoll and McFarland, 2006; Frishman and Tal, 2008; Leydesdorff and Schank, 2008). A challenge with this approach is to preserve the *mental map* (Misue et al., 1995) between graph snapshots so that the transition between frames in the animation can be easily interpreted by a human viewer. In particular, it is undesirable for a large number of nodes to drastically change positions between frames, which may cause the viewer to lose reference of the previous layout.

Some of the early work on dynamic network visualization simply involved creating interpolated transition layouts (Moody et al., 2005; Bender-deMoll and McFarland, 2006). While interpolation does make an animation more aesthetically pleasing, it does not constrain the successive layouts in any way to make them more interpretable. In many real networks, individual snapshots have high variance, so creating a layout for each snapshot using a static graph layout method could result in large node movements between time steps. Often, this is not due to a failure of the static graph layout algorithm but simply a consequence of the cost function it is attempting to optimize, which does not consider any other snapshots. When dealing with dynamic networks, better performance can be obtained by using *regularized* methods that consider the historical snapshots in addition to the current snapshot. Such an approach has been used to develop regularized clustering algorithms for dynamic networks, also known as evolutionary clustering algorithms, which will be discussed in detail in Chapter IV. In the context of dynamic network visualization, regularization encourages layouts to be stable over time, thus preserving the mental map

between snapshots. The concept of regularization has been employed in many problems in statistics and machine learning, including ridge regression (Hoerl and Kennard, 1970), also known as Tikhonov regularization, the LASSO (Tibshirani, 1996), and penalized matrix decomposition (Witten et al., 2009). It is often used to introduce additional information or constraints and to give preference to solutions with certain desirable properties such as sparsity, smoothness, or in this chapter, dynamic stability in order to preserve the mental map.

I introduce a framework for dynamic network visualization using *regularized graph layouts*. The framework is designed to generate layouts in the *on-line* setting where only present and past snapshots are available, although it can also be used in the off-line setting. It involves optimizing a modified cost function that augments the cost function of a static graph layout algorithm with two penalties:

1. A *grouping penalty*, which discourages nodes from deviating too far from other nodes belonging to the same group;
2. A *temporal penalty*, which discourages nodes from deviating too far from their previous positions.

Groups could correspond to a priori knowledge, such as participant affiliations in social networks. If no a priori group knowledge is available, groups can be learned from the network using, for example, the evolutionary clustering algorithms in Chapter IV. The grouping penalty keeps group members close together in the sequence of layouts, which helps to preserve the mental map because nodes belonging to the same group often evolve over time in a similar fashion. The temporal penalty helps to preserve the mental map by discouraging large node movements that may cause a human to lose reference of previous node positions, such as multiple nodes moving unnecessarily from one side of the layout to the opposite side. Using the proposed framework, I develop two dynamic layout algorithms, *dynamic multidimensional scaling* (DMDS) and *dynamic graph Laplacian layout* (DGLL), that are regularized versions of MDS and GLL, respectively.

To the best of my knowledge, this is the first work that presents a general regularization

framework for dynamic network visualization, whereas previously proposed methods have been specific to a particular static graph layout method, typically force-directed layout. The algorithms DMDS and DGLL are also unique in that they add both grouping and temporal regularization to MDS and GLL, respectively. Existing methods for temporal regularization in MDS (Baur and Schank, 2008) and grouping regularization in GLL (Costa and Hero III, 2005) are subsumed by DMDS and DGLL, respectively. The methods for grouping regularization in MDS and temporal regularization in GLL used in this chapter are novel. I apply the proposed algorithms on a selection of dynamic network data sets to demonstrate the importance of both grouping and temporal regularization in creating interpretable visualizations.

3.1 Background

I begin by specifying the notation I shall use in this chapter. Time-indexed quantities are indicated using square brackets, e.g. $X[t]$. I represent a dynamic network by a discrete-time sequence of graph snapshots. Each snapshot is represented by a graph adjacency matrix $W[t]$ where $w_{ij}[t]$ denotes the weight of the edge between nodes i and j at time t (chosen to be 1 for unweighted graphs), and $w_{ij}[t] = 0$ if no edge is present. I assume all graphs are undirected, so that $w_{ij}[t] = w_{ji}[t]$. For simplicity of notation, I typically drop the time index for all quantities at time step t, i.e. W is assumed to denote $W[t]$.

I refer to a graph layout by a matrix $X \in \mathbb{R}^{n \times s}$, where n denotes the number of nodes present at time t, and each row $\mathbf{x}_{(i)}$ corresponds to the s-dimensional position of node i. One is usually interested in 2-D visualization ($s = 2$), although the proposed methods can also be applied to other values of s. The ith column of X is denoted by $\mathbf{x}_i$, and the individual entries by x_{ij}. The superscript in $\mathbf{x}_i^{(h)}$ is used to denote the value of $\mathbf{x}_i$ at iteration h of an algorithm. The norm operator $\|\cdot\|$ refers to the l_2-norm, and $\mathrm{tr}(\cdot)$ denotes the matrix trace operator. I denote the all ones column vector by $\mathbf{1}$.

I now summarize the static graph layout methods of multidimensional scaling and graph

Laplacian layout, which operate on a single graph snapshot. I develop regularized versions of these methods for dynamic networks in Section 3.2.

3.1.1 Multidimensional scaling

Multidimensional scaling (MDS) is a family of statistical methods that aim to find an optimal layout $X \in \mathbb{R}^{n \times s}$ such that the distance between $\mathbf{x}_{(i)}$ and $\mathbf{x}_{(j)}$ for all $i \neq j$ is as close as possible to a desired distance δ_{ij}. There are a variety of different cost functions and associated algorithms for MDS; I refer interested readers to Borg and Groenen (2005). Here I describe the cost function known as stress and its associated majorization algorithm. The stress of a layout X is given by

$$\text{stress}(X) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} v_{ij} \left(\delta_{ij} - \|\mathbf{x}_{(i)} - \mathbf{x}_{(j)}\| \right)^2, \tag{3.1}$$

where v_{ij} denotes the weight or importance of maintaining the desired distance δ_{ij}. I refer to the matrix $V = [v_{ij}]$ as the *MDS weight* matrix to avoid confusion with the graph adjacency matrix W, which is also sometimes referred to as a weight matrix.

In order to use stress MDS for graph layout, the graph adjacency matrix W is first converted into a desired distance matrix $\Delta = [\delta_{ij}]$. This is done by defining a distance metric over the graph and calculating distances between all pairs of nodes. The distance between two nodes i and j is typically taken to be the length of the shortest path between the nodes (Gansner et al., 2004). For weighted graphs, it is assumed that the edge weights denote dissimilarities; if the edge weights instead denote similarities, they must first be converted into dissimilarities before computing Δ.

The MDS weights v_{ij} play a crucial role in the aesthetics of the layout. The commonly used Kamada-Kawai (KK) force-directed layout (Kamada and Kawai, 1989) is a special case of stress MDS where the weights are chosen to be $v_{ij} = \delta_{ij}^{-2}$ for $i \neq j$ and $v_{ii} = 0$.

The objective of stress MDS is to find a layout X that minimizes (3.1). (3.1) can be

decomposed into

$$\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} v_{ij}\delta_{ij}^2 + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} v_{ij}\|\mathbf{x}_{(i)} - \mathbf{x}_{(j)}\|^2 - \sum_{i=1}^{n}\sum_{j=1}^{n} v_{ij}\delta_{ij}\|\mathbf{x}_{(i)} - \mathbf{x}_{(j)}\|. \tag{3.2}$$

Note that the first term of (3.2) is independent of X. The second term of (3.2) can be written as $\mathrm{tr}(X^T R X)$ where the $n \times n$ matrix R is given by

$$r_{ij} = \begin{cases} -v_{ij} & i \neq j, \\ \sum_{k \neq i} v_{ik} & i = j. \end{cases} \tag{3.3}$$

$\mathrm{tr}(X^T R X)$ is quadratic and convex in X and is easily optimized.

The third term of (3.2) cannot be written as a quadratic form. However, it can be optimized by an iterative majorization method known as "scaling by majorizing a complicated function" (SMACOF) (Borg and Groenen, 2005). This non-quadratic term is iteratively majorized, and the resulting upper bound for the stress is then optimized by differentiation. For a matrix $Z \in \mathbb{R}^{n \times s}$, define the matrix-valued function $S(Z)$ by

$$s_{ij}(Z) = \begin{cases} -v_{ij}\delta_{ij}/\|\mathbf{z}_{(i)} - \mathbf{z}_{(j)}\| & i \neq j, \\ -\sum_{k \neq i} s_{ik}(Z) & i = j. \end{cases} \tag{3.4}$$

Then, it is shown in (Gansner et al., 2004; Borg and Groenen, 2005) that

$$-\mathrm{tr}(X^T S(Z) Z) \geq -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} v_{ij}\delta_{ij}\|\mathbf{x}_{(i)} - \mathbf{x}_{(j)}\|$$

so that an upper bound for the stress is

$$\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} v_{ij}\delta_{ij}^2 + \mathrm{tr}(X^T R X) - 2\,\mathrm{tr}(X^T S(Z) Z). \tag{3.5}$$

By setting the derivative of (3.5) with respect to X to 0, the minimizer of the upper bound

is found to be the solution to the equation $RX = S(Z)Z$.

The algorithm for optimizing stress is iterative. Let $X^{(0)}$ denote an initial layout. Then at each iteration h, solve

$$R\,\mathbf{x}_a^{(h)} = S\left(X^{(h-1)}\right)\mathbf{x}_a^{(h-1)} \tag{3.6}$$

for $\mathbf{x}_a^{(h)}$ for each $a = 1, \ldots, s$. (3.6) can be solved using a standard linear equation solver. Note that R is rank-deficient; this is a consequence of the stress function being translation-invariant (Gansner et al., 2004). The translation-invariance can be removed by fixing the location of one point, e.g. by setting $\mathbf{x}_{(1)} = 0$, removing the first row and column of R, and removing the first row of $S\left(X^{(h-1)}\right)X^{(h-1)}$ (Gansner et al., 2004). (3.6) can then be solved efficiently using Cholesky factorization. The iteration can be terminated when

$$\frac{\text{stress}\left(X^{(h-1)}\right) - \text{stress}\left(X^{(h)}\right)}{\text{stress}\left(X^{(h-1)}\right)} < \epsilon, \tag{3.7}$$

where ϵ is the convergence tolerance of the iterative process.

3.1.2 Graph Laplacian layout

Graph Laplacian layout (GLL) methods optimize a quadratic function associated with the graph Laplacian matrix (Koren, 2003), which I call the GLL energy. The graph Laplacian is obtained from the adjacency matrix by $L = D - W$, where D is the diagonal matrix of node degrees defined by $d_{ii} = \sum_{j=1}^{n} w_{ij}$. For weighted graphs, GLL assumes that the weights correspond to similarities between nodes, rather than dissimilarities as in MDS. GLL is also referred to as "spectral layout" because the optimal solution involves the eigenvectors of the Laplacian, as I will show. The GLL energy function is defined by

$$\text{energy}(X) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\|\mathbf{x}_{(i)} - \mathbf{x}_{(j)}\|^2. \tag{3.8}$$

It is easily shown that $\text{energy}(X) = \text{tr}(X^T L X)$. The GLL problem can be formulated as (Hall, 1970; Koren, 2003):

$$\min_X \quad \text{tr}(X^T L X) \tag{3.9}$$

$$\text{subject to} \quad X^T X = nI \tag{3.10}$$

$$X^T \mathbf{1} = \mathbf{0}. \tag{3.11}$$

From (3.8), it can be seen that minimizing the GLL energy function aims to make edge lengths short by placing nodes connected by heavy edges close together. (3.11) removes the trivial solution $\mathbf{x}_a = \mathbf{1}$, which places all nodes at the same location in one dimension. It can also be viewed as removing a degree of freedom in the layout due to translation invariance (Belkin and Niyogi, 2003) by setting the mean of $\mathbf{x}_a$ to 0 for all a. Since $\mathbf{x}_a$ has zero-mean, $(\mathbf{x}_a^T \mathbf{x}_a)/n$ corresponds to the variance or scatter of the layout in dimension a. Thus (3.10) constrains the layout to have unit variance in each dimension and zero covariance between dimensions so that each dimension of the layout provides as much additional information as possible. Moreover, one can see that (3.10) differs slightly from the usual constraint $X^T X = I$ (Belkin and Niyogi, 2003; Koren, 2003), which constrains the layout to have variance $1/n$ in each dimension. In the dynamic network setting where n can vary over time, this is undesirable because the layout would change scale between time steps if the number of nodes changes.

By a generalization of the Rayleigh-Ritz theorem (Lütkepohl, 1997), an optimal solution to the GLL problem is given by $X^* = \sqrt{n}[\mathbf{v}_2, \mathbf{v}_3, \ldots, \mathbf{v}_{s+1}]$, where $\mathbf{v}_i$ denotes the eigenvector corresponding to the ith smallest eigenvalue of L. Note that $\mathbf{v}_1 = (1/\sqrt{n})\mathbf{1}$ is excluded because it violates the zero-mean constraint (3.11). Using the property that $\text{tr}(ABC) = \text{tr}(CAB)$, the cost function (3.9) is easily shown to be invariant to rotation and reflection, so $X^* R$ is also an optimal solution for any $R^T R = I$.

In practice, it has been found that using degree-normalized eigenvectors often results in

more aesthetically pleasing layouts (Belkin and Niyogi, 2003; Koren, 2003). The degree-normalized layout problem differs only in that the dot product in each of the constraints is replaced with the degree-weighted dot product, resulting in the following optimization problem:

$$\begin{aligned} \min_{X} \quad & \operatorname{tr}(X^T L X) \\ \text{subject to} \quad & \operatorname{tr}(X^T D X) = \operatorname{tr}(D) I \\ & X^T D \mathbf{1} = \mathbf{0}. \end{aligned}$$

The optimal solution is given by $X^* = \sqrt{\operatorname{tr}(D)}\,[\mathbf{u}_2, \mathbf{u}_3, \ldots, \mathbf{u}_{s+1}]$ or any rotation or reflection of X^*, where $\mathbf{u}_i$ denotes the generalized eigenvector corresponding to the ith smallest generalized eigenvalue of (L, D). Again, $\mathbf{u}_1 = \left(1/\sqrt{\operatorname{tr}(D)}\right)\mathbf{1}$ is excluded because it violates the zero-mean constraint. A discussion on the merits of the degree normalization can be found in (Koren, 2003).

3.2 Regularized layout methods

3.2.1 Regularization framework

The aforementioned static layout methods can be applied snapshot-by-snapshot to create a visualization of a dynamic network; however, the resulting visualization is often difficult to interpret due to the lack of regularization. I propose a regularized layout framework that uses a *modified cost* function, defined by

$$\mathcal{C}_{\text{modified}} = \mathcal{C}_{\text{static}} + \alpha \mathcal{C}_{\text{grouping}} + \beta \mathcal{C}_{\text{temporal}}.$$

The static cost $\mathcal{C}_{\text{static}}$ corresponds to the cost function optimized by the static layout algorithm. For example, in MDS, it is the stress function defined in (3.1), and in GLL, it is the energy defined in (3.8). The grouping cost $\mathcal{C}_{\text{grouping}}$ is chosen to discourage nodes from deviating too far from other group members; α controls the importance of the grouping

cost, so I refer to $\alpha\mathcal{C}_{\text{grouping}}$ as the grouping penalty. Similarly, the temporal cost $\mathcal{C}_{\text{temporal}}$ is chosen to discourage nodes from deviating too far from their previous positions; β controls the importance of the temporal cost, so I refer to $\beta\mathcal{C}_{\text{temporal}}$ as the temporal penalty. I propose quadratic forms for these penalties, similar to ridge regression (Hoerl and Kennard, 1970).

Let k denote the number of groups. Define the group membership by an $n \times k$ matrix C where

$$c_{il} = \begin{cases} 1 & \text{node } i \text{ is in group } l \text{ at time step } t, \\ 0 & \text{otherwise.} \end{cases} \tag{3.12}$$

I introduce grouping regularization by adding group representatives, which also get mapped to an s-dimensional position, stored in the matrix $Y \in \mathbb{R}^{k \times s}$. The proposed grouping cost is given by

$$\mathcal{C}_{\text{grouping}}(X, Y) = \sum_{l=1}^{k} \sum_{i=1}^{n} c_{il} \|\mathbf{x}_{(i)} - \mathbf{y}_{(l)}\|^2, \tag{3.13}$$

where $\mathbf{y}_{(l)}$ denotes the position of the lth representative. Notice that the grouping cost is decreased by moving $\mathbf{y}_{(l)}$ and $\mathbf{x}_{(i)}$ towards each other if node i is in group l. Notice also that I do not require knowledge of the group membership of every node. Nodes with unknown group memberships correspond to all-zero rows in C and are not subject to any grouping penalty.

I introduce temporal regularization on nodes present at both time steps t and $t-1$ by discouraging node positions from deviating significantly from their previous positions. Define the diagonal matrix E by

$$e_{ii} = \begin{cases} 1 & \text{node } i \text{ was present at time step } t-1, \\ 0 & \text{otherwise.} \end{cases} \tag{3.14}$$

The proposed temporal cost is then given by

$$\mathcal{C}_{\text{temporal}}(X, X[t-1]) = \sum_{i=1}^{n} e_{ii} \|\mathbf{x}_{(i)} - \mathbf{x}_{(i)}[t-1]\|^2. \tag{3.15}$$

The temporal cost is decreased by moving $\mathbf{x}_{(i)}$ towards $\mathbf{x}_{(i)}[t-1]$, but unlike in the grouping cost, $\mathbf{x}_{(i)}[t-1]$ is fixed because it was assigned at the previous time step. Thus the previous node position acts as an anchor for the current node position.

Next I demonstrate how the grouping and temporal penalties can be introduced into MDS and GLL as examples of the proposed regularization framework.

3.2.2 Dynamic multidimensional scaling

The dynamic multidimensional scaling (DMDS) modified cost is given by the modified stress function

$$\begin{aligned} \text{Mstress}(X, Y) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} v_{ij} \big(\delta_{ij} - \|\mathbf{x}_{(i)} - \mathbf{x}_{(j)}\|\big)^2 \\ + \alpha \sum_{l=1}^{k} \sum_{i=1}^{n} c_{il} \|\mathbf{x}_{(i)} - \mathbf{y}_{(k)}\|^2 + \beta \sum_{i=1}^{n} e_{ii} \|\mathbf{x}_{(i)} - \mathbf{x}_{(i)}[t-1]\|^2. \end{aligned} \tag{3.16}$$

The first term of (3.16) is the usual MDS stress function, while the second term corresponds to the grouping penalty, and the third term corresponds to the temporal penalty. The constants α and β are the grouping and temporal regularization parameters, respectively.

To optimize (3.16), I begin by re-writing the first two terms into a single term. Define the augmented MDS weight matrix by

$$\tilde{V} = \begin{bmatrix} V & \alpha C \\ \alpha C^T & 0 \end{bmatrix}, \tag{3.17}$$

where the zero corresponds to an appropriately sized all-zero matrix. Similarly, define the $(n+k) \times (n+k)$ augmented desired distance matrix $\tilde{\Delta}$ by filling the added rows and

columns with zeros, i.e.

$$\tilde{\Delta} = \begin{bmatrix} \Delta & 0 \\ 0 & 0 \end{bmatrix} \tag{3.18}$$

Let

$$\tilde{X} = \begin{bmatrix} X \\ Y \end{bmatrix} \tag{3.19}$$

denote the positions of the both the nodes and the group representatives. Then, the first two terms of (3.16) can be written as

$$\frac{1}{2}\sum_{i=1}^{n+k}\sum_{j=1}^{n+k}\tilde{v}_{ij}\left(\tilde{\delta}_{ij} - \|\tilde{\mathbf{x}}_{(i)} - \tilde{\mathbf{x}}_{(j)}\|\right)^2,$$

which has the same form as in the usual stress defined in (3.1). The third term in (3.16) can be written as a quadratic function of $\tilde{X}$, namely

$$\beta\left[\operatorname{tr}\left(\tilde{X}^T\tilde{E}\tilde{X}\right) - 2\operatorname{tr}\left(\tilde{X}^T\tilde{E}\tilde{X}[t-1]\right) + \operatorname{tr}\left(\tilde{X}^T[t-1]\tilde{E}\tilde{X}[t-1]\right)\right],$$

where the $(n+k)\times(n+k)$ matrix $\tilde{E}$ and the $(n+k)\times s$ matrix $\tilde{X}[t-1]$ are constructed by zero-filling as in the definition of $\tilde{\Delta}$.

Following the derivation in Section 3.1.1, for any $(n+k)\times s$ matrix Z, (3.16) can be majorized by

$$\begin{aligned}\frac{1}{2}\sum_{i=1}^{n+k}\sum_{j=1}^{n+k}\tilde{v}_{ij}\tilde{\delta}_{ij}^2 + \operatorname{tr}(\tilde{X}^T\tilde{R}\tilde{X}) - 2\operatorname{tr}(\tilde{X}^T\tilde{S}(Z)Z) + \beta\Big[&\operatorname{tr}(\tilde{X}^T\tilde{E}\tilde{X}) \\ &- 2\operatorname{tr}(\tilde{X}^T\tilde{E}\tilde{X}[t-1]) + \operatorname{tr}(\tilde{X}^T[t-1]\tilde{E}\tilde{X}[t-1])\Big],\end{aligned} \tag{3.20}$$

where $\tilde{R}$ and $\tilde{S}$ are defined by substituting the augmented matrices $\tilde{V}$ and $\tilde{\Delta}$ for V and Δ, respectively, in (3.3) and (3.4). (3.20) is quadratic and convex in X so the minimizer of the

upper bound is found by setting the derivative of (3.20) to 0, resulting in the equation

$$\big(\tilde{R} + \beta\tilde{E}\big)\tilde{X} = \tilde{S}\big(Z\big)Z + \beta\tilde{E}\tilde{X}[t-1].$$

This can again be solved sequentially over each dimension. As in Section 3.1.1, I solve this iteratively using the previous iteration as the majorizer, i.e. at iteration h, solve

$$\big(\tilde{R} + \beta\tilde{E}\big)\tilde{\mathbf{x}}_a^{(h)} = \tilde{S}\big(\tilde{X}^{(h-1)}\big)\tilde{\mathbf{x}}_a^{(h-1)} + \beta\tilde{E}\tilde{\mathbf{x}}_a[t-1]. \tag{3.21}$$

for $\tilde{\mathbf{x}}_a^{(h)}$ for each $a = 1, \ldots, s$. The process is iterated until the convergence criterion (3.7) is attained. The first iterate can be taken to be simply the previous layout $\tilde{\mathbf{x}}_a[t-1]$. Unlike in ordinary MDS, the system of linear equations in (3.21) has a unique solution provided that at least a single node was present at time step $t-1$, because $\tilde{R} + \beta\tilde{E}$ has full rank in this case.

Pseudocode for the DMDS algorithm for $t = 1, 2, \ldots$ is shown in Algorithm 3.1, where $\mathrm{shortest\_paths}(\cdot)$ computes the matrix of shortest paths between all pairs of nodes, and $\mathrm{MDS\_weights}(\cdot)$ computes the MDS weight matrix. (3.21) can be solved by performing a Cholesky factorization on $(\tilde{R} + \beta\tilde{E})$ followed by back substitution. At the initial time step ($t = 0$), there are no previous node positions to initialize with, so a random initialization is used. Also, at $t = 0$, the position of one node should be fixed before solving (3.21) due to the translation-invariance discussed in Section 3.1.1. The time complexity of the algorithm at all subsequent time steps is dominated by the $O(n^3)$ complexity of the Cholesky factorization, assuming $k \ll n$, but the factorization only needs to be computed at the initial iteration ($h = 1$). All subsequent iterations require only matrix-vector products and back substitution and thus have $O(n^2)$ complexity.

Algorithm 3.1 Pseudocode for the DMDS algorithm.

for $t = 1, 2, \ldots$ **do**
  $\Delta \leftarrow \text{shortest\_paths}(W)$
  $V \leftarrow \text{MDS\_weights}(\Delta)$
  Construct $\tilde{V}$ and $\tilde{\Delta}$ using (3.17) and (3.18), respectively
  Construct $\tilde{R}$ by substituting $\tilde{V}$ for V in (3.3)
  $h \leftarrow 0$
  $\tilde{X}^{(0)} \leftarrow \tilde{X}[t-1]$
  repeat
    $h \leftarrow h + 1$
    Construct $\tilde{S}\big(\tilde{X}^{(h-1)}\big)$ by substituting $\tilde{V}$, $\tilde{\Delta}$, and $\tilde{X}^{(h-1)}$ for V, Δ, and Z, respectively, in (3.4)
    for $a = 1, \ldots, s$ **do**
      Solve $\big(\tilde{R} + \beta\tilde{E}\big)\tilde{\mathbf{x}}_a^{(h)} = \tilde{S}\big(\tilde{X}^{(h-1)}\big)\tilde{\mathbf{x}}_a^{(h-1)} + \beta\tilde{E}\tilde{\mathbf{x}}_a[t-1]$ for $\tilde{\mathbf{x}}_a^{(h)}$
    end for
  until $\big[\,\text{Mstress}\,\big(\tilde{X}^{(h-1)}\big) - \text{Mstress}\,\big(\tilde{X}^{(h)}\big)\big]\,/\,\text{Mstress}\,\big(\tilde{X}^{(h-1)}\big) < \epsilon$
  $\tilde{X} \leftarrow \tilde{X}^{(h)}$
end for

3.2.3 Dynamic graph Laplacian layout

The dynamic graph Laplacian layout (DGLL) modified cost is given by the modified energy function

$$\begin{aligned}\text{Menergy}(X, Y) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\|\mathbf{x}_{(i)} - \mathbf{x}_{(j)}\|^2 + \alpha\sum_{l=1}^{k}\sum_{i=1}^{n} c_{il}\|\mathbf{x}_{(i)} - \mathbf{y}_{(l)}\|^2 \\ + \beta\sum_{i=1}^{n} e_{ii}\|\mathbf{x}_{(i)} - \mathbf{x}_{(i)}[t-1]\|^2.\end{aligned} \tag{3.22}$$

Like with DMDS, the first term of (3.22) is the usual GLL energy function, while the second term corresponds to the grouping penalty, and the third term corresponds to the temporal penalty. Again, the parameters α and β correspond to the grouping and temporal regularization parameters, respectively.

I first re-write (3.22) in a more compact form using the graph Laplacian. Define the

augmented adjacency matrix by

$$\tilde{W} = \begin{bmatrix} W & \alpha C \\ \alpha C^T & 0 \end{bmatrix}. \tag{3.23}$$

Notice that the group representatives have been added as nodes to the graph, with edges between each node and its associated representative of weight α. Define the augmented degree matrix by $\tilde{D}$ by $\tilde{d}_{ii} = \sum_{j=1}^{n+k} \tilde{w}_{ij}$, and the augmented graph Laplacian by $\tilde{L} = \tilde{D} - \tilde{W}$. The first two terms of (3.22) can thus be written as $\mathrm{tr}(\tilde{X}^T \tilde{L} \tilde{X})$, where $\tilde{X}$ is as defined in (3.19). The third term of (3.22) can be written as

$$\beta \left[\mathrm{tr}(\tilde{X}^T \tilde{E} \tilde{X}) - 2\,\mathrm{tr}(\tilde{X}^T \tilde{E} \tilde{X}[t-1]) + \mathrm{tr}(\tilde{X}^T[t-1] \tilde{E} \tilde{X}[t-1]) \right], \tag{3.24}$$

where $\tilde{E}$ is zero-filled as described in Section 3.2.2. The final term in (3.24) is independent of $\tilde{X}$ and is henceforth dropped from the modified cost.

I now consider the constraints, which differ depending on whether the layout is degree-normalized, as discussed in Section 3.1.2. I derive the constraints for the degree-normalized layout; the equivalent constraints for the unnormalized layout can simply be obtained by replacing $\tilde{D}$ with the identity matrix in the derivation. First I note that, due to the temporal regularization, the optimal layout is no longer translation-invariant, so I can remove the zero-mean constraint. As a result, the variance and orthogonality constraints become more complicated because I need to subtract the mean. Denote the degree-weighted mean in dimension a by

$$\tilde{\mu}_a = \frac{1}{\sum_{i=1}^{n+k} \tilde{d}_{ii}} \sum_{i=1}^{n+k} \tilde{d}_{ii} \tilde{x}_{ia}.$$

Then the degree-weighted covariance between the a^{th} and b^{th} dimensions is given by

$$\begin{aligned}
\operatorname{cov}(\tilde{\mathbf{x}}_a, \tilde{\mathbf{x}}_b) &= \frac{1}{\sum_{i=1}^{n+k} \tilde{d}_{ii}} \sum_{i=1}^{n+k} \tilde{d}_{ii}(\tilde{x}_{ia} - \tilde{\mu}_a)(\tilde{x}_{ib} - \tilde{\mu}_b) \\
&= \frac{1}{\sum_{i=1}^{n+k} \tilde{d}_{ii}} \sum_{i=1}^{n+k} \tilde{d}_{ii}\tilde{x}_{ia}\tilde{x}_{ib} - \frac{1}{\left(\sum_{i=1}^{n+k} \tilde{d}_{ii}\right)^2} \left(\sum_{i=1}^{n+k} \tilde{d}_{ii}\tilde{x}_{ia}\right)\left(\sum_{i=1}^{n+k} \tilde{d}_{ii}\tilde{x}_{ib}\right) \\
&= \frac{\tilde{\mathbf{x}}_a^T M \tilde{\mathbf{x}}_b}{\operatorname{tr}(\tilde{D})},
\end{aligned}$$

where M is the centering matrix defined by

$$M = \tilde{D} - \frac{\tilde{D}\mathbf{1}\mathbf{1}^T\tilde{D}}{\operatorname{tr}(\tilde{D})}. \tag{3.25}$$

Combining the modified cost function with the modified constraints, the normalized DGLL problem is as follows:

$$\min_{\tilde{X}} \quad \operatorname{tr}(\tilde{X}^T\tilde{L}\tilde{X}) + \beta\left[\operatorname{tr}(\tilde{X}^T\tilde{E}\tilde{X}) - 2\tilde{X}^T\tilde{E}\tilde{X}[t-1])\right] \tag{3.26}$$

$$\text{subject to} \quad \operatorname{tr}(\tilde{X}^T M\tilde{X}) = \operatorname{tr}(\tilde{D})I. \tag{3.27}$$

Again, the unnormalized problem can be obtained by replacing $\tilde{D}$ with the identity matrix in (3.25) and (3.27). Note that (3.26) contains a linear term in $\tilde{X}$. Hence the optimal solution is not given by scaled generalized eigenvectors as in the static GLL problem. (3.26) can be solved using standard algorithms for constrained nonlinear optimization (Bazaraa et al., 2006). The cost function and constraints consist only of linear and quadratic terms, so the gradient and Hessian are easily computed in closed form (see Appendix 3.A). Unfortunately, the problem is not convex due to the equality constraints; thus a good initialization is important. The natural choice is to initialize using the previous layout $\tilde{X}^{(0)} = \tilde{X}[t-1]$. To avoid getting stuck in poor local minima, one could use multiple restarts with random initialization.

Algorithm 3.2 Pseudocode for the DGLL algorithm.

for $t = 1, 2, \ldots$ **do**
  Construct $\tilde{W}$ using (3.23) and its corresponding Laplacian $\tilde{L} = \tilde{D} - \tilde{W}$
  Construct the centering matrix M using (3.25)
  $\tilde{X}^{(0)} \leftarrow \tilde{X}[t-1]$
  Solve (3.26) using the forms for ∇f, g, H, and J in Appendix 3.A
  for $r = 1 \rightarrow max\_restarts$ **do** {if multiple random restarts are necessary}
    Randomly assign $\tilde{X}^{(0)}$
    Solve (3.26) using the forms for ∇f, g, H, and J in Appendix 3.A
  end for
  $\tilde{X} \leftarrow$ best solution to (3.26) over all initializations
end for

Pseudocode for the DGLL algorithm for $t = 1, 2, \ldots$ is shown in Algorithm 3.2. I use the interior-point algorithm of Byrd et al. (1999) to solve (3.26). I find in practice that random restarts are not necessary unless β is extremely small because the temporal regularization penalizes solutions that deviate too far from the previous layout. For other choices of β, I find that the interior-point algorithm indeed converges to the global minimum when initialized using the previous layout. The most time-consuming operation in solving (3.26) consists of a Cholesky factorization, which must be updated at each iteration. At the initial time step ($t = 0$), there are no previous node positions, and hence, no linear term in (3.26), so the layout is obtained using scaled generalized eigenvectors, as described in Section 3.1.2. The time complexity at all subsequent time steps is dominated by the $O(n^3)$ complexity of the Cholesky factorization.

3.2.4 Discussion

I chose to demonstrate the proposed framework with MDS and GLL; however, it is also applicable to other graph layout methods, such as the Fruchterman-Reingold method of force-directed layout (Fruchterman and Reingold, 1991). Since the static cost functions of MDS and GLL encourage different appearances, the same is true of DMDS and DGLL. Ultimately, the decision of which type of layout to use depends on the type of network and user preferences. Kamada-Kawai MDS layouts are often preferred in 2-D because

they discourage nodes from overlapping due to the large MDS weights assigned to maintaining small desired distances. On the other hand, if a 1-D layout is desired, so that the entire sequence can be plotted as a time series, node overlap is a lesser concern. For such applications, DGLL may be a better choice.

Another decision that needs to be made by the user is the choice of the parameters α and β, which can be tuned as desired to create a meaningful animation. Unlike in supervised learning tasks such as classification, there is no ground truth in visualization so the selection of parameters in layout methods is typically done in an ad-hoc fashion. Furthermore, multiple layouts created by differing choices of parameters could be useful for visualizing different portions of the network or yielding different insights (Witten and Tibshirani, 2011). This is particularly true of the grouping regularization parameter α. When a high value of α is used, nodes belonging to the same group are placed much closer together than nodes belonging to different groups. The resulting visualization emphasizes node movements between groups (for nodes that change group between time steps) while sacrificing the quality of the node movements within groups. On the other hand, when a low value of α is used, node movements within groups are more clearly visible, but node movements between groups are more difficult to see. I explore the effect of changing parameters on the resulting animation in several experiments in Section 3.4.

3.3 Related work

The regularized graph layout framework proposed in this chapter employs two types of penalties: a penalty that places nodes belonging to the same group together and a penalty that places nodes near their positions at neighboring time steps. The former problem has been investigated, albeit in a static setting, in the related field of supervised dimensionality reduction. The latter problem has been formulated as an objective in previous work on dynamic network visualization.

3.3.1 Supervised dimensionality reduction

The objective of dimensionality reduction (DR) is to find a mapping $\phi : \mathbb{R}^p \to \mathbb{R}^s$, $p > s$ from a high-dimensional space to a lower-dimensional space while preserving many of the characteristics of the data representation in the high-dimensional space (Lee and Verleysen, 2007). For example, MDS is a DR method that attempts to preserve pairwise distances between data points. In the supervised DR setting, one also has a priori knowledge of the group memberships of some of the data. Supervised DR methods pose the additional constraint that data points within the same group should be closer together in the low-dimensional space than points in separate groups. Notice that this is the same grouping constraint posed in the regularized layout framework proposed in this chapter.

Witten and Tibshirani (2011) proposed a supervised version of MDS (SMDS) that optimizes the following cost function over X:

$$\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\delta_{ij} - \|\mathbf{x}_{(i)} - \mathbf{x}_{(j)}\|)^2 + \alpha \sum_{i,j:y_j>y_i}(y_j - y_i)\sum_{a=1}^{s}\left(\frac{\delta_{ij}}{\sqrt{s}} - (x_{ja} - x_{ia})\right)^2, \quad (3.28)$$

where y_i is an ordinal value denoting the group membership of data point i. Notice that the first term in (3.28) is the ordinary MDS stress with $v_{ij} = 1$ for all i, j, while the second term provides the grouping regularization. α controls the trade-off between the two terms. The key difference between the SMDS grouping penalty and the DMDS grouping penalty proposed in this chapter is in the way groups are treated. SMDS assumes that groups are labeled with an ordinal value that allows them to be ranked, and the form of the grouping penalty in (3.28) does indeed tend to rank groups in $\mathbb{R}^s$ by encouraging $x_{ja} > x_{ia}, a = 1, \ldots, s$ for all $i, j : y_j > y_i$. On the other hand, the proposed grouping penalty in this chapter treats group labels as categorical. It does not rank groups in $\mathbb{R}^s$ but simply pulls nodes belonging to the group together.

Another related method for supervised DR is classification constrained dimensionality reduction (CCDR) (Costa and Hero III, 2005), which is a supervised version of Laplacian

eigenmaps (Belkin and Niyogi, 2003). CCDR optimizes the following cost function over (X, Y):

$$\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\|\mathbf{x}_{(i)}-\mathbf{x}_{(j)}\|^2+\alpha\sum_{l=1}^{k}\sum_{i=1}^{n}c_{il}\|\mathbf{x}_{(i)}-\mathbf{y}_{(l)}\|^2.$$

Notice that this cost function is simply the sum of the GLL energy and the DGLL grouping penalty. Indeed, DGLL can be viewed as an extension of CCDR to time-varying data. The CCDR solution is given by the matrix of generalized eigenvectors $\tilde{U} = [\tilde{\mathbf{u}}_2, \ldots, \tilde{\mathbf{u}}_{s+1}]$ of $(\tilde{L}, \tilde{D})$, where $\tilde{L}$ and $\tilde{D}$ are as defined in Section 3.2.3. Although the addition of the temporal regularization due to the anchoring presence of the previous layout $X[t-1]$ prevents the DGLL problem from being solved using generalized eigenvectors, it discourages large node movements between time steps in order to better preserve the mental map.

3.3.2 Layout of dynamic networks

There have been several previous studies on the problem of laying out dynamic networks while preserving the mental map between time snapshots. Moody et al. (2005) proposed to generate dynamic layouts using a static layout method such as Kamada-Kawai MDS and to initialize at each time step using the layout generated at the previous time step. The idea of this approach is to anchor the nodes initially so that the entire layout does not get rotated. The anchoring differs from the temporal regularization proposed in this chapter, which penalizes changes in node positions over time and can be thought of as anchoring all iterations rather than just the initial iteration. The approach of (Moody et al., 2005) is implemented in the social network visualization software SoNIA (Bender-deMoll and McFarland, 2012). Experimentally, I find that solely anchoring the initialization is insufficient at preventing drastic node movements over time (see Section 3.4 for examples).

Baur and Schank (2008) proposed a temporally regularized MDS algorithm that uses the following localized update rule at each iteration h for each node i at each time step t:

$$x_{ia}^{(h)} = \frac{\tilde{x}_{ia}^{(h-1)} + \beta(e_{ii}x_{ia}[t-1] + f_{ii}x_{ia}[t+1])}{\sum_{j\neq i} v_{ij} + \beta(e_{ii}+f_{ii})}, \tag{3.29}$$

where

$$\tilde{x}_{ia}^{(h-1)} = \sum_{j \neq i} v_{ij} \left(x_{ja}^{(h-1)} + \delta_{ij} \frac{x_{ia}^{(h-1)} - x_{ja}^{(h-1)}}{\|\mathbf{x}_{(i)}^{(h-1)} - \mathbf{x}_{(j)}^{(h-1)}\|} \right),$$

and F is the diagonal matrix defined by

$$f_{ii} = \begin{cases} 1 & \text{node } i \text{ is present at time step } t+1, \\ 0 & \text{otherwise.} \end{cases}$$

This algorithm is implemented in the social network analysis and visualization software Visone (Visone–WWW) and was used in (Leydesdorff and Schank, 2008) for visualizing similarities in journal content over time. (3.29) is an off-line update because it uses both the node positions at time steps $t-1$ and $t+1$ to compute the node position at time step t, whereas the methods I propose, including DMDS, are on-line methods that use only current and past data. (3.29) can be modified into an on-line update by removing the terms involving f_{ii}. It was shown in (Baur and Schank, 2008) that the localized update of (3.29) optimizes the sum of the $\mathrm{Mstress}$ function in (3.16) over all t with $k = 0$, i.e. without a grouping penalty. Likewise, the on-line modification optimizes the $\mathrm{Mstress}$ function at a single time step with $k = 0$. Hence the proposed DMDS layout method can be viewed as an on-line modification of the method of (Baur and Schank, 2008) with the addition of a grouping penalty.

Other methods for layout of dynamic networks have also been proposed (Erten et al., 2004; Frishman and Tal, 2008). TGRIP (Erten et al., 2004) is a modified force-directed layout method with added edges between vertices present at multiple time steps. The user-selected weights of these added edges control the amount of temporal regularization in the layouts. The method of Frishman and Tal (2008) is also a modified force-directed layout. It is an on-line method that uses pinning weights to previous node positions to achieve temporal regularization and a GPU-based implementation to reduce run-time. The emphasis in both methods is on improving scalability to deal with extremely large networks

by using approximations and graph coarsening. As a result, they are applicable to much larger networks than the $O(n^3)$ methods proposed in this chapter. However, these methods do not incorporate any sort of grouping regularization to discourage nodes from deviating too far from other nodes in the same group.

3.4 Experiments

I demonstrate the proposed framework by applying DMDS and DGLL on a simulated data set and two real data sets. Several snapshots of the resulting visualizations are presented.

In the second experiment, there is no a priori group knowledge. Hence I learn the groups using the AFFECT evolutionary spectral clustering algorithm, which I describe in Chapter IV. In the other two experiments, there is a priori group knowledge. I compute layouts both using the known groups and the groups learned by clustering. I also compute layouts using several existing methods for comparison. DMDS is compared to the method of Moody et al. (2005) used in SoNIA (Bender-deMoll and McFarland, 2012) and the method of Baur and Schank (2008) used in Visone (Visone–WWW). The SoNIA layouts are created by anchoring the initial node positions, as described in Section 3.3.2. To provide a fair comparison with DMDS and SoNIA, which are on-line methods, I use an on-line modification of the Visone method, also described in Section 3.3.2. DGLL is compared to the CCDR method of Costa and Hero III (2005) and the standard spectral GLL solution (Koren, 2003).

Summary statistics from the experiments are presented in Tables 3.1 and 3.2 for the MDS- and GLL-based methods, respectively, and are discussed in the individual subsections. The Kamada-Kawai choice of MDS weights is used for all of the MDS-based methods, and degree-normalized layout is used for all of the GLL-based methods.

I define three measures of layout quality: static cost, centroid cost, and temporal cost. The *static cost* measures how well the current layout coordinates fit the current graph snapshot. It is the cost function that would be optimized by a static graph layout algorithm. The

static cost for the MDS-based methods is taken to be the static MDS stress defined in (3.1). The static cost for the GLL-based methods is the GLL energy defined in (3.8). The *centroid cost* is the sum of squared distances between each node and its group centroid, which is also the cost function of the well-known method of k-means clustering (MacQueen, 1967). It is used to measure how close nodes are to members of their group[1]. When prior knowledge of the groups is available, I calculate the centroid cost with respect to the known groups, even for the layouts where groups are learned by clustering. When prior knowledge is not available, I calculate the centroid cost with respect to the learned groups. The *temporal cost* (3.15) is a measure of the amount of node movement between consecutive layouts. The costs displayed are appropriately normalized (either by the number of nodes or pairs of nodes, depending on the quantity) so they are comparable across different data sets. For the MDS-based methods, I also compare the number of iterations required for convergence to a tolerance of $\epsilon = 10^{-4}$.

As expected, DMDS and DGLL have lower centroid and temporal costs than the competing methods due to the incorporation of both grouping and temporal regularization. The lower centroid and temporal costs are achieved by choosing node positions with a higher static cost. Also notice that DMDS requires significantly less iterations to converge than SoNIA, which employs no regularization at all, and slightly less than Visone, which employs only temporal regularization. The results for each experiment will be discussed in greater detail in the following subsections.

3.4.1 Stochastic blockmodel

In this experiment, I generate simulated networks using a stochastic blockmodel (SBM) (Holland et al., 1983). An SBM creates networks with k groups, where nodes in a group are stochastically equivalent, i.e. the probability of forming an edge between nodes i and j

[1]Note that I cannot simply use the grouping cost (3.13) because it is not defined for methods that do not incorporate grouping regularization.

Experiment	Algorithm	MDS stress	Centroid cost	Temporal cost	MDS iterations
SBM	DMDS (known)	0.161 ± 0.000	$\mathbf{0.258 \pm 0.001}$	$\mathbf{0.264 \pm 0.002}$	$\mathbf{46.1 \pm 0.4}$
	DMDS (learned)	0.160 ± 0.000	0.306 ± 0.002	0.296 ± 0.002	$\mathbf{46.8 \pm 0.4}$
	Visone	0.157 ± 0.000	0.428 ± 0.003	0.341 ± 0.002	51.6 ± 0.4
	SoNIA	$\mathbf{0.136 \pm 0.000}$	0.623 ± 0.003	1.206 ± 0.009	107.4 ± 0.9
Newcomb	DMDS (learned)	0.136	**0.645**	**0.089**	**14.8**
	Visone	0.107	1.231	0.125	16.9
	SoNIA	**0.080**	1.523	1.246	57.1
MIT	DMDS (known)	0.155	**1.334**	**0.208**	**38.4**
	DMDS (learned)	0.154	1.527	0.240	40.0
	Visone	0.145	1.923	0.301	47.8
	SoNIA	**0.094**	2.575	3.144	104.0

Table 3.1: Mean costs of MDS-based layouts ($\pm$ standard error for SBM simulation experiment). The smallest quantity (within one standard error) in each column for each experiment is bolded. DMDS results using both a priori known groups (when available) and groups learned by clustering are shown.

Experiment	Algorithm	GLL energy	Centroid cost	Temporal cost
SBM	DGLL (known)	0.658 ± 0.002	$\mathbf{0.296 \pm 0.002}$	$\mathbf{0.752 \pm 0.008}$
	DGLL (learned)	0.655 ± 0.002	0.411 ± 0.005	0.881 ± 0.011
	CCDR	0.629 ± 0.002	0.413 ± 0.003	4.650 ± 0.040
	Spectral	$\mathbf{0.604 \pm 0.002}$	0.964 ± 0.008	4.655 ± 0.029
Newcomb	DGLL (learned)	0.820	**1.325**	**0.379**
	CCDR	0.786	1.334	5.230
	Spectral	**0.761**	1.373	3.981
MIT	DGLL (known)	0.131	**1.251**	**0.278**
	DGLL (learned)	0.128	1.356	0.319
	CCDR	0.099	1.300	4.591
	Spectral	**0.090**	1.643	4.783

Table 3.2: Mean costs of GLL-based layouts ($\pm$ standard error for SBM simulation experiment). The smallest quantity (within one standard error) in each column for each experiment is bolded. DGLL results using both a priori known groups (when available) and groups learned by clustering are shown.

is dependent only on the groups to which i and j belong. An SBM is completely specified by the set of probabilities $\{p_{cd}, c = 1, \ldots, k, d = c, c+1, \ldots, k\}$, which represent the probability of forming an edge between any particular node in group c and any particular node in group d.

I generate 20 independent samples from a 30-node 4-group SBM with parameters $p_{ii} = 0.6$ and $p_{ij} = 0.2$ for all $i \neq j$. Each sample corresponds to a graph snapshot at a single time step. The group memberships are randomly assigned at the initial time step and remain unchanged up to $t = 9$. At $t = 10$, $1/4$ of the nodes are randomly re-assigned to different groups to simulate a change in the network structure. The group memberships are then held constant. I create layouts of the network using parameters $\alpha = \beta = 1$.

In Figure 3.1, I plot the variation over time of the static, centroid, and temporal costs of the MDS-based methods. The costs are averaged over 100 simulation runs. As expected, the static cost is higher for the regularized layouts than for the unregularized SoNIA layout. However, the grouping regularization in DMDS results in lower centroid cost. When the groups are learned by clustering, the centroid cost is slightly higher than with the known groups, which is to be expected because the groups are not perfectly recovered by clus-

tering, but the centroid cost is still much lower than that of Visone and SoNIA. Although Visone has only temporal regularization, notice that it also has a lower centroid cost than SoNIA. This is because the SBM parameters are held constant from time steps 0 to 9 and from time steps 10 to 19, so that the group structure can be partially revealed by temporal regularization alone once enough time samples have been collected. The temporal regularization of both DMDS and Visone results in a significantly lower temporal cost than SoNIA. The grouping regularization of DMDS also decreases the temporal cost slightly compared to Visone. An added benefit of the regularization in DMDS is the significant reduction in the number of iterations required for the MDS algorithm to converge, as shown in Table 3.1. On average, DMDS required less than half as many iterations as SoNIA, and slightly less than Visone.

In Figure 3.2, I plot the variation over time of the static, centroid, and temporal costs of the GLL-based methods. Similar to the MDS-based methods, the static cost is higher for the regularized layouts, but the centroid and temporal costs are much lower. In particular, only DGLL is able to generate layouts with low temporal cost. The grouping regularization in CCDR reduces the centroid cost but does not improve the temporal cost. The temporal regularization in DGLL also helps to lower the centroid cost; in particular, from Table 3.2, it should be noted that the mean centroid cost of CCDR, with known groups, is within one standard error of the mean centroid cost of DGLL, with unknown groups learned by clustering.

I demonstrate the effects of varying the regularization parameters in DMDS and DGLL, respectively, in Figures 3.3 and 3.4. I generate layouts using 10 choices each of α and β, uniformly distributed on a log scale between 0.1 and 10. The observations are similar for both DMDS and DGLL. As expected, the temporal cost decreases for increasing β. For low values of β, increasing α also decreases the temporal cost. This is a sensible result because nodes can move significantly over time but must remain close to the group representative, which lowers the temporal cost. The result is slightly different when it comes to the centroid

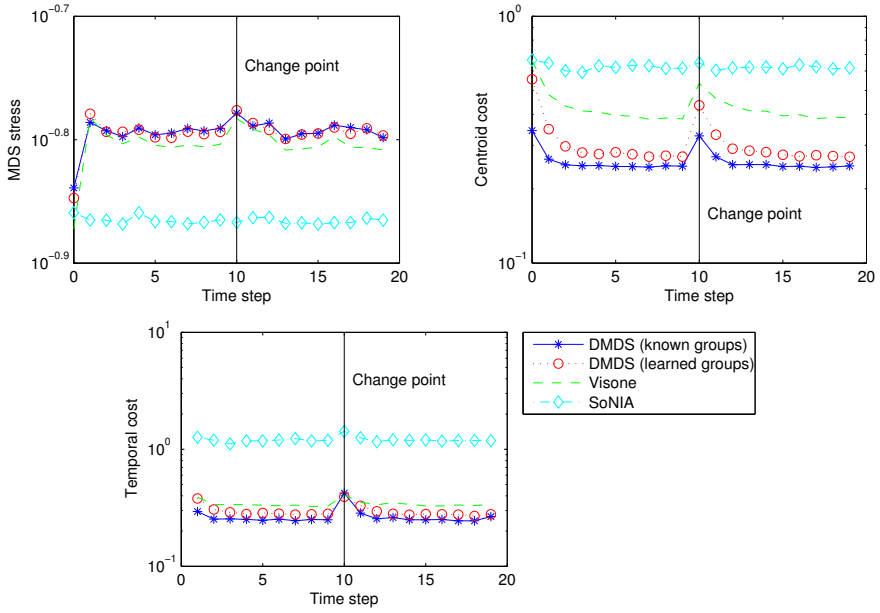


Figure 3.1: Costs of MDS-based layouts in the SBM experiment at each time step. The DMDS layouts have the lowest centroid and temporal costs but the highest MDS stress due to the regularization.

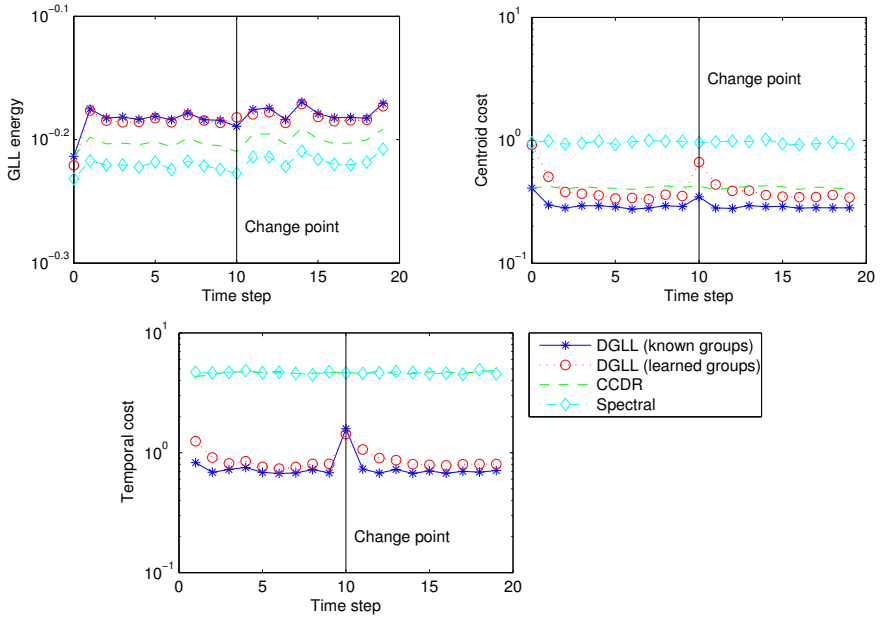


Figure 3.2: Costs of GLL-based layouts in the SBM experiment at each time step. The DGLL layouts have the lowest centroid and temporal costs but the highest GLL energy.

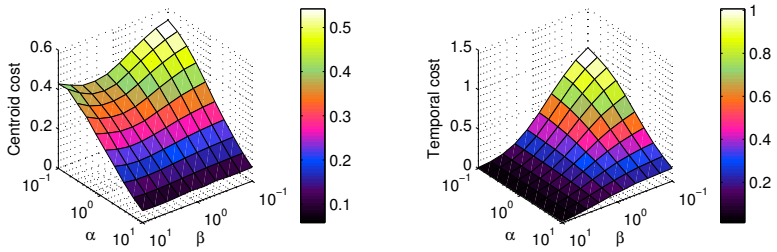


Figure 3.3: Mean centroid and temporal costs of DMDS layouts in the SBM experiment as functions of α and β. The centroid and temporal costs decrease as α and β are increased, respectively; however, α also affects the temporal cost, and β also affects the centroid cost.

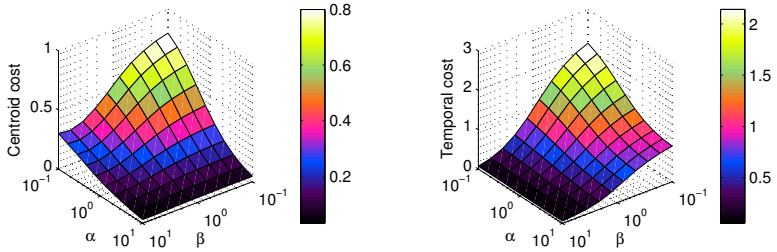


Figure 3.4: Mean centroid and temporal costs of DGLL layouts in the SBM experiment as functions of α and β. The behavior of both costs as functions of α and β is similar to their behavior in DMDS.

cost. As expected, increasing α decreases centroid cost. For low values of α, increasing β also decreases centroid cost to a point, but a very high β may actually increase centroid cost, especially in DMDS. This is also a sensible result because a very high β places too much weight on the initial time step and prevents nodes from moving towards their group representative at future time steps.

From this experiment one can see that there is a coupled effect between grouping and temporal regularization, and that a combination of both can sometimes result in better performance with respect to both centroid and temporal cost. However, it is important to note that this is not always true. For example, if a node changes group between two time steps, then the two penalties can oppose each other, with the temporal penalty attempting to pull the node towards its previous position and the grouping penalty attempting to pull the node towards its current representative, which could be quite far from the node's previous posi-

tion. This is reflected by the spike in both temporal and grouping costs at $t = 10$ in Figures 3.1 and 3.2 for DMDS and DGLL, respectively.

3.4.2 Newcomb's fraternity

This data set was collected by Nordlie and Newcomb (Nordlie, 1958; Newcomb, 1961) as part of an experiment on interpersonal relations. It has been examined in previous studies including (Moody et al., 2005; Bender-deMoll and McFarland, 2006). 17 incoming male transfer students at the University of Michigan were housed together in fraternity housing. Each week, the participants ranked their preference of each of the other individuals in the house, in private, from 1 to 16. Data was collected over 15 weeks in a semester, with one week of data missing during week 9, corresponding to Fall break.

I process the rank data in the same manner as (Moody et al., 2005; Bender-deMoll and McFarland, 2006) to give a fair comparison with SoNIA. Graph snapshots are created by connecting each participant to his 4 most preferred students with weights from 4 decreasing to 1 corresponding to the most preferred to the 4th most preferred student. The graph is converted to an undirected graph by taking the edge weight between i and j to be the larger of the directed edge weights. The weights are converted into dissimilarities by dividing each similarity weight by the maximum similarity of 4. No group information is known a priori, so the group structure is learned using the AFFECT clustering algorithm.

In Figure 3.5, I show a time plot of 1-D layouts created using DGLL, where the color of a line segment between time steps t and $t + 1$ denotes the group membership of the node at time step t, and the location of the endpoints correspond to the node's position in the layouts at time steps t and $t + 1$. The regularization parameters are chosen to be $\alpha = \beta = 1$. While a 1-D layout does a poor job of conveying the topology of the network, some temporal trends can be seen. For example, two mostly stable groups form after several weeks, but two nodes appear to switch from the blue to the red group around Fall break. In Figure 3.6, I show the same time plot created using CCDR, which uses only grouping regularization

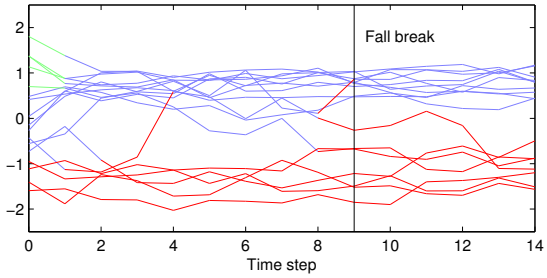


Figure 3.5: Time plots of 1-D DGLL layouts of Newcomb's fraternity, colored by learned groups. Node positions in the layout are relatively stable over time unless nodes are changing group.

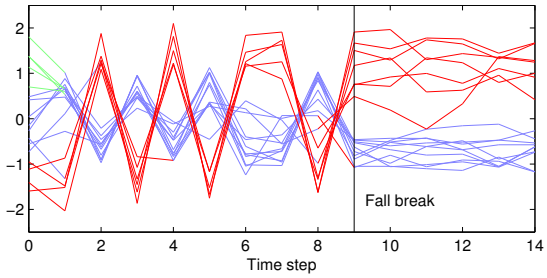


Figure 3.6: Time plots of 1-D CCDR layouts of Newcomb's fraternity, colored by learned groups. Node positions are unstable so it is difficult to see when nodes change group.

as discussed in Section 3.3.2. Notice that the groups are well-separated, but the temporal evolution is difficult to interpret due to the lack of temporal regularization. In particular, the movement of all of the nodes obscures the previous observation of two nodes switching from the blue to the red group around Fall break.

In Figures 3.7-3.9, I present a comparison of the first four snapshots from the layouts created using DMDS, Visone, and SoNIA, respectively. In all of the figures, the top row corresponds to the layouts, and the bottom row illustrates the movements of each node over time. In the plots on the bottom row, each node is drawn twice: once at its current position at time t and once at its previous position at time $t-1$. An edge connects these two positions; the length of the edge indicates how far a node has moved between time steps $t-1$ and t.

At $t = 0$, the blue and green groups are mixed together in the Visone and SoNIA

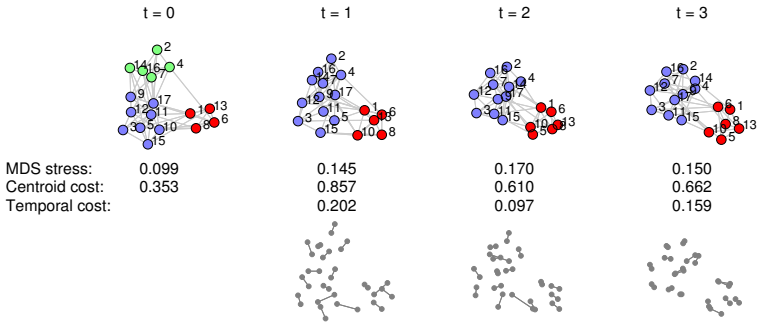


Figure 3.7: Layouts of Newcomb's fraternity at four time steps (top row) generated using proposed DMDS algorithm and node movements between layouts (bottom row). The groups remain well-separated.

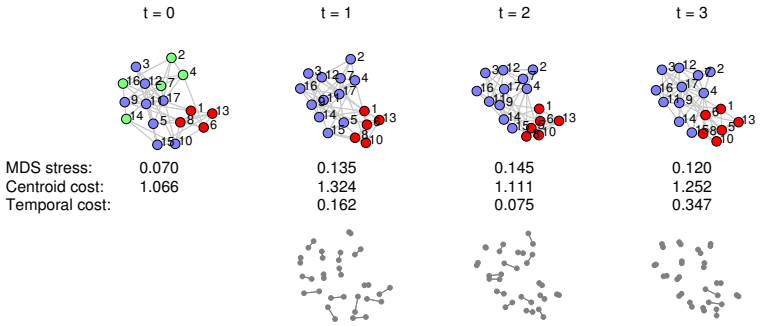


Figure 3.8: Layouts of Newcomb's fraternity at four time steps (top row) using the Visone algorithm and node movements between layouts (bottom row). The groups are not as well-separated as in the DMDS layouts.

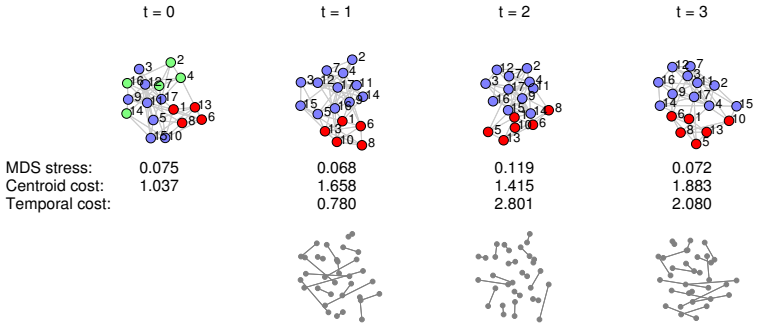


Figure 3.9: Layouts of Newcomb's fraternity at four time steps (top row) using the SoNIA algorithm and node movements between layouts (bottom row). There is excessive node movement, and the groups are not as well-separated as in the DMDS layouts.

layouts, while they are easily distinguished in the DMDS layout due to the grouping regularization. Furthermore, the node movements over time in the SoNIA layouts are much more extreme than in the DMDS layouts. The excessive node movement is reflected in the substantially higher temporal cost of SoNIA compared to DMDS, as shown in Table 3.1. DMDS also outperforms Visone in both mean centroid and temporal cost, although the improvement in temporal cost is smaller than compared to SoNIA because Visone also employs temporal regularization. Finally, the mean number of iterations required for the SoNIA layout to converge is almost four times that of DMDS, so DMDS presents significant computational savings in addition to better preservation of the mental map.

3.4.3 MIT Reality Mining

The MIT Reality Mining data set (Eagle et al., 2009) was collected as part of an experiment on inferring social networks by using cell phones as sensors. 94 students and staff at MIT were given access to smartphones that were monitored over two semesters. The phones were equipped with Bluetooth sensors, and each phone recorded the Media Access Control addresses of nearby Bluetooth devices at five-minute intervals. Using this proximity data, I construct a sequence of graph snapshots where each participant is connected to the 5 participants he or she was in highest proximity to during a time step. I divide the data into time steps of one week, resulting in 46 time steps between August 2004 and June 2005. From the MIT academic calendar (MIT–WWW), the dates of important events such as the beginning and end of school terms are known. It is also known that 26 of the participants were incoming students at the university's business school, while the rest were colleagues working in the same building. These affiliations are used as the known groups.

The DMDS layouts at four time steps computed using the known groups with $\alpha = 1, \beta = 3$ are shown in Figure 3.10. A higher value of β is chosen compared to the previous experiments in order to create more stable layouts due to the higher number of nodes. Node labels are not displayed to reduce clutter in the figure. $t = 5$ corresponds to the first

week of classes. Notice that the two groups are slightly overlapped at this time step. As time progresses, the group of incoming students separates quite clearly from the colleagues working in the same building. This result suggests that the incoming students are spending more time in proximity with each other than with the remaining participants, which one would expect as the students gain familiarity with each other as the semester unfolds.

The same observation can be made from the DMDS layouts computed using the groups learned by the AFFECT clustering algorithm. Initially at $t = 5$, the separation between groups is not clear so many nodes are not correctly classified, but at subsequent time steps when the separation is clearer, almost all of the nodes are correctly classified. Between time steps 5 and 6 many nodes switch groups. This can be seen in Figure 3.11, where the colors correspond to the learned groups rather than the true groups. These layouts are created using $\alpha = 5, \beta = 3$; the high value of α emphasizes the node movements between groups while sacrificing the quality of movements within groups, as discussed in Section 3.2.4. Compare these layouts to those shown in Figure 3.12, which are created using $\alpha = 1/5, \beta = 3$. The low value of α better shows movements within groups, but the large changes in groups between time steps 5 and 6 is not as obvious as in Figure 3.11. Both layouts are useful and provide slightly different insights into the network dynamics; however, the observation of the incoming students separating from the other participants is evident in both visualizations.

The benefits of the regularization can be seen once again from the statistics in Tables 3.1 and 3.2. With group information provided, the DMDS and DGLL layouts have lower mean centroid and temporal costs compared to all competing layouts. Without group information, the DMDS and DGLL layouts using groups learned by clustering still outperform competing methods in temporal cost, and only CCDR, which uses the known group information, outperforms DGLL without group information in terms of centroid cost. The DMDS methods also converge more quickly than both Visone and SoNIA.

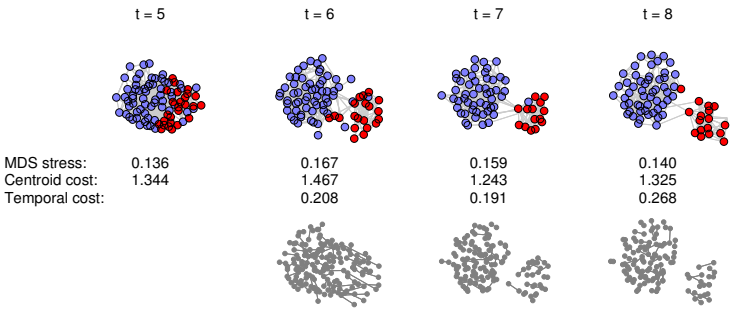


Figure 3.10: DMDS layouts of MIT Reality Mining data at four time steps using the known groups. Blue nodes denote colleagues working in the same building, and red nodes denote incoming students. The incoming students separate from the others after the first week of classes ($t = 5$).

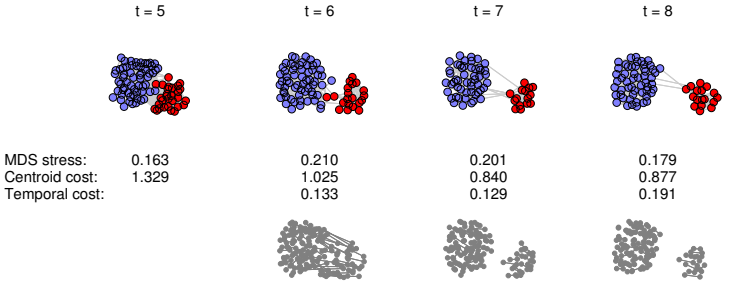


Figure 3.11: DMDS layouts of MIT Reality Mining data at four time steps with $\alpha = 5, \beta = 3$ using groups learned by clustering. Colors correspond to learned groups. There is a lot of node movement between groups but very little movement within groups, resulting in high MDS stress.

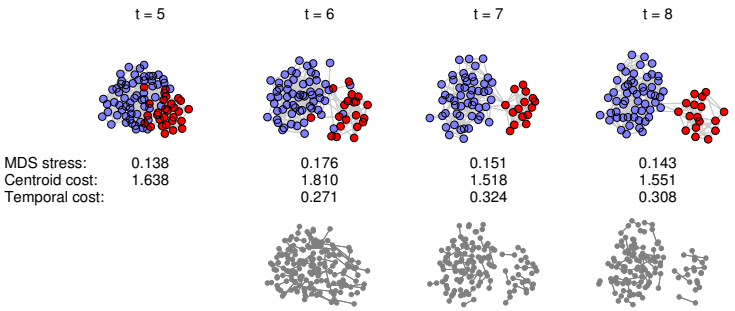


Figure 3.12: DMDS layouts of MIT Reality Mining data at four time steps with $\alpha = 1/5, \beta = 3$ using groups learned by clustering. Colors correspond to learned groups. There is more movement within groups, resulting in lower MDS stress, but it is more difficult to identify movement between groups.

3.5 Summary

In this chapter I proposed a regularized graph layout framework for dynamic network visualization. The proposed framework incorporates grouping and temporal regularization into graph layout in order to discourage nodes from deviating too far from other nodes in the same group and from their previous position, respectively. The layouts are generated in an on-line manner using only present and past data. I introduced two dynamic layout algorithms, DMDS and DGLL, which are regularized versions of their static counterparts, and demonstrated the importance of the regularizers for preserving the mental map in multiple experiments. The proposed methods generalize existing approaches for temporal regularization in MDS and grouping regularization in GLL.

3.A DGLL solution in 2-D

I derive the expressions for the terms ∇f, g, H, and J used in DGLL for the 2-D case. These vectors and matrices are computed at each iteration in the DGLL algorithm to solve (3.26) using the interior-point algorithm (Byrd et al., 1999) as discussed in Section 3.2.3. The constraints can be written as $g(\tilde{X}) = 0$ where

$$g(\tilde{X}) = \begin{bmatrix} \tilde{\mathbf{x}}_1^T M \tilde{\mathbf{x}}_1 - \mathrm{tr}(\tilde{D}) \\ \tilde{\mathbf{x}}_2^T M \tilde{\mathbf{x}}_2 - \mathrm{tr}(\tilde{D}) \\ \tilde{\mathbf{x}}_2^T M \tilde{\mathbf{x}}_1 \end{bmatrix}.$$

The gradient of the objective function is given by

$$\nabla f(\tilde{X}) = \begin{bmatrix} (2\tilde{L} + 2\beta\tilde{E})\tilde{\mathbf{x}}_1 - 2\beta\tilde{E}\tilde{\mathbf{x}}_1[t-1] \\ (2\tilde{L} + 2\beta\tilde{E})\tilde{\mathbf{x}}_2 - 2\beta\tilde{E}\tilde{\mathbf{x}}_2[t-1] \end{bmatrix}.$$

The Jacobian of the constraints is given by

$$J(\tilde{X}) = \begin{bmatrix} 2\tilde{\mathbf{x}}_1^T M & \mathbf{0} \\ \mathbf{0} & 2\tilde{\mathbf{x}}_2^T M \\ \tilde{\mathbf{x}}_2^T M & \tilde{\mathbf{x}}_1^T M \end{bmatrix}.$$

Finally, the Hessian is obtained by

$$\begin{aligned} H(\tilde{X}, \boldsymbol{\mu}) &= \nabla^2 f(\tilde{X}) + \mu_1 \nabla^2 g_1(\tilde{X}) + \mu_2 \nabla^2 g_2(\tilde{X}) + \mu_3 \nabla^2 g_3(\tilde{X}) \\ &= \begin{bmatrix} 2\tilde{L} + 2\beta\tilde{E} + 2\mu_1 M & \mu_3 M \\ \mu_3 M & 2\tilde{L} + 2\beta\tilde{E} + 2\mu_2 M \end{bmatrix}. \end{aligned}$$

CHAPTER IV

TRACKING COMMUNITIES IN DYNAMIC NETWORKS

An empirical finding from many real networks in different fields is the presence of communities or clusters of nodes (Girvan and Newman, 2002). I use the terms "community" and "cluster" interchangeably to denote a group of nodes with stronger ties to other nodes within the group than to nodes outside the group. In a dynamic network where nodes and edges evolve over time, one might expect the communities to evolve over time as well. One could track the evolution of communities in a dynamic network by running a static community detection algorithm on each time snapshot individually; indeed, such an approach is commonly employed in the literature. However, this approach is extremely sensitive to noise and other short-term variations, and the detected communities are unstable and inconsistent with communities detected at neighboring time steps. Better performance can be obtained using *evolutionary clustering* algorithms that take advantage of the dynamic nature of the network. Specifically, evolutionary clustering algorithms aim to produce clustering results that reflect long-term drifts in the statistical properties of the network while being robust to short-term variations[1].

Several evolutionary clustering algorithms have recently been proposed by adding a temporal smoothness penalty to the cost function of a static clustering method. This penalty

[1]The term "evolutionary clustering" has also been used to refer to clustering algorithms motivated by biological evolution, which are unrelated to the methods discussed in this chapter.

prevents the clustering result at any given time from deviating too much from the clustering results at neighboring time steps. This approach has produced evolutionary extensions of commonly used static clustering methods such as agglomerative hierarchical clustering (Chakrabarti et al., 2006), k-means (Chakrabarti et al., 2006), Gaussian mixture models (Zhang et al., 2009), and spectral clustering (Chi et al., 2007; Tang et al., 2008) among others. How to choose the penalty weight in an optimal manner in practice, however, remains an open problem.

In this chapter, I propose a different approach to evolutionary clustering by treating it as a problem of *proximity tracking* followed by static clustering (Section 4.2). I model the observed matrix of proximities between objects at each time step, which can be either similarities or dissimilarities, as a linear combination of a *true proximity matrix* and a zero-mean noise matrix. The true proximities, which vary over time, can be viewed as *unobserved states* of a dynamic system. Our approach involves estimating these states using both current and past proximities, then performing static clustering on the state estimates.

The states are estimated using a restricted class of estimators known as shrinkage estimators. I develop a method for estimating the optimal weight to place on past proximities so as to minimize the mean squared error (MSE) between the true proximities and our estimates. I call this weight the *forgetting factor*. One advantage of our approach is that it provides an explicit formula for the optimal forgetting factor, unlike existing evolutionary clustering methods. The forgetting factor is estimated adaptively, which allows it to vary over time to adjust to the conditions of the dynamic system.

The proposed framework, which I call Adaptive Forgetting Factor for Evolutionary Clustering and Tracking (AFFECT), can extend any static clustering algorithm that uses pairwise similarities or dissimilarities into an evolutionary clustering algorithm. It is flexible enough to handle changes in the number of clusters over time and to accommodate objects entering and leaving the data set between time steps. I demonstrate how AFFECT can be used to extend three popular static clustering algorithms, namely hierarchical clus-

tering, k-means, and spectral clustering, into evolutionary clustering algorithms (Section 4.3).

The proposed framework is tested on several synthetic and real data sets (Section 4.4). I find that it not only outperforms static clustering, but also outperforms existing evolutionary clustering algorithms due to the adaptively selected forgetting factor.

4.1 Background

4.1.1 Static clustering algorithms

I begin by reviewing three commonly used static clustering algorithms. I demonstrate the evolutionary extensions of these algorithms in Section 4.3, although the AFFECT framework can be used to extend many other static clustering algorithms. The term "clustering" is used in this chapter to refer to both data clustering and graph clustering. The notation $i \in c$ is used to denote object i being assigned to cluster c. $|c|$ denotes the number of objects in cluster c, and $\mathcal{C}$ denotes a clustering result (the set of all clusters).

In the case of data clustering, I assume that the n objects in the data set are stored in an $n \times p$ matrix X, where object i is represented by a p-dimensional feature vector $\mathbf{x}_i$ corresponding to the ith row of X. From these feature vectors, one can create a proximity matrix W, where w_{ij} denotes the proximity between objects i and j, which could be their Euclidean distance or any other similarity or dissimilarity measure.

For graph clustering, I assume that the n nodes in the graph are represented by an $n \times n$ adjacency matrix W where w_{ij} denotes the weight of the edge between nodes i and j. If there is no edge between i and j, then $w_{ij} = 0$. For the usual case of undirected graphs with non-negative edge weights, an adjacency matrix is equivalent to a similarity matrix, so I shall refer to it also as a proximity matrix.

Algorithm 4.1 A general algorithm for agglomerative hierarchical clustering.

1: Assign each object to its own cluster
2: **repeat**
3: Compute dissimilarities between each pair of clusters
4: Merge clusters with the lowest dissimilarity
5: **until** all objects are merged into one cluster
6: **return** dendrogram

Agglomerative hierarchical clustering

Agglomerative hierarchical clustering algorithms are greedy algorithms that create a hierarchical clustering result, often represented by a dendrogram (Hastie et al., 2001). The dendrogram can be cut at a certain level to obtain a flat clustering result. There are many variants of agglomerative hierarchical clustering. A general algorithm for agglomerative hierarchical clustering is provided in Algorithm 4.1. Varying the definition of dissimilarity between a pair of clusters often changes the clustering results. Three common choices are to use the minimum dissimilarity between objects in the two clusters (single linkage), the maximum dissimilarity (complete linkage), or the average dissimilarity (average linkage) (Hastie et al., 2001).

k-means

k-means clustering (MacQueen, 1967; Hastie et al., 2001) attempts to find clusters that minimize the sum of squares cost function

$$\mathcal{D}(X, \mathcal{C}) = \sum_{c=1}^{k} \sum_{i \in c} \|\mathbf{x}_i - \mathbf{m}_c\|^2, \tag{4.1}$$

where $\|\cdot\|$ denotes the l_2-norm, and $\mathbf{m}_c$ is the centroid of cluster c, given by

$$\mathbf{m}_c = \frac{\sum_{i \in c} \mathbf{x}_i}{|c|}.$$

Each object is assigned to the cluster with the closest centroid. The cost of a clustering result $\mathcal{C}$ is simply the sum of squared Euclidean distances between each object and its

Algorithm 4.2 k-means algorithm implemented using similarity matrix.

1: $i \leftarrow 0$
2: $\mathcal{C}^{(0)} \leftarrow$ vector of random integers in $\{1, \ldots, k\}$
3: Compute similarity matrix W
4: **repeat**
5: $\quad i \leftarrow i + 1$
6: $\quad$ Calculate squared distance between all objects and centroids using (4.2)
7: $\quad$ Compute $\mathcal{C}^{(i)}$ by assigning each object to its closest centroid
8: **until** $\mathcal{C}^{(i)} = \mathcal{C}^{(i-1)}$
9: **return** $\mathcal{C}^{(i)}$

closest centroid. The squared distance in (4.1) can be rewritten as

$$\|\mathbf{x}_i - \mathbf{m}_c\|^2 = w_{ii} - \frac{2\sum_{j \in c} w_{ij}}{|c|} + \frac{\sum_{j,l \in c} w_{jl}}{|c|^2}, \tag{4.2}$$

where $w_{ij} = \mathbf{x}_i \mathbf{x}_j^T$, the dot product of the feature vectors. Using the form of (4.2) to compute the k-means cost in (4.1) allows the k-means algorithm to be implemented with only the similarity matrix $W = [w_{ij}]_{i,j=1}^n$ consisting of all pairs of dot products. The algorithm is shown in Algorithm 4.2.

Spectral clustering

Spectral clustering (Shi and Malik, 2000; Ng et al., 2001; von Luxburg, 2007) is a popular modern clustering technique inspired by spectral graph theory. It can be used for both data and graph clustering. When used for data clustering, the first step in spectral clustering is to create a similarity graph with nodes corresponding to the objects and edge weights corresponding to the similarities between objects. I represent the graph by an adjacency matrix W with edge weights w_{ij} given by a positive definite similarity function $s(\mathbf{x}_i, \mathbf{x}_j)$. The most commonly used similarity function is the Gaussian similarity function $s(\mathbf{x}_i, \mathbf{x}_j) = \exp\{-\|\mathbf{x}_i - \mathbf{x}_j\|^2/(2\rho^2)\}$ (Ng et al., 2001), where ρ is a scaling parameter. Let D denote a diagonal matrix with elements corresponding to row sums of W. Define the unnormalized graph Laplacian matrix by $L = D - W$ and the normalized Laplacian matrix (Chung, 1997) by $\mathcal{L} = I - D^{-1/2} W D^{-1/2}$.

Three common variants of spectral clustering are average association (AA), ratio cut

Algorithm 4.3 Normalized cut spectral clustering algorithm.

1: $Z \leftarrow k$ smallest eigenvectors of $\mathcal{L}$
2: **for** $i = 1$ to k **do**
3: $\quad \mathbf{z}_i \leftarrow \mathbf{z}_i / \|\mathbf{z}_i\|$
4: **end for**
5: $\mathcal{C} \leftarrow \mathrm{kmeans}(Z)$
6: **return** $\mathcal{C}$

(RC), and normalized cut (NC) (Shi and Malik, 2000). Each variant is associated with an NP-hard graph optimization problem. The normalized cut graph optimization problem was previously described in Section 2.2.1. Spectral clustering solves relaxed versions of these problems. The relaxed problems can be written as (von Luxburg, 2007; Chi et al., 2007)

$$\mathrm{AA}(Z) = \max_{Z \in \mathbb{R}^{n \times k}} \mathrm{tr}(Z^T W Z) \text{ subject to } Z^T Z = I \tag{4.3}$$

$$\mathrm{RC}(Z) = \min_{Z \in \mathbb{R}^{n \times k}} \mathrm{tr}(Z^T L Z) \text{ subject to } Z^T Z = I \tag{4.4}$$

$$\mathrm{NC}(Z) = \min_{Z \in \mathbb{R}^{n \times k}} \mathrm{tr}(Z^T \mathcal{L} Z) \text{ subject to } Z^T Z = I. \tag{4.5}$$

These are variants of a trace optimization problem; the solutions are given by a generalized Rayleigh-Ritz theorem (Lütkepohl, 1997). The optimal solution to (4.3) consists of the matrix containing the eigenvectors corresponding to the k highest eigenvalues of W as columns. Similarly, the optimal solutions to (4.4) and (4.5) consist of the matrices containing the eigenvectors corresponding to the k smallest eigenvalues of L and $\mathcal{L}$, respectively. The optimal relaxed solution Z is then discretized to obtain a clustering result, typically by running the standard k-means algorithm on the rows of Z or a normalized version of Z.

An algorithm (Ng et al., 2001) for normalized cut spectral clustering is shown in Algorithm 4.3. To perform ratio cut spectral clustering, compute eigenvectors of L instead of $\mathcal{L}$ and skip the for loop. Similarly, to perform average association spectral clustering, compute instead the k largest eigenvectors of W and skip the for loop.

4.1.2 Related work

I now summarize existing work in evolutionary clustering and the related area of incremental clustering.

Incremental clustering

The term "incremental clustering" has typically been used to describe two types of clustering problems[2]:

1. Sequentially clustering objects that are each observed only once.
2. Clustering objects that are each observed over multiple time steps by incrementally updating clustering results at previous time steps.

Type 1 is also known as data stream clustering, and the focus is on clustering the data in a single pass and with limited memory (Charikar et al., 2004; Gupta and Grossman, 2004). It is not directly related to this chapter because in data stream clustering each object is observed only once.

Type 2 is of greater relevance to this chapter and targets the same problem setting. Several incremental algorithms of this type have been proposed (Li et al., 2004; Sun et al., 2007; Ning et al., 2010). These incremental clustering algorithms could also be applied to the type of problems I consider; however, the focus of incremental clustering is on low computational cost at the expense of clustering quality. The incremental clustering result is often worse than the result of performing static clustering at each time step, which is already a suboptimal approach as mentioned at the beginning of the chapter. On the other hand, evolutionary clustering is concerned with improving clustering quality by intelligently combining data from multiple time steps and is capable of outperforming static clustering.

---

[2]It is also sometimes used to refer to the simple approach of performing static clustering at each time step.

Evolutionary clustering

The topic of evolutionary clustering has attracted significant attention in recent years. Chakrabarti et al. (2006) introduced the problem and proposed a general framework for evolutionary clustering by adding a temporal smoothness penalty to a static clustering method. Evolutionary extensions for agglomerative hierarchical clustering and k-means were presented as examples of the framework. Chi et al. (2007) expanded on this idea by proposing two frameworks for evolutionary spectral clustering, which they called Preserving Cluster Quality (PCQ) and Preserving Cluster Membership (PCM). Both frameworks proposed to optimize the modified cost function

$$C_{\text{total}} = \alpha\, C_{\text{temporal}} + (1 - \alpha)\, C_{\text{snapshot}}, \tag{4.6}$$

where C_{snapshot} denotes the ordinary spectral clustering cost, which is typically taken to be the average association, ratio cut, or normalized cut as discussed in Section 4.1.1. The two frameworks differ in how the temporal smoothness penalty C_{temporal} is defined. In PCQ, C_{temporal} is defined to be the cost of applying the clustering result at time t to the similarity matrix at time $t - 1$. In other words, it penalizes clustering results that disagree with past similarities. In PCM, C_{temporal} is defined to be a measure of distance between the clustering results at time t and $t - 1$. In other words, it penalizes clustering results that disagree with past clustering results. My work takes a different approach than that of Chi et al. (2007) but the resulting framework shares some similarities with the PCQ framework. In particular, AFFECT paired with average association spectral clustering is an extension of PCQ to longer history, which I discuss in Section 4.3.3.

Following these works, other evolutionary clustering algorithms that attempt to optimize the modified cost function defined in (4.6) have been proposed (Tang et al., 2008; Lin et al., 2009; Zhang et al., 2009; Mucha et al., 2010). The definitions of snapshot and temporal cost and the clustering algorithms vary by approach. None of the aforementioned

works address the problem of how to choose the parameter α in (4.6), which determines how much weight to place on historic data or clustering results. It has typically been suggested (Chi et al., 2007; Lin et al., 2009) to choose it in an ad-hoc manner according to the user's subjective preference on the temporal smoothness of the clustering results.

It could also be beneficial to allow α to vary with time. Zhang et al. (2009) proposed to choose α adaptively by using a test statistic for checking dependency between two data sets (Gretton et al., 2007). However, this test statistic also does not satisfy any optimality properties for evolutionary clustering and still depends on a global parameter reflecting the user's preference on temporal smoothness, which is undesirable.

The existing method that is most similar to AFFECT is that of Rosswog and Ghose (2008), which I refer to as RG. The authors proposed evolutionary extensions of k-means and agglomerative hierarchical clustering by filtering the feature vectors using a Finite Impulse Response (FIR) filter, which combines the last $l + 1$ measurements of the feature vectors by the weighted sum $\mathbf{y}_i^t = b_0\mathbf{x}_i^t + b_1\mathbf{x}_i^{t-1} + \cdots + b_l\mathbf{x}_i^{t-l}$, where l is the order of the filter, $\mathbf{y}_i^t$ is the filter output at time t, and $b_0, \ldots, b_l$ are the filter coefficients. The proximities are then calculated between the filter outputs rather than the feature vectors.

The main resemblance between RG and AFFECT is that RG is also based on tracking followed by static clustering. In particular, RG adaptively selects the filter coefficients based on the dissimilarities between cluster centroids at the past l time steps. However, RG cannot accommodate varying numbers of clusters over time nor can it deal with objects entering and leaving at various time steps. It also struggles to adapt to changes in clusters, as I demonstrate in Section 4.4. AFFECT, on the other hand, is able to adapt quickly to changes in clusters and is applicable to a much larger class of problems.

Finally, there has also been recent interest in model-based evolutionary clustering. In addition to the aforementioned method involving mixtures of exponential families (Zhang et al., 2009), methods have also been proposed using semi-Markov models (Wang et al., 2007), Dirichlet process mixtures (DPMs) (Ahmed and Xing, 2008; Xu et al., 2008b), and

hierarchical DPMs (Xu et al., 2008a,b; Zhang et al., 2010). For these methods, the temporal evolution is controlled by hyperparameters that can be estimated in some cases.

4.2 Proposed evolutionary framework

The proposed framework treats evolutionary clustering as a tracking problem followed by ordinary static clustering. In the case of data clustering, I assume that the feature vectors have already been converted into a proximity matrix, as discussed in Section 4.1.1.

I treat the proximity matrices, denoted by W^t, as realizations from a non-stationary random process indexed by discrete time steps, denoted by the superscript t. Furthermore I posit the linear observation model

$$W^t = \Psi^t + N^t, \qquad t = 0, 1, 2, \ldots \tag{4.7}$$

where Ψ^t is an unknown deterministic matrix of unobserved states, and N^t is a zero-mean noise matrix. Ψ^t changes over time to reflect long-term drifts in the proximities. Although I do not place a dynamic model on Ψ^t, I assume that it is varying smoothly over time. I refer to Ψ^t as the *true proximity matrix*, and my goal is to accurately estimate it at each time step. On the other hand, N^t reflects short-term variations due to noise. Thus I assume that $N^t, N^{t-1}, \ldots, N^0$ are mutually independent.

A common approach for tracking unobserved states in a dynamic system is to use a Kalman filter (Harvey, 1989; Haykin, 2001) or some variant. Since the states correspond to the true proximities, there are $O(n^2)$ states and $O(n^2)$ observations, which makes the Kalman filter impractical for two reasons. First, it involves specifying a parametric model for the state evolution over time, and secondly, it requires the inversion of an $O(n^2) \times O(n^2)$ covariance matrix, which is large enough in most evolutionary clustering applications to make matrix inversion computationally infeasible. I present a simpler approach that involves a recursive update of the state estimates using only a single parameter α^t, which I

define in (4.8).

4.2.1 Smoothed proximity matrix

If the true proximity matrix Ψ^t is known, one would expect to see improved clustering results by performing static clustering on Ψ^t rather than on the current proximity matrix W^t because Ψ^t is free from noise. My objective is to accurately estimate Ψ^t at each time step. I can then perform static clustering on my estimate, which should also lead to improved clustering results.

The naïve approach of performing static clustering on W^t at each time step can be interpreted as using W^t itself as an estimate for Ψ^t. The main disadvantage of this approach is that it suffers from high variance due to the observation noise N^t. As a consequence, the obtained clustering results can be highly unstable and inconsistent with clustering results from adjacent time steps.

A better estimate can be obtained using the *smoothed proximity matrix* $\hat{\Psi}^t$ defined by

$$\hat{\Psi}^t = \alpha^t \hat{\Psi}^{t-1} + (1 - \alpha^t) W^t \tag{4.8}$$

for $t \geq 1$ and by $\hat{\Psi}^0 = W^0$. Notice that $\hat{\Psi}^t$ is a function of current and past data only, so it can be computed in an on-line setting. $\hat{\Psi}^t$ incorporates proximities not only from time $t-1$, but potentially from all previous time steps and allows us to suppress the observation noise. The parameter α^t controls the rate at which past proximities are forgotten; hence I refer to it as the *forgetting factor*. The forgetting factor in the proposed framework can change over time, allowing the amount of temporal smoothing to vary.

4.2.2 Shrinkage estimation of true proximity matrix

The smoothed proximity matrix $\hat{\Psi}^t$ is another natural candidate for estimating Ψ^t. It is a convex combination of two estimators: W^t and $\hat{\Psi}^{t-1}$. Since N^t is zero-mean, W^t is an unbiased estimator but has high variance because it uses only a single observation. $\hat{\Psi}^{t-1}$ is a

weighted combination of past observations so it should have lower variance than W^t, but it is likely to be biased since the past proximities may not be representative of the current ones as a result of long-term drift in the statistical properties of the objects. Thus the problem of estimating the optimal forgetting factor α^t may be considered as a bias-variance trade-off problem.

A similar bias-variance trade-off has been investigated in the problem of shrinkage estimation of covariance matrices (Ledoit and Wolf, 2003; Schäfer and Strimmer, 2005; Chen et al., 2010), where a shrinkage estimate of the covariance matrix is taken to be $\hat{\Sigma} = \lambda T + (1 - \lambda)S$, a convex combination of a suitably chosen target matrix T and the standard estimate, the sample covariance matrix S. Notice that the shrinkage estimate has the same form as the smoothed proximity matrix given by (4.8) where the smoothed proximity matrix at the previous time step $\hat{\Psi}^{t-1}$ corresponds to the shrinkage target T, the current proximity matrix W^t corresponds to the sample covariance matrix S, and α^t corresponds to the shrinkage intensity λ. I derive the optimal choice of α^t in a manner similar to the derivation of the optimal λ for shrinkage estimation of covariance matrices by Ledoit and Wolf (2003).

As in (Ledoit and Wolf, 2003; Schäfer and Strimmer, 2005; Chen et al., 2010), I choose to minimize the squared Frobenius norm of the difference between the true proximity matrix and the smoothed proximity matrix. That is, I take the loss function to be

$$L\left(\alpha^t\right) = \left\|\hat{\Psi}^t - \Psi^t\right\|_F^2 = \sum_{i=1}^{n}\sum_{j=1}^{n}\left(\hat{\psi}_{ij}^t - \psi_{ij}^t\right)^2.$$

I define the risk to be the conditional expectation of the loss function given all of the previous observations

$$R\left(\alpha^t\right) = \mathrm{E}\left[\left\|\hat{\Psi}^t - \Psi^t\right\|_F^2 \,\middle|\, W^{(t-1)}\right]$$

where $W^{(t-1)}$ denotes the set $\{W^{t-1}, W^{t-2}, \ldots, W^0\}$. Note that the risk function is differentiable and can be easily optimized if Ψ^t is known. However, Ψ^t is the quantity that I am

trying to estimate so it is not known. I first derive the optimal forgetting factor assuming it is known. I shall henceforth refer to this as the *oracle forgetting factor*.

Under the linear observation model of (4.7),

$$\mathrm{E}\left[W^t \middle| W^{(t-1)}\right] = \mathrm{E}\left[W^t\right] = \Psi^t \tag{4.9}$$

$$\mathrm{var}\left(W^t \middle| W^{(t-1)}\right) = \mathrm{var}\left(W^t\right) = \mathrm{var}\left(N^t\right) \tag{4.10}$$

because $N^t, N^{t-1}, \ldots, N^0$ are mutually independent and have zero mean. From the definition of $\hat{\Psi}^t$ in (4.8), the risk can then be expressed as

$$\begin{aligned} R\left(\alpha^t\right) &= \mathrm{E}\left[\left(\alpha^t \hat{\psi}_{ij}^{t-1} + \left(1-\alpha^t\right) w_{ij}^t - \psi_{ij}^t\right)^2 \middle| W^{(t-1)}\right] \\ &= \mathrm{var}\left(\alpha^t \hat{\psi}_{ij}^{t-1} + \left(1-\alpha^t\right) w_{ij}^t - \psi_{ij}^t \middle| W^{(t-1)}\right) \\ &\qquad + \mathrm{E}\left[\alpha^t \hat{\psi}_{ij}^{t-1} + \left(1-\alpha^t\right) w_{ij}^t - \psi_{ij}^t \middle| W^{(t-1)}\right]^2. \end{aligned} \tag{4.11}$$

(4.11) can be simplified using (4.9) and (4.10) and by noting that the conditional variance of $\hat{\psi}_{ij}^{t-1}$ is zero and that ψ_{ij}^t is deterministic. Thus

$$R\left(\alpha^t\right) = \sum_{i=1}^{n}\sum_{j=1}^{n}\left\{\left(1-\alpha^t\right)^2 \mathrm{var}\left(n_{ij}^t\right) + \left(\alpha^t\right)^2\left(\hat{\psi}_{ij}^{t-1} - \psi_{ij}^t\right)^2\right\}. \tag{4.12}$$

From (4.12), the first derivative is easily seen to be

$$R'\left(\alpha^t\right) = 2\sum_{i=1}^{n}\sum_{j=1}^{n}\left\{\left(\alpha^t - 1\right)\mathrm{var}\left(n_{ij}^t\right) + \alpha^t\left(\hat{\psi}_{ij}^{t-1} - \psi_{ij}^t\right)^2\right\}.$$

To determine the oracle forgetting factor $\left(\alpha^t\right)^*$, simply set $R'(\alpha^t) = 0$. Rearranging to

isolate α^t, I obtain

$$\left(\alpha^t\right)^* = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} \operatorname{var}\left(n_{ij}^t\right)}{\sum_{i=1}^{n}\sum_{j=1}^{n}\left\{\left(\hat{\psi}_{ij}^{t-1} - \psi_{ij}^t\right)^2 + \operatorname{var}\left(n_{ij}^t\right)\right\}}. \tag{4.13}$$

$(\alpha^t)^*$ does indeed minimize the risk because $R''(\alpha^t) \geq 0$ for all α^t.

The oracle forgetting factor $(\alpha^t)^*$ leads to the best estimate in terms of minimizing risk but is not implementable because it requires oracle knowledge of the true proximity matrix Ψ^t, which is what I am trying to estimate, as well as the noise variance $\operatorname{var}(N^t)$. It was suggested in (Schäfer and Strimmer, 2005) to replace the unknowns with their sample equivalents. In this setting, I would replace ψ_{ij}^t with the sample mean of w_{ij}^t and $\operatorname{var}(n_{ij}^t) = \operatorname{var}(w_{ij}^t)$ with the sample variance of w_{ij}^t. However, Ψ^t and potentially $\operatorname{var}(N^t)$ are time-varying so I cannot simply use the temporal sample mean and variance. Instead, I propose to use the sample mean and variance *over other proximities*. Since objects belong to clusters, it is reasonable to assume that the structure of Ψ^t and $\operatorname{var}(N^t)$ should reflect the cluster memberships. Hence I make an assumption about the structure of Ψ^t and $\operatorname{var}(N^t)$ in order to proceed.

4.2.3 Block model for true proximity matrix

I propose a block model for the true proximity matrix Ψ^t and $\operatorname{var}(N^t)$ and use the assumptions of this model to compute the desired sample means and variances. The assumptions of the block model are as follows:

1. $\psi_{ii}^t = \psi_{jj}^t$ for any two objects i, j that belong to the same cluster.
2. $\psi_{ij}^t = \psi_{lm}^t$ for any two distinct objects i, j and any two distinct objects l, m such that i, l belong to the same cluster, and j, m belong to the same cluster.

The structure of the true proximity matrix Ψ^t under these assumptions is shown in Fig. 4.1. $\psi_{(c)}^t$ denotes ψ_{ii}^t for all objects i in cluster c, and $\psi_{(cd)}^t$ denotes ψ_{ij}^t for all distinct objects i, j

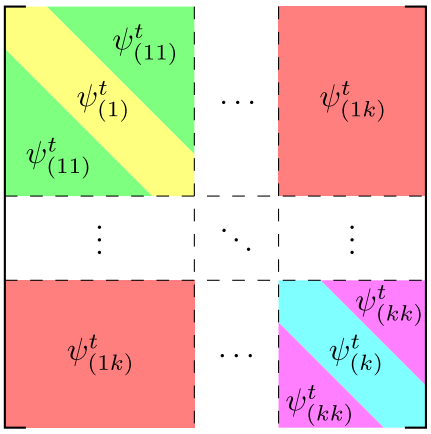


Figure 4.1: Block structure of true proximity matrix Ψ^t.

such that i is in cluster c and j is in cluster d. In short, I am assuming that the true proximity is equal inside the clusters and different between clusters. I make the same assumptions on $\mathrm{var}\,(N^t)$ that I do on Ψ^t, namely that it also possesses the assumed block structure.

One scenario where the block assumptions are completely satisfied is the case where the data at each time t are realizations from a dynamic Gaussian mixture model (GMM) (Carmi et al., 2009), which is described as follows. Assume that the k components of the dynamic GMM are parameterized by the k time-varying mean vectors $\{\boldsymbol{\mu}_c^t\}_{c=1}^k$ and covariance matrices $\{\Sigma_c^t\}_{c=1}^k$. Let $\{\phi_c\}_{c=1}^k$ denote the mixture weights. Objects are generated in the following manner:

1. (Only at $t = 0$) Draw n samples $\{z_i\}_{i=1}^n$ from the categorical distribution specified by $\{\phi_c\}_{c=1}^k$ to determine the component membership of each object.
2. (For all t) For each object i, draw a sample $\mathbf{x}_i^t$ from the Gaussian distribution parameterized by $\left(\boldsymbol{\mu}_{z_i}^t, \Sigma_{z_i}^t\right)$.

Notice that while the parameters of the individual components change over time, the component memberships do not, i.e. objects stay in the same components at all times.

The dynamic GMM is a model for clusters that move over time. In Appendix 4.A, I show that at each time t, the mean and variance of the dot product similarity matrix W^t, which correspond to Ψ^t and $\mathrm{var}\,(N^t)$ respectively under the observation model of (4.7), do indeed satisfy the assumed block structure. This scenario forms the basis of the experiment

in Section 4.4.1.

Although the proposed block model is rather simplistic, I believe that it is a reasonable choice when there is no prior information about the shapes of clusters. A nice feature of the block model is that it is *permutation invariant* with respect to the clusters; that is, it does not require objects to be ordered in any particular manner.

4.2.4 Adaptive estimation of forgetting factor

Under the block model assumption, the means and variances of proximities are identical in each block. As a result, one can sample over all proximities in a block to obtain sample means and variances. Unfortunately, the true true block structure is unknown because the cluster memberships are unknown.

To work around this problem, I estimate the cluster memberships along with $(\alpha^t)^*$ in an iterative fashion. First I initialize the cluster memberships. Two logical choices are to use the cluster memberships from the previous time step or the memberships obtained from performing static clustering on the current proximities. I then sample over each block to estimate the entries of Ψ^t and $\mathrm{var}\,(N^t)$ as detailed below, and substitute them into (4.13) to obtain an estimate $(\hat{\alpha}^t)^*$ of $(\alpha^t)^*$. Now substitute $(\hat{\alpha}^t)^*$ into (4.8) and perform static clustering on $\hat{\Psi}^t$ to obtain an updated clustering result. This clustering result is then used to refine the estimate of $(\alpha^t)^*$, and this iterative process is repeated to improve the quality of the clustering result. I find, empirically, that improvements are rarely seen after the third iteration.

To estimate the entries of $\Psi^t = \mathrm{E}\,[W^t]$, I proceed as follows. For two distinct objects i and j both in cluster c, I estimate ψ_{ij}^t using the sample mean

$$\widehat{\mathrm{E}}\left[w_{ij}^t\right] = \frac{1}{|c|\,(|c|-1)} \sum_{l \in c} \sum_{\substack{m \in c \\ m \neq l}} w_{lm}^t.$$

Similarly, I estimate ψ_{ii}^t by

$$\widehat{\mathrm{E}}\left[w_{ii}^t\right] = \frac{1}{|c|}\sum_{l\in c} w_{ll}^t.$$

For distinct objects i in cluster c and j in cluster d with $c \neq d$, I estimate ψ_{ij}^t by

$$\widehat{\mathrm{E}}\left[w_{ij}^t\right] = \frac{1}{|c||d|}\sum_{l\in c}\sum_{m\in d} w_{lm}^t.$$

$\mathrm{var}\left(N^t\right) = \mathrm{var}\left(W^t\right)$ can be estimated in a similar manner by taking unbiased sample variances over the blocks.

4.3 Evolutionary algorithms

From the derivation in Section 4.2.4, I obtain a generic algorithm for AFFECT, shown in Algorithm 4.4. This generic algorithm applies to any static clustering algorithm $\mathrm{cluster}(\cdot)$ that takes a similarity or dissimilarity matrix as input and returns a flat clustering result.

I provide some details and interpretation of this generic algorithm when used with three popular static clustering algorithms: agglomerative hierarchical clustering, k-means, and spectral clustering.

4.3.1 Agglomerative hierarchical clustering

The proposed evolutionary extension of agglomerative hierarchical clustering has an interesting interpretation in terms of the modified cost function defined in (4.6). Recall that agglomerative hierarchical clustering is a greedy algorithm that merges the two clusters with the lowest dissimilarity at each iteration. The dissimilarity between two clusters can be interpreted as the cost of merging them. Thus, performing agglomerative hierarchical clustering on $\hat{\Psi}^t$ results in merging the two clusters with the lowest modified cost at each iteration. The snapshot cost of a merge corresponds to the cost of making the merge at time t using the dissimilarities given by W^t. The temporal cost of a merge is a weighted combination of the costs of making the merge at each time step $s \in \{0, 1, \ldots, t-1\}$ using

Algorithm 4.4 Generic algorithm for AFFECT evolutionary clustering.

$\mathcal{C}^t \leftarrow \mathcal{C}^{t-1}$
for $i = 1, 2, \ldots$ **do**
  Compute $\widehat{\mathrm{E}}\left[W^t\right]$ and $\widehat{\mathrm{var}}\left(W^t\right)$ using $\mathcal{C}^t$
  Calculate $\left(\hat{\alpha}^t\right)^*$ by substituting estimates $\widehat{\mathrm{E}}\left[W^t\right]$ and $\widehat{\mathrm{var}}\left(W^t\right)$ into (4.13)
  $\hat{\Psi}^t \leftarrow \left(\hat{\alpha}^t\right)^* \hat{\Psi}^{t-1} + \left[1 - \left(\hat{\alpha}^t\right)^*\right] W^t$
  $\mathcal{C}^t \leftarrow \mathrm{cluster}(\hat{\Psi}^t)$
end for
return $\mathcal{C}^t$

the dissimilarities given by W^s. This can be seen by expanding the recursive update in (4.8) to obtain

$$\begin{aligned}\hat{\Psi}^t = \left(1-\alpha^t\right) W^t + \alpha^t\left(1-\alpha^{t-1}\right)W^{t-1} + \alpha^t\alpha^{t-1}\left(1-\alpha^{t-2}\right) W^{t-2} + \cdots \\ + \alpha^t\alpha^{t-1}\cdots\alpha^2\left(1-\alpha^1\right) W^1 + \alpha^t\alpha^{t-1}\cdots\alpha^2\alpha^1 W^0.\end{aligned} \tag{4.14}$$

4.3.2 k-means

k-means is an iterative clustering algorithm and requires an initial set of cluster memberships to begin the iteration. In static k-means, typically a random initialization is employed. A good initialization can significantly speed up the algorithm by reducing the number of iterations required for convergence. For evolutionary k-means, an obvious choice is to initialize using the clustering result at the previous time step. I use this initialization in our experiments in Section 4.4.

The proposed evolutionary k-means algorithm can also be interpreted as optimizing the modified cost function of (4.6). The snapshot cost is $\mathcal{D}\left(X^t, \mathcal{C}^t\right)$ where $\mathcal{D}(\cdot,\cdot)$ is the sum of squares cost defined in (4.1). The temporal cost is a weighted combination of $\mathcal{D}\left(X^t, \mathcal{C}^s\right), s \in \{0, 1, \ldots, t-1\}$, i.e. the cost of the clustering result applied to the data at time s. Hence the modified cost measures how well the current clustering result fits both current and past data.

4.3.3 Spectral clustering

The proposed evolutionary average association (AA) spectral clustering algorithm involves computing and discretizing eigenvectors of $\hat{\Psi}^t$ rather than W^t. It can also be interpreted in terms of the modified cost function of (4.6). Recall that the cost in static AA spectral clustering is $\mathrm{tr}\left(Z^T W Z\right)$. Performing AA spectral clustering on $\hat{\Psi}^t$ optimizes

$$\mathrm{tr}\left(Z^T \left[\sum_{s=0}^{t} \beta^s W^s\right] Z\right) = \sum_{s=0}^{t} \beta^s \,\mathrm{tr}\left(Z^T W^s Z\right), \tag{4.15}$$

where β^s corresponds to the coefficient in front of W^s in (4.14). Thus, the snapshot cost is simply $\mathrm{tr}\left(Z^T W^t Z\right)$ while the temporal cost corresponds to the remaining t terms in (4.15). I note that in the case where $\alpha^{t-1} = 0$, this modified cost is identical to that of PCQ, which incorporates historical data from time $t-1$ only. Hence the proposed generic framework reduces to PCQ in this special case.

Chi et al. (2007) noted that PCQ can easily be extended to accommodate longer history and suggested to do so by using an exponentially weighted forgetting factor. The proposed framework uses an adaptively weighted forgetting factor, which should improve clustering performance, especially if the rate at which the statistical properties of the data are evolving is time-varying.

Evolutionary ratio cut and normalized cut spectral clustering can be performed by forming the appropriate graph Laplacian, L^t or $\mathcal{L}^t$, respectively, using $\hat{\Psi}^t$ instead of W^t. They do not admit any obvious interpretation in terms of a modified cost function since they operate on L^t and $\mathcal{L}^t$ rather than W^t.

4.3.4 Practical issues

Adding and removing objects over time

Up to this point, I have assumed that the same objects are being observed at multiple time steps. In many application scenarios, however, new objects are often introduced over time

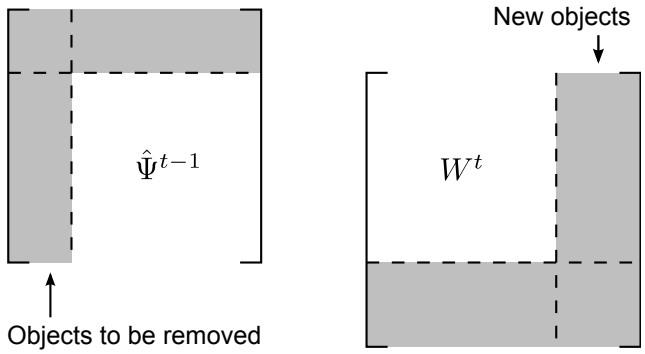


Figure 4.2: Adding and removing objects over time. Shaded rows and columns are to be removed before computing $\hat{\Psi}^t$. The rows and columns for the new objects are then appended to $\hat{\Psi}^t$.

while some existing objects may no longer be observed. In such a scenario, the indices of the proximity matrices W^t and $\hat{\Psi}^{t-1}$ correspond to different objects, so one cannot simply combine them as described in (4.8).

These types of scenarios can be dealt with in the following manner. Objects that were observed at time $t-1$ but not at time t can simply be removed from $\hat{\Psi}^{t-1}$ in (4.8). New objects introduced at time t have no corresponding rows and columns in $\hat{\Psi}^{t-1}$. These new objects can be naturally handled by adding rows and columns to $\hat{\Psi}^t$ after performing the smoothing operation in (4.8). In this way, the new nodes have no influence on the update of the forgetting factor α^t yet contribute to the clustering result through $\hat{\Psi}^t$. This process is illustrated graphically in Fig. 4.2.

Selecting the number of clusters

The task of optimally choosing the number of clusters at each time step is a difficult model selection problem that is beyond the scope of this chapter. However, since the proposed framework involves simply forming a smoothed proximity matrix followed by static clustering, heuristics used for selecting the number of clusters in static clustering can also be used with the proposed evolutionary clustering framework. One such heuristic applicable to many clustering algorithms is to choose the number of clusters to maximize the average silhouette width (Rousseeuw, 1987). For hierarchical clustering, selection of the number

of clusters is often accomplished using a stopping rule; a review of many such rules can be found in (Milligan and Cooper, 1985). The eigengap heuristic (von Luxburg, 2007) and the modularity criterion (Newman, 2006) are commonly used heuristics for spectral clustering. Such heuristics can be employed at each time step to choose the number of clusters, which can change over time.

Matching clusters between time steps

While evolutionary clustering provides a clustering result at each time that is consistent with past results, one still faces the challenge of matching clusters at time t with those at times $t-1$ and earlier. This requires permuting the clusters in the clustering result at time t. If the number of clusters k is relatively small, the optimal permutation can be found by enumerating all $k!$ possible permutations and choosing the one that maximizes agreement with previous clustering results; however, this is computationally infeasible for most applications.

A scalable heuristic approach for matching clusters is to use a greedy method. The clusters at time t and $t-1$ with the highest fraction of common nodes are matched, and this process is repeated until either all clusters at time t or at time $t-1$ have been exhausted. This approach was investigated by Dimitriadou et al. (2002) in the context of ensemble clustering and can be easily extended to match clustering results at time t with multiple previous time steps.

4.4 Experiments

I investigate the performance of the proposed AFFECT framework in four experiments involving both synthetic and real data sets. Tracking performance is measured in terms of the MSE $\mathrm{E}\left[\|\hat{\Psi}^t - \Psi^t\|_F^2\right]$, which is the criterion I seek to optimize. Clustering performance is measured by the Rand index (Rand, 1971), which is a quantity between 0 and 1 that indicates the amount of agreement between a clustering result and a set of labels, which

are taken to be the ground truth. A higher Rand index indicates higher agreement, with a Rand index of 1 corresponding to perfect agreement. I run at least one experiment for each of hierarchical clustering, k-means, and spectral clustering and compare the performance of AFFECT against the competing evolutionary clustering methods RG, PCQ, and PCM.

4.4.1 Well-separated Gaussians

This experiment is designed to test the tracking ability of AFFECT. I draw 40 samples equally from a mixture of two 2-D Gaussian distributions with mean vectors $(4, 0)$ and $(-4, 0)$ and with both covariance matrices equal to $0.1I$. At each time step, the means of the two distributions are moved according to a one-dimensional random walk in the first dimension with step size 0.1, and a new sample is drawn with the component memberships fixed, as described in Section 4.2.3. At time 19, I change the covariance matrices to $0.3I$ to test how well the framework can respond to a sudden change.

I run this experiment 100 times over 40 time steps using evolutionary k-means clustering. The two clusters are well-separated so even static clustering is able to correctly identify them. However the tracking performance is improved significantly by incorporating historical data, which can be seen in Fig. 4.3 where the MSE between the estimated and true similarity matrices is plotted for several choices of forgetting factor, including the estimated α^t. I also compare to the oracle α^t, which can be calculated using the true moments and cluster memberships of the data as shown in Appendix 4.A but is not implementable in a real application. Notice that the estimated α^t performs very well, and its MSE is very close to that of the oracle α^t. The estimated α^t also outperforms all of the constant forgetting factors.

The estimated α^t is plotted as a function of time in Fig. 4.4a. Since the clusters are well-separated, only a single iteration is performed to estimate α^t. Notice that both the oracle and estimated forgetting factors quickly increase from 0 then level off to a nearly constant value until time 19 when the covariance matrix is changed. After the transient due

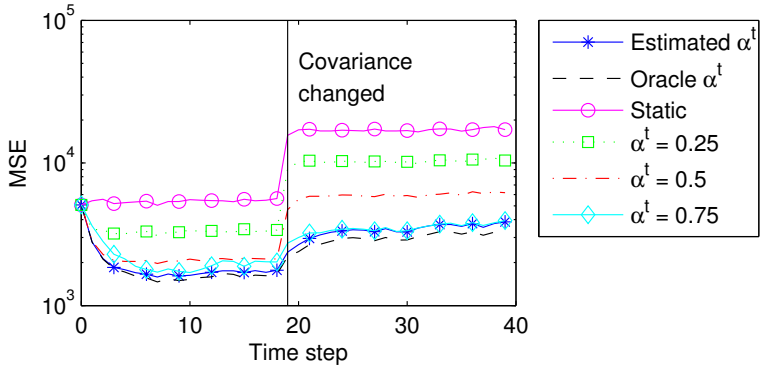
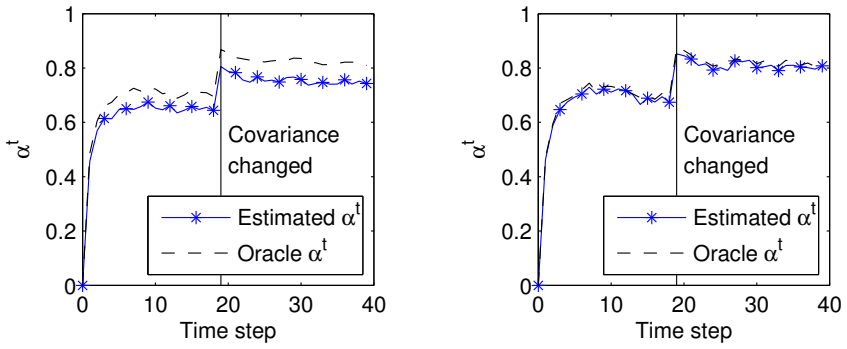


Figure 4.3: Comparison of MSE in well-separated Gaussians experiment. The estimated α^t performs best and approaches the MSE of the oracle α^t.



(a) 40 samples (b) 200 samples

Figure 4.4: Comparison of oracle and estimated forgetting factors in well-separated Gaussians experiment. The gap between the estimated and oracle forgetting factors decreases as the sample size increases.

to the change in covariance, both the oracle and estimated forgetting factors again level off. This behavior is to be expected because the two clusters are moving according to random walks. Notice that the estimated α^t does not converge to the same value the oracle α^t appears to. This bias is due to the finite sample size. The estimated and oracle forgetting factors are plotted in Fig. 4.4b for the same experiment but with 200 samples rather than 40. The gap between the steady-state values of the estimated and oracle forgetting factors is much smaller now, and it continues to decrease as the sample size increases.

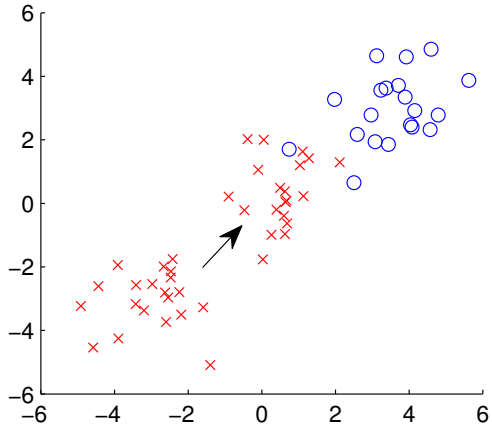


Figure 4.5: Setup of two colliding Gaussians experiment: one cluster is slowly moved toward the other, then a change in cluster membership is simulated.

4.4.2 Two colliding Gaussians

The objective of this experiment is to test the effectiveness of the AFFECT framework when a cluster moves close enough to another cluster so that they overlap. I also test the ability of the framework to adapt to a change in cluster membership.

The setup of this experiment is illustrated in Fig. 4.5. I draw 40 samples from a mixture of two 2-D Gaussian distributions, both with covariance matrix equal to identity. The mixture proportion (the proportion of samples drawn from the second cluster) is initially chosen to be $1/2$. The first cluster has mean $(3, 3)$ and remains stationary throughout the experiment. The second cluster's mean is initially at $(-3, -3)$ and is moved toward the first cluster from time steps 0 to 9 by $(0.4, 0.4)$ at each time. At times 10 and 11, I switch the mixture proportion to $3/8$ and $1/4$, respectively, to simulate objects changing cluster. From time 12 onwards, both the cluster means and mixture proportion are unchanged. At each time, I draw a new sample.

I run this experiment 100 times using evolutionary k-means clustering. The MSE in this experiment for varying α^t is shown in Fig. 4.6. As before, the oracle α^t is calculated using the true moments and cluster memberships and is not implementable in practice. It can be seen that the choice of α^t affects the MSE significantly. The estimated α^t performs the best, excluding the oracle α^t, which is not implementable. Notice also that $\alpha^t = 0.5$

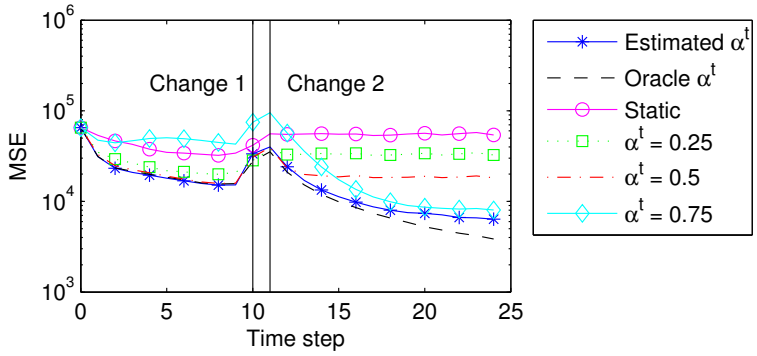


Figure 4.6: Comparison of MSE in two colliding Gaussians experiment. The estimated α^t performs best both before and after the change points.

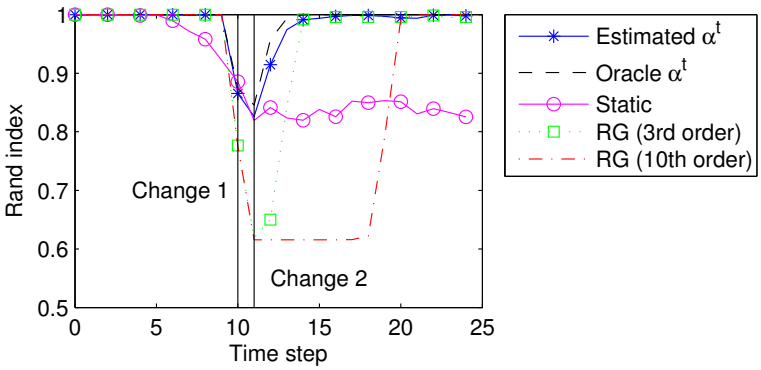


Figure 4.7: Comparison of Rand index in two colliding Gaussians experiment. The estimated α^t detects the changes in clusters quickly unlike the RG method.

performs well before the change in cluster memberships at time 10, i.e. when cluster 2 is moving, while $\alpha^t = 0.75$ performs better after the change when both clusters are stationary.

The clustering accuracy for this experiment is plotted in Fig. 4.7. Since this experiment involves k-means clustering, I compare to the RG method. I simulate two filter lengths for RG: a short-memory 3rd-order filter and a long-memory 10th-order filter. In Fig. 4.7 it can be seen that the estimated α^t also performs best in Rand index, approaching the performance of the oracle α^t. The static method performs poorly as soon as the clusters begin to overlap at around time step 7. All of the evolutionary methods handle the overlap well, but the RG method is slow to respond to the change in clusters, especially the long-memory filter. In Table 4.1, I present the mean and standard error (over the simulation runs) of the mean Rand indices of each method over all time steps. For AFFECT, I also show

Method	Parameters	Rand index
Static	-	0.897 ± 0.001
AFFECT	Estimated α^t	$\mathbf{0.982 \pm 0.001}$
	$\alpha^t = 0.5$	0.974 ± 0.001
RG	$l = 3$	0.954 ± 0.001
	$l = 10$	0.860 ± 0.001

Table 4.1: Mean and standard error of k-means Rand indices in two colliding Gaussians experiment. Bolded number indicates best performer.

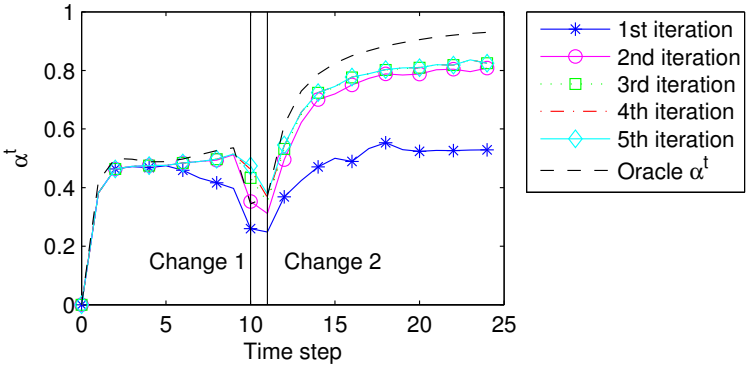


Figure 4.8: Comparison of oracle and estimated forgetting factors in two colliding Gaussians experiment. There is no noticeable improvement after the third iteration.

the performance of arbitrarily setting $\alpha^t = 0.5$, which also happens to outperform the RG method in this experiment. The poorer performance of the RG method is to be expected because it places more weight on time steps where the cluster centroids are well-separated, which again results in too much weight on historical data after the cluster memberships are changed.

The estimated α^t is plotted by iteration in Fig. 4.8 along with the oracle α^t. Notice that the estimate gets better over the first three iterations, while the fourth and fifth show no visible improvement. The plot of the suggests why the estimated α^t is able to outperform the constant α^t's. It is almost constant at the beginning of the experiment when the second cluster is moving, then it decreases over the two times when cluster memberships are changed, and finally it increases when the two clusters are both stationary. The values of the oracle α^t before and after the change corroborate the previous observation that $\alpha^t = 0.5$ performs well before the change, but $\alpha^t = 0.75$ performs better afterwards. Notice that the

estimated α^t appears to converge to a lower value than the oracle α^t. This is once again due to the finite-sample effect discussed in Section 4.4.1.

4.4.3 Flocks of boids

This experiment involves simulation of a natural phenomenon, namely the flocking behavior of birds. To simulate this phenomenon I use the bird-oid objects (boids) model proposed by Reynolds (1987). The boids model allows us to simulate natural movements of objects and clusters.

The behavior of the boids are governed by three main rules:

1. Boids try to fly towards the average position (centroid) of local flock mates.
2. Boids try to keep a small distance away from other boids.
3. Boids try to fly towards the average heading of local flock mates.

My implementation of the boids model is based on the pseudocode of Parker (2007). At each time step, I move each boid $1/100$ of the way towards the average position of local flock mates, double the distance between boids that are within 10 units of each other, and move each boid $1/8$ of the way towards the average heading.

I run two experiments using the boids model; one with a fixed number of flocks over time and one where the number of flocks varies over time.

Fixed number of flocks

Four flocks of 25 boids are initially distributed uniformly in separate $60 \times 60 \times 60$ cubes. To simulate boids moving continuously in time while being observed at regular time intervals, I allow each boid to perform five movements per time step according to the aforementioned rules. Similar to Reynolds (1987), I use goal setting to push the flocks along parallel paths. Note that unlike in the previous experiments, the flocking behavior makes it possible to simulate natural changes in cluster, simply by changing the flock membership of a boid. I

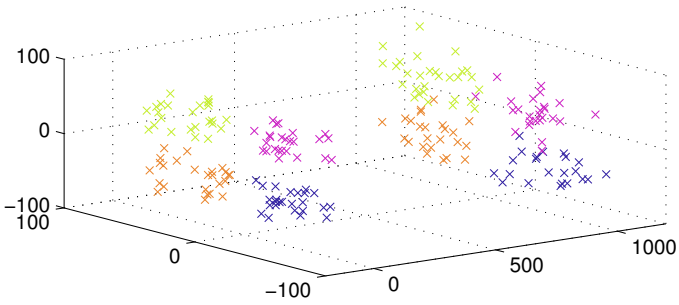


Figure 4.9: Setup of boids experiment: four flocks fly along parallel paths (start and end positions shown). At each time step, a randomly selected boid joins one of the other flocks.

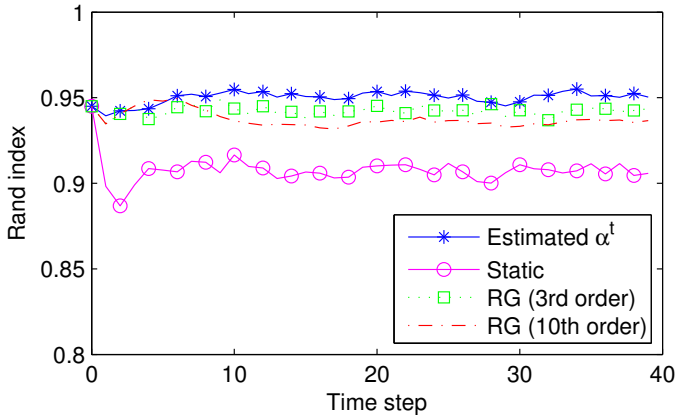


Figure 4.10: Comparison of complete linkage Rand index in boids experiment. The estimated α^t outperforms both static clustering and the RG method.

change the flock memberships of a randomly selected boid at each time step. The initial and final positions of the flocks for one realization are shown in Fig. 4.9.

I run this experiment 100 times using complete linkage hierarchical clustering. Unlike in the previous experiments, the true proximity matrix is unknown so MSE cannot be calculated. Clustering accuracy, however, can still be computed using the true flock memberships. The clustering performance of the various approaches is displayed in Fig. 4.10. Notice that AFFECT once again performs better than RG, both with short and long memory, although the difference is only visible after a significant number of boids have changed flocks. The mean and standard error of the Rand indices for the various methods are listed in Table 4.2. Again, it can be seen that AFFECT is the best performer. The estimated α^t in this experiment is roughly constant at around 0.6. This is not a surprise because all

Method	Parameters	Rand index
Static	-	0.908 ± 0.001
AFFECT	Estimated α^t	$\mathbf{0.950 \pm 0.001}$
	$\alpha^t = 0.5$	0.944 ± 0.001
RG	$l = 3$	0.942 ± 0.001
	$l = 10$	0.937 ± 0.000

Table 4.2: Mean and standard error of complete linkage Rand indices in boids experiment.

movements in this experiment, including changes in clusters, are smooth as a result of the flocking motions of the boids. This also explains the good performance of simply choosing $\alpha^t = 0.5$ in this particular experiment.

Variable number of flocks

The difference between this second boids experiment and the first is that the number of flocks changes over time in this experiment. Up to time 16, this experiment is identical to the previous one. At time 17, I simulate a scattering of the flocks by no longer moving them toward the average position of local flock mates as well as increasing the distance at which boids repel each other to 20 units. The boids are then rearranged at time 19 into two flocks rather than four.

I run this experiment 100 times. The RG framework cannot handle changes in the number of clusters over time, thus I switch to normalized cut spectral clustering and compare AFFECT to PCQ and PCM. The number of clusters at each time step is estimated using the modularity criterion (Newman, 2006). PCQ and PCM are not equipped with methods for selecting α. As a result, for each run of the experiment, I first perform a training run where the true flock memberships are used to compute the Rand index. The α which maximizes the Rand index is then used for the test run.

The clustering performance is shown in Fig. 4.11. The Rand indices for all methods drop after the flocks are scattered, which is to be expected. Shortly after the boids are rearranged into two flocks, the Rand indices improve once again as the flocks separate from each other. AFFECT once again outperforms the other methods, which can also be

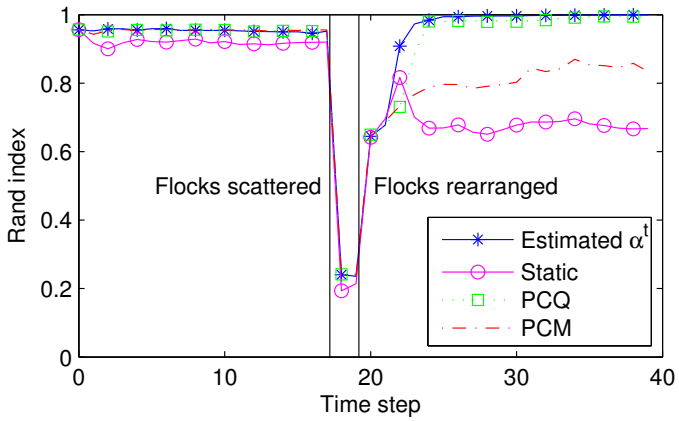


Figure 4.11: Comparison of spectral clustering Rand index in boids experiment. The estimated α^t and PCQ perform well, but PCM performs poorly.

seen from the summary statistics presented in Table 4.3. The performance of PCQ and PCM with both the trained α and arbitrarily chosen $\alpha = 0.5$ are listed. From Fig. 4.11, it can be seen that the estimated α^t responds quickest to the rearrangement of the flocks. The estimated forgetting factor by iteration is shown in Fig. 4.12. Notice that the estimated α^t drops when the flocks are scattered. Thus the quick response of AFFECT may be attributed to the adaptive forgetting factor. As before, the estimates of α^t do not appear to change after the third iteration. Unlike in the previous experiments, $\alpha^t = 0.5$ does not perform well in this experiment.

Another interesting observation is that the most accurate estimate of the number of clusters at each time is obtained when using AFFECT, as shown in Fig. 4.13. Prior to the flocks being scattered, using AFFECT, PCQ, or PCM all result in good estimates for the number of clusters, while using the static method results in overestimates. However, after the rearrangement of the flocks, the number of clusters is only accurately estimated when using AFFECT or PCQ, which partially contributes to the poorer Rand index of PCM in Fig. 4.11.

Method	Parameters	Rand index
Static	-	0.765 ± 0.001
AFFECT	Estimated α^t	$\mathbf{0.920 \pm 0.001}$
	$\alpha^t = 0.5$	0.872 ± 0.002
PCQ	Trained α	0.910 ± 0.001
	$\alpha = 0.5$	0.823 ± 0.001
PCM	Trained α	0.842 ± 0.002
	$\alpha = 0.5$	0.810 ± 0.001

Table 4.3: Mean and standard error of spectral clustering Rand indices in boids experiment.

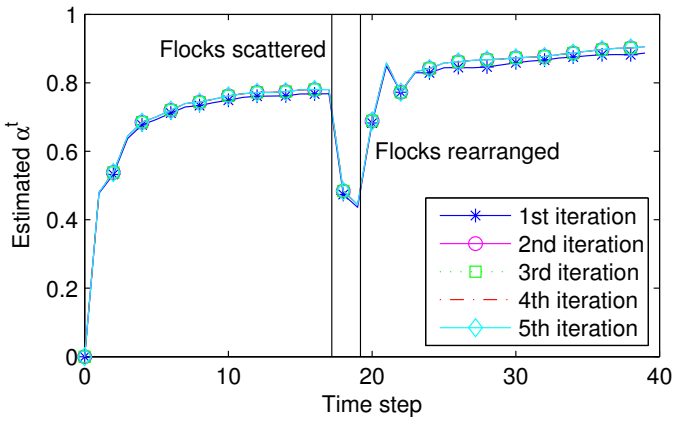


Figure 4.12: Comparison of estimated spectral clustering forgetting factor by iteration in boids experiment. The estimated forgetting factor drops at the change point, i.e. when the flocks are scattered.

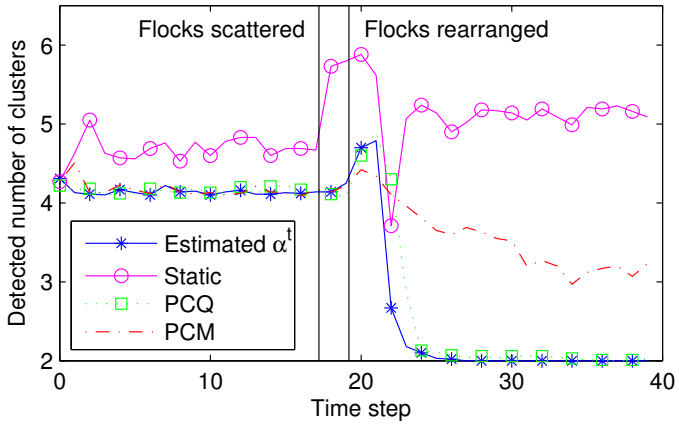


Figure 4.13: Comparison of number of clusters detected using the modularity criterion in boids experiment. Using the estimated α^t results in the best estimates of the number of flocks (4 before the change point and 2 after).

4.4.4 Reality Mining

The objective of this experiment is to test the proposed framework on a real data set with objects entering and leaving at different time steps. The experiment is conducted on the MIT Reality Mining data set (Eagle et al., 2009). The data was collected by recording cell phone activity of 94 students and staff at MIT over a year. Each phone recorded the Media Access Control (MAC) addresses of nearby Bluetooth devices at five-minute intervals. Using this device proximity data, I construct a similarity matrix where the similarity between two students corresponds to the number of intervals where they were in physical proximity. I divide the data into time steps of one week, resulting in 46 time steps between August 2004 and June 2005. In this data set there is partial ground truth. Namely the affiliations of each participant are available. Eagle et al. (2009) found that two dominant clusters could be identified from the Bluetooth proximity data, corresponding to new students at the Sloan business school and coworkers who work in the same building. The affiliations are likely to be representative of the cluster structure, at least during the school year.

I perform normalized cut spectral clustering into two clusters for this experiment. Since this experiment involves real data, I cannot simulate training sets to select α for PCQ and PCM. Instead, I use 2-fold cross-validation, which I believe is the closest substitute. A comparison of clustering performance is given in Table 4.4. Both the mean Rand indices over the entire 46 weeks and only during the school year are listed. AFFECT is the best performer in both cases. Surprisingly, PCQ and PCM barely outperform static spectral clustering when the cross-validated α is used and even worse than static clustering when $\alpha = 0.5$ is used. I believe this is due to the way PCQ and PCM suboptimally handle objects entering and leaving at different time steps by estimating previous similarities and memberships, respectively. On the contrary, the method used by AFFECT, described in Section 4.3.4, performs well even with objects entering and leaving over time.

The estimated α^t is shown in Fig. 4.14. Six important dates are labeled. The start and end dates of the terms were taken from the MIT academic calendar (MIT–WWW) to

Method	Parameters	Rand index	
		Entire trace	School year
Static	-	0.852	0.905
AFFECT	Estimated α^t	**0.891**	**0.953**
	$\alpha^t = 0.5$	0.884	0.949
PCQ	Cross-validated α	0.854	0.908
	$\alpha = 0.5$	0.767	0.822
PCM	Cross-validated α	0.855	0.922
	$\alpha = 0.5$	0.552	0.532

Table 4.4: Mean spectral clustering Rand indices for Reality Mining experiment.

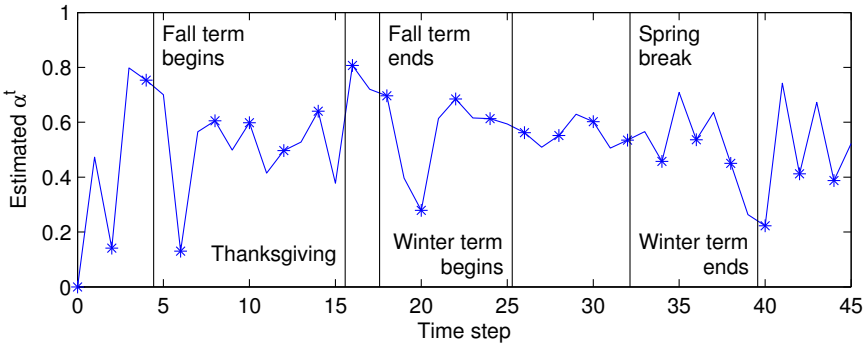


Figure 4.14: Estimated α^t over entire Reality Mining data trace. Six important dates are indicated. The sudden drops in the estimated α^t indicate change points in the network.

be the first and last day of classes, respectively. Notice that the estimated α^t appears to drop around several of these dates. These drops suggest that physical proximities changed around these dates, which is reasonable, especially for the students because their physical proximities depend on their class schedules. For example, the similarity matrices at time steps 18 and 19, before and after the beginning of winter break, are shown in Fig. 4.15. The detected clusters using the estimated α^t are superimposed onto both matrices, with rows and columns permuted according to the clusters. The empty cluster in the upper left consists of inactive students. Notice that the similarities, corresponding to time spent in physical proximity of other students, are much lower at time 19, particularly in the smaller cluster. The change in the structure of the similarity matrix, along with the knowledge that the fall term ended and the winter break began around this time, suggests that the low estimated forgetting factor at time 19 is appropriate.

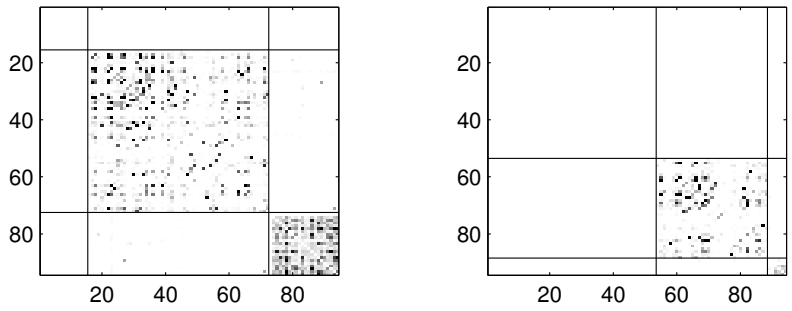


Figure 4.15: Cluster structure before (left) and after (right) beginning of winter break in Reality Mining data trace. Darker entries correspond to greater time spent in physical proximity. The empty cluster to the upper left consists of inactive participants during the time step.

4.5 Summary

In this chapter I proposed a novel adaptive framework for evolutionary clustering by performing tracking followed by static clustering. The objective of the framework was to accurately track the true proximity matrix at each time step. This was accomplished using a recursive update with an adaptive forgetting factor that controlled the amount of weight to apply to historic data. I proposed a method for estimating the optimal forgetting factor in order to minimize mean squared tracking error. The main advantages of my approach are its universality, allowing almost any static clustering algorithm to be extended to an evolutionary one, and that it provides an explicit method for selecting the forgetting factor, unlike existing methods. The proposed framework was evaluated on several synthetic and real data sets and displayed good performance in tracking and clustering. It was able to outperform both static clustering algorithms and existing evolutionary clustering algorithms.

4.A True similarity matrix for dynamic Gaussian mixture model

I derive the true similarity matrix Ψ and the matrix of variances of similarities $\mathrm{var}(W)$, where the similarity is taken to be the dot product, for data sampled from the dynamic

Gaussian mixture model described in Section 4.2.3. These matrices are required in order to calculate the oracle forgetting factor for the experiments in Sections 4.4.1 and 4.4.2. I drop the superscript t to simplify the notation.

Consider two arbitrary objects $\mathbf{x}_i \sim N(\boldsymbol{\mu}_c, \Sigma_c)$ and $\mathbf{x}_j \sim N(\boldsymbol{\mu}_d, \Sigma_d)$ where the entries of $\boldsymbol{\mu}_c$ and Σ_c are denoted by μ_{ck} and σ_{ckl}, respectively. For any distinct i, j the mean is

$$\mathrm{E}\left[\mathbf{x}_i \mathbf{x}_j^T\right] = \sum_{k=1}^{p} \mathrm{E}\left[x_{ik} x_{jk}\right] = \sum_{k=1}^{p} \mu_{ck}\mu_{dk},$$

and the variance is

$$\begin{aligned}
\mathrm{var}\left(\mathbf{x}_i \mathbf{x}_j^T\right) &= \mathrm{E}\left[\left(\mathbf{x}_i \mathbf{x}_j^T\right)^2\right] - \mathrm{E}\left[\mathbf{x}_i \mathbf{x}_j^T\right]^2 \\
&= \sum_{k=1}^{p}\sum_{l=1}^{p} \left\{\mathrm{E}\left[x_{ik} x_{jk} x_{il} x_{jl}\right] - \mu_{ck}\mu_{dk}\mu_{cl}\mu_{dl}\right\} \\
&= \sum_{k=1}^{p}\sum_{l=1}^{p} \left\{\left(\sigma_{ckl} + \mu_{ck}\mu_{cl}\right)\left(\sigma_{dkl} + \mu_{dk}\mu_{dl}\right) - \mu_{ck}\mu_{dk}\mu_{cl}\mu_{dl}\right\} \\
&= \sum_{k=1}^{p}\sum_{l=1}^{p} \left\{\sigma_{ckl}\sigma_{dkl} + \sigma_{ckl}\mu_{dk}\mu_{dl} + \sigma_{dkl}\mu_{ck}\mu_{cl}\right\}
\end{aligned}$$

by independence of $\mathbf{x}_i$ and $\mathbf{x}_j$. This holds both for $\mathbf{x}_i, \mathbf{x}_j$ in the same cluster, i.e. $c = d$, and for $\mathbf{x}_i, \mathbf{x}_j$ in different clusters, i.e. $c \neq d$. Along the diagonal,

$$\mathrm{E}\left[\mathbf{x}_i \mathbf{x}_i^T\right] = \sum_{k=1}^{p} \mathrm{E}\left[x_{ik}^2\right] = \sum_{k=1}^{p} \left(\sigma_{ckk} + \mu_{ck}^2\right).$$

The calculation for the variance is more involved. I first note that

$$\mathrm{E}\left[x_{ik}^2 x_{il}^2\right] = \mu_{ck}^2\mu_{cl}^2 + \mu_{ck}^2\sigma_{cll} + 4\mu_{ck}\mu_{cl}\sigma_{ckl} + \mu_{cl}^2\sigma_{ckk} + 2\sigma_{ckl}^2 + \sigma_{ckk}\sigma_{cll},$$

which can be derived from the characteristic function of the multivariate Gaussian distri-

bution (Anderson, 2003). Thus

$$\begin{aligned}\operatorname{var}\left(\mathbf{x}_i\mathbf{x}_i^T\right) &= \sum_{k=1}^{p}\sum_{l=1}^{p}\left\{\mathrm{E}\left[x_{ik}^2x_{il}^2\right] - \left(\sigma_{ckk}+\mu_{ck}^2\right)\left(\sigma_{cll}+\mu_{cl}^2\right)\right\} \\ &= \sum_{k=1}^{p}\sum_{l=1}^{p}\left\{4\mu_{ck}\mu_{cl}\sigma_{ckl}+2\sigma_{ckl}^2\right\}.\end{aligned}$$

The calculated means and variances are then substituted into (4.13) to compute the oracle forgetting factor. Since the expressions for the means and variances depend only on the clusters and not any objects in particular, it is confirmed that both Ψ and $\operatorname{var}(W)$ do indeed possess the assumed block structure discussed in Section 4.2.3.

CHAPTER V

STATE-SPACE MODELS FOR DYNAMIC NETWORKS

In Section 4.2, I introduced the concept of a *network state* Ψ^t along with the linear observation model $W^t = \Psi^t + N^t$, where N^t is a zero-mean noise matrix. Such a model assumes a state for each pair of nodes, and as mentioned in Section 4.2, it presents a computational problem when it comes to tracking the states in an optimal manner due to the need to invert an $O(|V^t|^2) \times O(|V^t|^2)$ matrix, where $|V^t|$ denotes the number of nodes at time t. In this chapter, I explore methods for modeling a dynamic network using much fewer than $O(|V^t|^2)$ states, which enables us to use more sophisticated, near-optimal methods for tracking and prediction.

Specifically I propose a state-space model for dynamic networks that combines two types of models: a *static model* for the individual network snapshots and a *temporal model* for the state evolution. Statistical models for static networks have a long history in statistics and sociology among other fields. A survey of the literature in statistical network models can be found in Goldenberg et al. (2010). Several of these models are of particular interest because they extend very naturally to the dynamic setting.

1. Exponential random graph models (ERGMs), also known as p^* models, attempt to replicate certain network statistics such as the number of edges, stars, and triangles (Wasserman and Pattison, 1996; Robins et al., 2007).

2. Latent space models (LSMs) represent nodes in a low-dimensional space where the distance between nodes and the values of node- or dyad-specific attributes determine the probability of forming an edge between a pair of nodes (Hoff et al., 2002).

3. Stochastic blockmodels (SBMs) divide nodes in the network into multiple classes and

generate edges independently with probabilities dependent on the class memberships of the nodes (Holland et al., 1983; Nowicki and Snijders, 2001; Airoldi et al., 2008).

In this chapter, I propose a state-space SBM that extends the SBM to the dynamic setting. Using a Central Limit Theorem approximation, I develop a near-optimal on-line inference procedure using the extended Kalman filter. I apply the proposed procedure to perform tracking and prediction on several simulated and real dynamic networks.

5.1 Related work

There has been previous work on extending each of the three static models mentioned in the introduction to dynamic networks. For ERGMs, temporal extensions include the hidden temporal ERGM (htERGM) (Guo et al., 2007) and the TESLA model (Ahmed and Xing, 2009). For LSMs, Sarkar and Moore (2005) propose a temporal extension using multidimensional scaling (MDS) with a Procrustes transform to track node positions over time. Westveld and Hoff (2011) propose a temporal extension along with a Markov chain Monte Carlo (MCMC) inference method.

More closely related to the state-space model I propose are several temporal extensions of SBMs. Xing et al. (2010) propose a temporal extension of the mixed-membership SBM (Airoldi et al., 2008) involving a linear state-space model for a single prior distribution of class memberships for all nodes. Ho et al. (2011) extend this model to multiple prior distributions corresponding to clusters of nodes, where each node belongs to a single cluster. Yang et al. (2011) propose a dynamic SBM involving a transition matrix that specifies the probability that a node in class i at time t switches to class j at time $t+1$ for all i, j, t and fit the model using a combination of Gibbs sampling and simulated annealing, which they refer to as probabilistic simulated annealing (PSA). This model is similar to the one I propose; hence the performance of the PSA algorithm serves as a good baseline for comparison with the inference procedure proposed in this chapter.

5.2 Stochastic blockmodels for static networks

I first introduce some notation and provide a summary of the static stochastic blockmodel (SSBM), which I use as the static model for the individual network snapshots. I represent a dynamic network by a time-indexed sequence of graph snapshots, with $W^t = [w_{ij}^t]$ denoting the adjacency matrix of the graph observed at time step t. $w_{ij}^t = 1$ if there is an edge from node i to node j at time t, and $w_{ij}^t = 0$ otherwise. Unless otherwise specified, I assume that the graphs are directed, so that $w_{ij}^t \neq w_{ji}^t$ in general. I assume for simplicity that there are no self-edges, i.e. $w_{ii}^t = 0$. $W^{(s)}$ denotes the set of all snapshots up to time s, $\{W^s, W^{s-1}, \ldots, W^1\}$. The notation $i \in a$ is used to indicate that node i is a member of class a. $|a|$ denotes the number of nodes in class a. The classes of all nodes at time t is given by a vector $\mathbf{c}^t$ with $c_i^t = a$ if $i \in a$ at time t. I denote the submatrix of W^t corresponding to the relations between nodes in class a and class b by $W_{[a][b]}^t$. I denote the vectorized equivalent of a matrix X, i.e. the vector obtained by simply stacking columns of X on top of one another, by $\mathbf{x}$. Doubly-indexed subscripts such as x_{ij} denote entries of matrix x, while singly-indexed subscripts such as x_i denote entries of vector $\mathbf{x}$.

Consider a snapshot at an arbitrary time step t. An SSBM is parameterized by a $k \times k$ matrix $\Theta^t = [\theta_{ab}^t]$, where θ_{ab}^t denotes the probability of forming an edge between a node in class a and a node in class b, and k denotes the number of classes. It decomposes the adjacency matrix into k^2 blocks, where each block is associated with relations between nodes in two classes a and b. Each block corresponds to a submatrix $W_{[a][b]}^t$ of the adjacency matrix W. Thus, given the class membership vector $\mathbf{c}^t$, each entry of W^t is an independent realization of a Bernoulli random variable with a block-dependent parameter; that is, $w_{ij}^t \sim$ Bernoulli $\left(\theta_{c_i^t c_j^t}^t\right)$.

SBMs are used in two settings:

1. the *a priori* blockmodeling setting, where class memberships are known or assumed, and the objective is to estimate the matrix of edge probabilities Θ^t, and
2. the *a posteriori* blockmodeling setting, where the objective is to simultaneously esti-

mate Θ^t and the class membership vector $\mathbf{c}^t$.

Since each entry of W^t is independent, the likelihood for the SBM is given by

$$\begin{aligned} f\left(W^t; \Phi^t\right) &= \prod_{i \neq j} \left(\theta^t_{c_i c_j}\right)^{w^t_{ij}} \left(1 - \theta^t_{c_i c_j}\right)^{1 - w^t_{ij}} \\ &= \exp\left\{ \sum_{a=1}^{k} \sum_{b=1}^{k} \left[m^t_{ab} \log\left(\theta^t_{ab}\right) + \left(n^t_{ab} - m^t_{ab}\right) \log\left(1 - \theta^t_{ab}\right) \right] \right\}, \end{aligned} \tag{5.1}$$

where

$$\begin{aligned} m^t_{ab} &= \sum_{i \in a} \sum_{j \in b} w^t_{ij} \\ n^t_{ab} &= \begin{cases} |a||b| & a \neq b \\ |a|(|a| - 1) & a = b. \end{cases} \end{aligned} \tag{5.2}$$

The parameters are given by $\Phi^t = \Theta^t$ in the a priori setting, and $\Phi^t = \{\Theta^t, \mathbf{c}^t\}$ in the a posteriori setting. In the a priori setting, a sufficient statistic for estimating Θ^t is given by the matrix Y^t of block densities (ratio of observed edges to possible edges within a block) defined by

$$y^t_{ab} = \frac{m^t_{ab}}{n^t_{ab}}. \tag{5.3}$$

Y^t also happens to be the maximum-likelihood (ML) estimate of Θ^t, which can be shown by setting the derivative of the logarithm of (5.1) to 0.

Estimation in the a posteriori setting is more involved, and many methods have been proposed, including expectation-maximization (EM) (Snijders and Nowicki, 1997), Gibbs sampling (Snijders and Nowicki, 1997; Nowicki and Snijders, 2001), label-switching methods (Karrer and Newman, 2011; Zhao et al., 2011), and spectral clustering (Rohe et al., 2011; Sussman et al., 2012). The label-switching methods use a heuristic for solving the combinatorial optimization problem of maximizing the likelihood (5.1) over the set of possible class memberships, which is often too large to perform an exhaustive search. On the

other hand, spectral clustering attempts to discover the optimal choice of class memberships using the eigenvectors of W^t or a similar matrix. Each node is mapped to a k-dimensional position in the eigenspace, and the estimated class memberships are obtained by clustering the nodes based on their positions in the eigenspace using, for example, k-means clustering.

5.3 State-space stochastic blockmodels for dynamic networks

I propose a state-space model for dynamic networks that consists of a temporal extension of the static stochastic blockmodel. First I present the model and inference procedure for a priori blockmodeling, then I discuss the additional steps necessary for a posteriori blockmodeling. The inference procedure is on-line, i.e. the state estimate at time t is formed using only observations from time t and earlier.

5.3.1 A priori blockmodeling

In the a priori SSBM setting, Y^t is a sufficient statistic for estimating Θ^t as discussed in Section 5.2. Thus in the a priori dynamic SBM setting, I can equivalently treat Y^t as the observation rather than W^t. By the Central Limit Theorem, y_{ab}^t is approximately Gaussian distributed with mean θ_{ab}^t and variance

$$\left(\sigma_{ab}^t\right)^2 = \frac{\theta_{ab}^t(1-\theta_{ab}^t)}{n_{ab}^t}, \tag{5.4}$$

where n_{ab}^t was defined in (5.2). I assume that y_{ab}^t is indeed Gaussian for all (a,b) and posit the linear observation model

$$Y^t = \Theta^t + Z^t, \tag{5.5}$$

where Z^t is a zero-mean independent and identically distributed (iid) Gaussian noise matrix with variance $\left(\sigma_{ab}^t\right)^2$ for the (a,b)th entry.

In the dynamic network setting where historical snapshots are indeed available, the

observations would be given by the set $Y^{(t)}$. The set $\Theta^{(t)}$ can be viewed as states of a dynamic system that is generating the noisy observation sequence. I complete the model by specifying a model for the state evolution over time. Since θ_{ab}^t is a probability and must be bounded between 0 and 1, I instead work with the matrix $\Psi^t = [\psi_{ab}^t]$ where $\psi_{ab}^t = \log(\theta_{ab}^t) - \log(1-\theta_{ab}^t)$, the logit of θ_{ab}^t. A simple model for the state evolution is the random walk model

$$\boldsymbol{\psi}^t = \boldsymbol{\psi}^{t-1} + \mathbf{v}^t,$$

where $\boldsymbol{\psi}^t$ is the vector representation of the matrix Ψ^t, and $\mathbf{v}^t$ is a random vector of zero-mean Gaussian entries, commonly referred to as process noise, with covariance matrix Γ. The entries of the process noise vector are not necessarily independent or identically distributed (unlike the entries of Z^t) to allow for states to evolve in a correlated manner. The observation equation (5.5) can then be written in terms of $\boldsymbol{\psi}^t$ as

$$\mathbf{y}^t = h\left(\boldsymbol{\psi}^t\right) + \mathbf{z}^t, \tag{5.6}$$

where I have converted the block densities Y^t and observation noise Z^t to their respective vector representations, and the function $h : \mathbb{R}^p \to \mathbb{R}^p$ is defined by $h_i(x) = 1/(1+e^{-x})$, i.e. the logistic function applied to each entry of X. I denote the covariance matrix of $\mathbf{z}^t$ by Σ^t, which is a diagonal matrix with entries given by (5.4)[1]. A graphical representation of the proposed model for the dynamic network is shown in Figure 5.1. The rectangular boxes denote observed quantities, and the ovals denote unobserved quantities.

To perform inference on this model, I assume the initial state $\boldsymbol{\psi}^0 \sim \mathcal{N}\left(\boldsymbol{\mu}^0, \Gamma^0\right)$ and that $\{\boldsymbol{\psi}^0, \mathbf{v}^1, \ldots, \mathbf{v}^t, \mathbf{z}^0, \ldots, \mathbf{z}^t\}$ are mutually independent. If (5.6) was linear in $\boldsymbol{\psi}^t$, then the optimal estimate of $\boldsymbol{\psi}^t$ in terms of minimum mean-squared error (MMSE) would be given by the Kalman filter. Due to the non-linearity, I apply the extended Kalman filter (EKF),

[1]The indices (a, b) for σ_{ab}^2 are converted into a single index i corresponding to the vector representation $\mathbf{z}^t$.

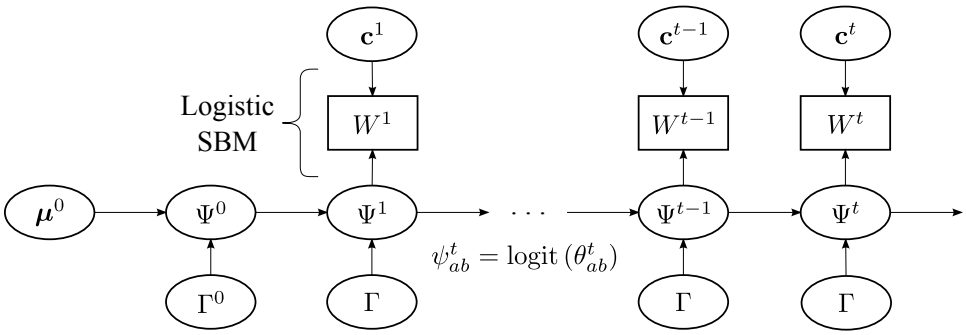


Figure 5.1: Graphical representation of proposed model for the dynamic network. The logistic SBM refers to applying the logistic function to each entry of Ψ^t to obtain Θ^t then generating W^t using Θ^t and $\mathbf{c}^t$.

which linearizes the dynamics about the predicted state and provides an approximately optimal estimate of $\boldsymbol{\psi}^t$. The Jacobian of h evaluated at the predicted state is a diagonal matrix with (i,i)th entry given by

$$h^t_{ii} = \frac{\exp\left(-\hat{\boldsymbol{\psi}}_i^{t|t-1}\right)}{\left[1+\exp\left(-\hat{\boldsymbol{\psi}}_i^{t|t-1}\right)\right]^2}.$$

The linearized model is thus given by the equations

$$\mathbf{y}^t = H^t\boldsymbol{\psi}^t + \mathbf{e}^t + \mathbf{z}^t \tag{5.7}$$

$$\boldsymbol{\psi}^t = \boldsymbol{\psi}^{t-1} + \mathbf{v}^t \tag{5.8}$$

where $\mathbf{e}^t = h\left(\hat{\boldsymbol{\psi}}^{t|t-1}\right) - H^t\hat{\boldsymbol{\psi}}^{t|t-1}$. For the model specified by (5.7) and (5.8), the EKF prediction equations are as follows (Haykin, 2001):

Prior state estimate: $$\hat{\boldsymbol{\psi}}^{t|t-1} = \hat{\boldsymbol{\psi}}^{t-1|t-1} \tag{5.9}$$

Prior estimate covariance: $$R^{t|t-1} = R^{t|t-1} + \Gamma \tag{5.10}$$

The EKF correction (update) equations are as follows (Haykin, 2001):

$$\text{Near-optimal Kalman gain:} \quad K^t = R^{t|t-1}\left(H^t\right)^T\left[H^t R^{t|t-1}\left(H^t\right)^T + \Sigma^t\right]^{-1} \tag{5.11}$$

$$\text{Posterior state estimate:} \quad \hat{\boldsymbol{\psi}}^{t|t} = \hat{\boldsymbol{\psi}}^{t|t-1} + K^t\left[\mathbf{y}^t - h\left(\hat{\boldsymbol{\psi}}^{t|t-1}\right)\right] \tag{5.12}$$

$$\text{Posterior estimate covariance:} \quad R^{t|t} = \left(I - K^t H^t\right) R^{t|t-1} \tag{5.13}$$

Using equations (5.9)–(5.13), a near-optimal prediction $\hat{\boldsymbol{\psi}}^{t|t-1}$ and corrected estimate $\hat{\boldsymbol{\psi}}^{t|t}$ can be obtained at each time step.

5.3.2 A posteriori blockmodeling

In many applications, the class memberships $\mathbf{c}^t$ are not known a priori and must be estimated along with Ψ^t. This can be done using label-switching methods similar to Karrer and Newman (2011) and Zhao et al. (2011); however, rather than maximizing the likelihood, I maximize the posterior state density given the entire sequence of observations $W^{(t)}$ up to time t.

For $t \geq 2$, the posterior state density is given by

$$\begin{aligned} f\left(\boldsymbol{\psi}^t \,|\, W^{(t)}\right) &\propto f\left(W^t \,|\, \boldsymbol{\psi}^t, W^{(t-1)}\right) f\left(\boldsymbol{\psi}^t \,|\, W^{(t-1)}\right) \\ &= f\left(W^t \,|\, \boldsymbol{\psi}^t\right) f\left(\boldsymbol{\psi}^t \,|\, W^{(t-1)}\right), \end{aligned} \tag{5.14}$$

where equality follows from the conditional independence of current and past observations given the current state. Since $\boldsymbol{\theta}^t = h(\boldsymbol{\psi}^t)$, the first term in (5.14) is simply obtained by substituting $h(\boldsymbol{\psi}^t)$ for $\boldsymbol{\theta}^t$ in (5.1). The second term in (5.14) is equivalent to $f\left(\boldsymbol{\psi}^t \,|\, \mathbf{y}^{(t-1)}\right)$ because the class memberships at all previous time steps have already been estimated. By applying the Kalman filter to the linearized model specified by (5.7) and (5.8), $f\left(\boldsymbol{\psi}^t \,|\, \mathbf{y}^{(t-1)}\right) \sim \mathcal{N}\left(\hat{\boldsymbol{\psi}}^{t|t-1}, R^{t|t-1}\right)$. Thus the logarithm of the posterior density for

$t \geq 2$ is given by

$$\begin{aligned}\log f\left(\boldsymbol{\psi}^t \mid W^{(t)}\right) = c - \frac{1}{2}\left(\boldsymbol{\psi}^t - \hat{\boldsymbol{\psi}}^{t|t-1}\right)^T \left(R^{t|t-1}\right)^{-1} \left(\boldsymbol{\psi}^t - \hat{\boldsymbol{\psi}}^{t|t-1}\right) \\ + \sum_{a=1}^{k}\sum_{b=1}^{k}\left\{m_{ab}^t \log\left[h\left(\psi_{ab}^t\right)\right] + \left(n_{ab}^t - m_{ab}^t\right)\log\left[1 - h\left(\psi_{ab}^t\right)\right]\right\},\end{aligned} \tag{5.15}$$

where c is a constant term independent of $\boldsymbol{\psi}^t$ that can be ignored.

At the initial time step $t = 1$, the posterior is given by

$$f\left(\boldsymbol{\psi}^1 \mid W^{(1)}\right) = f\left(\boldsymbol{\psi}^1 \mid W^1\right) \propto f\left(W^1 \mid \boldsymbol{\psi}^1\right) f\left(\boldsymbol{\psi}^1\right). \tag{5.16}$$

The first term is obtained again by substituting $h(\boldsymbol{\psi}^t)$ for $\boldsymbol{\theta}^t$ in (5.1). Since I assume the initial state $\boldsymbol{\psi}^0 \sim \mathcal{N}\left(\boldsymbol{\mu}^0, \Gamma^0\right)$, it follows from the state evolution model (5.8) that $\boldsymbol{\psi}^1 \sim \mathcal{N}\left(\boldsymbol{\mu}^0, \Gamma^0 + \Gamma\right)$. Thus the logarithm of the posterior density for $t = 1$ is given by

$$\begin{aligned}\log f\left(\boldsymbol{\psi}^1 \mid W^{(1)}\right) = d - \frac{1}{2}\left(\boldsymbol{\psi}^1 - \boldsymbol{\mu}^0\right)^T \left(\Gamma^0 + \Gamma\right)^{-1} \left(\boldsymbol{\psi}^1 - \boldsymbol{\mu}^0\right) \\ + \sum_{a=1}^{k}\sum_{b=1}^{k}\left\{m_{ab}^1 \log\left[h\left(\psi_{ab}^1\right)\right] + \left(n_{ab}^1 - m_{ab}^1\right)\log\left[1 - h\left(\psi_{ab}^1\right)\right]\right\},\end{aligned} \tag{5.17}$$

where d is another constant that can be ignored.

I use the appropriate log-posterior, either (5.15) for $t \geq 2$ or (5.17) for $t = 1$, as the objective function for label-switching. I find that a simple local search (hill climbing) initialized using the estimated class memberships at the previous time steps suffices, because only a small fraction of nodes change classes between time steps in most applications.

The inference procedure for a posteriori blockmodeling using local search and the EKF is presented in Algorithm 5.1. At the initial time step, multiple random initializations could be employed to avoid getting stuck in local optima. A faster approach is to use the spectral clustering algorithm of Sussman et al. (2012) for the SSBM as the initialization, which appears to work well in practice. Pseudocode for the SSBM spectral clustering algorithm

Algorithm 5.1 A posteriori blockmodel inference procedure at time t using the EKF.

while $iter \leq max\_iter$ **do**
  $\hat{\mathbf{c}}^t \leftarrow \hat{\mathbf{c}}^{t-1}$ {Initialize class memberships}
  Compute block densities Y^t using W^t and $\hat{\mathbf{c}}^t$
  Compute $\hat{\boldsymbol{\psi}}^{t|t}$ using EKF equations (5.9)–(5.13)
  Compute log-posterior p^t by substituting $\hat{\boldsymbol{\psi}}^{t|t}$ for $\boldsymbol{\psi}^t$ in (5.15) or (5.17)
  $\bar{p}^t \leftarrow -\infty$ {Posterior probability of best neighboring solution up to a constant}
  $\tilde{\mathbf{c}}^t \leftarrow \hat{\mathbf{c}}^t$ {Solution currently being visited}
  for $i = 1$ **to** $|V^t|$ **do** {Local search (hill climbing) algorithm}
    for $j = 1$ **to** k **do**
      $\tilde{c}_i^t \leftarrow j$ {Change class of a single node}
      Compute block densities $\tilde{Y}^t$ using W^t and $\tilde{\mathbf{c}}^t$
      Compute $\tilde{\boldsymbol{\psi}}^{t|t}$ using EKF equations (5.9)–(5.13)
      Compute log-posterior $\tilde{p}^t$ by substituting $\tilde{\boldsymbol{\psi}}^{t|t}$ for $\boldsymbol{\psi}^t$ in (5.15) or (5.17)
      if $\tilde{p}^t > \bar{p}^t$ **then** {Visited solution is better than best neighboring solution}
        $\left[\bar{p}^t, \bar{\boldsymbol{\psi}}^{t|t}, \bar{\mathbf{c}}^t\right] \leftarrow \left[\tilde{p}^t, \tilde{\boldsymbol{\psi}}^{t|t}, \tilde{\mathbf{c}}^t\right]$
      end if
      $\tilde{c}_i^t \leftarrow \hat{c}_i^{t-1}$ {Reset class membership of current node}
    end for
  end for
  if $\bar{p}^t > p^t$ **then** {Best neighboring solution is better than current best solution}
    $\left[p^t, \hat{\boldsymbol{\psi}}^{t|t}, \hat{\mathbf{c}}^t\right] \leftarrow \left[\bar{p}^t, \bar{\boldsymbol{\psi}}^{t|t}, \bar{\mathbf{c}}^t\right]$
  else {Reached local maximum}
    break
  end if
end while
return $\left[\hat{\boldsymbol{\psi}}^{t|t}, \hat{\mathbf{c}}^t\right]$

Algorithm 5.2 SSBM spectral clustering initialization.

Compute singular value decomposition of W^1
$\tilde{\Sigma} \leftarrow$ diagonal matrix of k largest singular values
$(\tilde{U}, \tilde{V}) \leftarrow$ left and right singular vectors for $\tilde{\Sigma}$
$\tilde{Z} \leftarrow \left[\tilde{U}\tilde{\Sigma}^{1/2}, \tilde{V}\tilde{\Sigma}^{1/2}\right]$ {concatenate scaled left and right singular vectors}
$\hat{\mathbf{c}}^0 \leftarrow$ k-means clustering on rows of Z
return $\hat{\mathbf{c}}^0$

is shown in Algorithm 5.2.

5.3.3 Time complexity

I begin with an analysis of the time complexity of the inference procedure for a priori blockmodeling at each time step. Calculating the matrix of block densities Y^t involves summing over all edges present at time t, which has $O(|E^t|)$ time complexity, where $|E^t|$ denotes the number of edges in the graph at time t. Application of the EKF requires only matrix-vector multiplications and a matrix inversion (to calculate the Kalman gain). The size of both the observation and state vectors in the EKF is $p = O(k^2)$, so the time complexity of the EKF is dominated by the $O(p^3) = O(k^6)$ complexity of the matrix inversion. Hence the overall time complexity at time t is $O(|E^t| + k^6)$.

A posteriori blockmodeling involves performing a local search at each time step. At each iteration of the local search, all $|V^t|(k-1)$ neighboring class assignments are visited. For each class assignment, I compute the EKF estimate $\hat{\boldsymbol{\psi}}^{t|t}$ and substitute it into the log-posterior (5.15) or (5.17). As previously mentioned, computing the EKF estimate is dominated by the $O(k^6)$ complexity of an inverting a $O(k^2) \times O(k^2)$ matrix. Evaluating the log-posterior also requires inversion of a $O(k^2) \times O(k^2)$ matrix. The matrix inversions are independent of the class assignments so they only need to be performed once at each time step rather than at each iteration of the local search. Thus the time complexity at each local search iteration is reduced to $O(k^4)$ for the matrix-vector multiplications. The overall time complexity at time t then becomes $O(|E^t| + k^6 + |V^t| l k^5)$, where l denotes the number of local search iterations.

5.3.4 Estimation of hyperparameters

The EKF requires four hyperparameters to be specified:

1. the mean $\boldsymbol{\mu}^0$ of the initial state $\boldsymbol{\psi}^0$,
2. the covariance matrix Γ^0 of the initial state $\boldsymbol{\psi}^0$,

3. the covariance matrix Σ^t of the observation noise $\mathbf{z}^t$, and
4. the covariance matrix Γ of the process (state evolution) noise $\mathbf{v}^t$.

The first two hyperparameters relate to the initial state. In the absence of prior information about the network, specifically the matrix Θ^0 of probabilities of forming edges, I employ a diffuse prior; that is, I let the variances of the initial states approach ∞. This can be implemented by simply taking $\hat{\boldsymbol{\psi}}^{1|1} = g(\mathbf{y}^1)$ and and $R^{1|1} = G^t\Sigma^1(G^t)^T$, where $g_i(x) = h_i^{-1}(x) = \log(x) - \log(1-x)$ is the logit of x, and G^t is the Jacobian of g, which is a diagonal matrix with entries given by $g_{ii}^t = 1/p + 1/(1-p)$. Thus the initial state mean and covariance are given by the transformed initial observation and its covariance.

The third hyperparameter Σ^t denotes the covariance matrix of the observation noise. In many Kalman filtering applications, it is assumed to be time-invariant and estimated jointly with Γ. In the state-space SBM setting, however, Σ^t is necessarily time-varying because it is related to the current state $\boldsymbol{\psi}^t$ through (5.4) and the logistic function h. Taking advantage of this relationship, I use a plug-in estimator for Σ^t by substituting $h\left(\hat{\boldsymbol{\psi}}^{t|t-1}\right)$ for Θ^t in (5.4).

The final hyperparameter Γ denotes the covariance matrix of the process noise $\mathbf{v}^t$. Unlike Σ^t, Γ is assumed to be time-invariant and is not necessarily diagonal because states could evolve in a correlated manner. For example, if ψ_{ab}^t increases from time $t-1$ to time t, it may be a result of some action by nodes in class a, which could also affect the other entries in row a of Ψ^t. Although Γ is not necessarily diagonal, it is desirable to impose some structure on Γ so that one does not have to estimate all $p^2 = O(k^4)$ covariances individually. Accordingly I assume the structure of Γ is such that

$$\gamma_{ij} = \begin{cases} s_{in}^2, & i = j \\ s_{out}^2, & i, j \text{ are neighboring cells in } \Psi^t \\ 0, & \text{otherwise,} \end{cases} \tag{5.18}$$

where i, j being neighboring cells means that the matrix indices (a_i, b_i) corresponding to i in Ψ^t are in the same row or column as matrix indices (a_j, b_j). This choice for Γ exploits the fact that the state Ψ^t is actually a matrix that has been flattened into a vector $\boldsymbol{\psi}^t$.

The objective is then to estimate s_{in} and s_{out}. Several methods have been proposed for learning hyperparameters in non-linear dynamic systems. One approach, often referred to as the joint EKF, involves treating the hyperparameters as additional states of the dynamic system. These hyperparameters are then estimated at each time step jointly with the states (Haykin, 2001). Another approach involves iteratively maximizing the likelihood of the observation sequence $W^{(t)}$ using the expectation-maximization (EM) algorithm because the true values of the states are unknown (Ghahramani and Roweis, 1998). This approach has been found to be successful in the off-line setting where past, present, and future observations are available to estimate the present state; however, not much is known about its performance in the on-line setting. I opt for the simpler approach of simply choosing hyperparameters to minimize the empirical average prediction error

$$\sum_{s=1}^{t-1} \left\| \mathbf{y}^{s+1} - h\left(\hat{\boldsymbol{\psi}}^{s+1|s}\right) \right\|_2^2.$$

5.4 Experiments

5.4.1 Simulated stochastic blockmodels

In this experiment I generate synthetic networks in a manner similar to Yang et al. (2011). The procedure is a dynamic extension of a synthetic network generator proposed by Newman and Girvan (2004). The network consists of 128 nodes initially split into 4 classes of 32 nodes each. The mean $\boldsymbol{\mu}^0$ of the initial state $\boldsymbol{\psi}^0$ is chosen so that $\mathrm{E}\left[\theta_{aa}^t\right] = 0.2580$ and $\mathrm{E}\left[\theta_{ab}^t\right] = 0.0834$ for $a, b = 1, 2, 3, 4; a \neq b$. This results in a fixed mean degree of 16 with the mean number of edges from a node to nodes in other classes $z = 4$. The initial state covariance Γ^0 is chosen to be a scaled identity matrix $s_0^2 I$.

Θ^t evolves through the random walk model defined on $\boldsymbol{\psi}^t$ in (5.8). Γ is constructed

using (5.18), where s_{in} and s_{out} are varied to simulate different types of environments. At each time step, 10% of the nodes are randomly selected to leave their class and are randomly assigned to one of the other three classes. 20 time steps are generated in each simulation run, and the results presented are averaged over 50 simulation runs. At each time, I draw a new graph snapshot from the SBM parameterized by Θ^t and $\mathbf{c}^t$. The graph snapshots in this experiment are undirected.

I compare the performance of the proposed state-space SBM fitted using the EKF to two baselines: the static stochastic block model (SSBM) fitted by maximum-likelihood and the probabilistic simulated annealing (PSA) algorithm proposed by Yang et al. (2011), which uses a combination of Gibbs sampling and simulated annealing to perform approximate inference. The PSA algorithm fits a dynamic SBM that is similar to the one I propose. There are, however, two differences:

1. Yang et al. (2011) explicitly model changes in class memberships over time using a transition matrix, as discussed in Section 5.1.
2. Yang et al. (2011) treat the matrix of edge probabilities Θ^t as a random time-invariant parameter, where each entry has a prior distribution of $\mathrm{Beta}(\alpha_{ab}, \beta_{ab})$.

In the a priori setting, only the EKF and SSBM are applicable, while all three methods are applicable in the a posteriori setting.

The mean and standard error (over the 50 simulation runs) of the MSE and mean adjusted Rand index (Hubert and Arabie, 1985) over all time steps are shown in Tables 5.1 and 5.2, respectively. Table 5.1 shows the MSE for both a priori and a posteriori methods. Four different simulated scenarios, corresponding to differing combinations of (s_0, s_{in}, s_{out}), are used:

1. $(s_0, s_{in}, s_{out}) = (0.02, 0.01, 0.005)$.
2. $(s_0, s_{in}, s_{out}) = (0.02, 0.1, 0.05)$.
3. $(s_0, s_{in}, s_{out}) = (0.2, 0.01, 0.005)$.
4. $(s_0, s_{in}, s_{out}) = (0.2, 0.1, 0.05)$.

Method		Simulation scenario			
		1	2	3	4
A priori	EKF	$\mathbf{0.16 \pm 0.01}$	$\mathbf{1.28 \pm 0.04}$	$\mathbf{0.36 \pm 0.02}$	$\mathbf{1.32 \pm 0.03}$
	SSBM	2.06 ± 0.04	2.11 ± 0.05	2.02 ± 0.05	2.17 ± 0.05
A posteriori	EKF	$\mathbf{0.27 \pm 0.02}$	$\mathbf{3.34 \pm 0.27}$	$\mathbf{1.30 \pm 0.12}$	$\mathbf{4.04 \pm 0.91}$
	SSBM	3.99 ± 0.21	10.54 ± 1.34	7.90 ± 1.00	12.16 ± 2.09
	PSA	1.76 ± 0.27	9.70 ± 0.78	4.22 ± 0.60	14.13 ± 2.17

Table 5.1: Mean and standard error of MSE in SBM experiments. Simulation scenario corresponds to choices of simulation parameters (s_0, s_{in}, s_{out}). Bolded number indicates best performer. All entries are $\times 10^{-3}$.

Method	Simulation scenario			
	1	2	3	4
EKF	0.803 ± 0.003	0.729 ± 0.015	0.752 ± 0.015	0.743 ± 0.017
SSBM	0.791 ± 0.004	0.703 ± 0.019	0.726 ± 0.017	0.718 ± 0.020
PSA	$\mathbf{0.881 \pm 0.004}$	$\mathbf{0.796 \pm 0.015}$	$\mathbf{0.813 \pm 0.015}$	$\mathbf{0.794 \pm 0.019}$

Table 5.2: Mean and standard error of adjusted Rand index in SBM experiments.

The results shown for both EKF and PSA are for optimally chosen hyperparameters. Notice that for each combination, the EKF has the lowest MSE in both the a priori and a posteriori settings. Since PSA assumes that Θ^t is time-invariant, one might expect it to perform poorly in the second and fourth scenarios where the variation of Θ^t over time is certainly not negligible. However, I find that PSA performs poorly in terms of MSE for all four combinations, including the first and third scenarios where Θ^t is almost time-invariant.

When it comes to accuracy in terms of estimating the true classes, PSA outperforms the EKF and SSBM as shown in Table 5.2. This is to be expected because the dynamic SBM of Yang et al. (2011) explicitly models changes in class memberships, whereas the state-space SBM I propose only implicitly constrains such changes by placing a state-space model on Θ^t. The variation of MSE and adjusted Rand indices over time for each method in the fourth simulation scenario are shown in Figure 5.2. Notice once again that PSA performs extremely poorly in terms of estimating Θ^t but performs well in estimating the true class memberships. However, the improved performance in estimating class memberships comes at the cost of higher computation time compared to the EKF. PSA required 743 seconds to

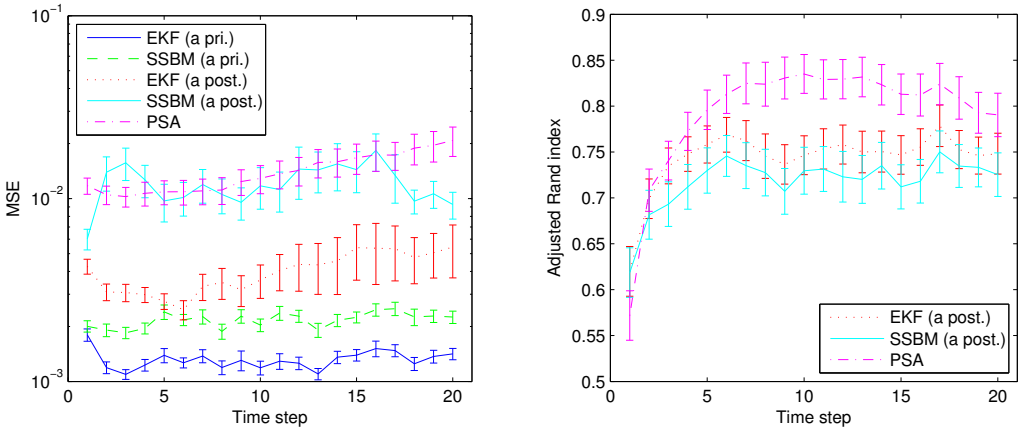
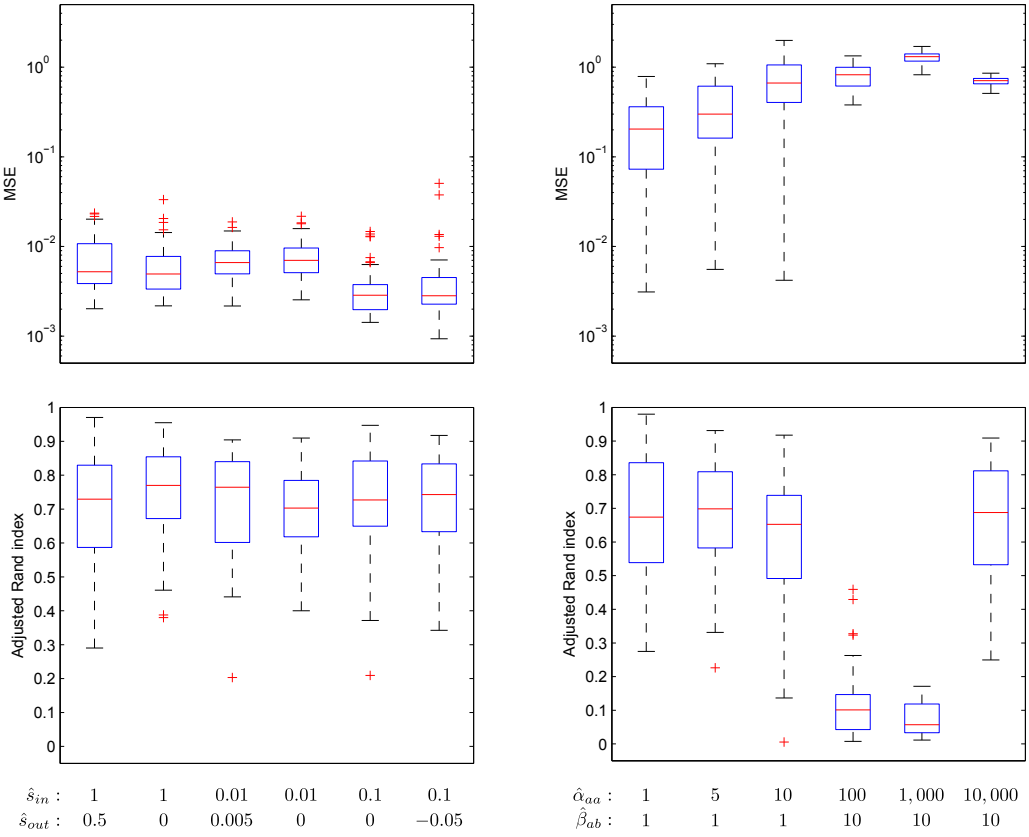


Figure 5.2: MSE (left) and adjusted Rand index (right) comparison in SBM experiment with $(s_0, s_{in}, s_{out}) = (0.2, 0.1, 0.05)$. Error bars denote one standard error.

process the data compared to 265 seconds for the EKF and 53 seconds for the SSBM. Recall that the EKF does not require any usage of Monte Carlo methods, whereas PSA involves Gibbs sampling, which accounts for its increased computation time.

While PSA is able to outperform the proposed EKF methods in adjusted Rand index (at the cost of higher computation time), it is also more sensitive to the choices of hyperparameters. In Figure 5.3, I plot the variation of MSE and adjusted Rand index for different choices of hyperparameters for both the a posteriori EKF and PSA. All plots are for the fourth simulation scenario. While the EKF is somewhat robust to poor choices of hyperparameters, PSA is extremely sensitive to poor choices. Yang et al. (2011) recommend choosing values of α_{ab} and β_{ab} that maximize the modularity criterion (Newman and Girvan, 2004), which is a measure of the strength of community structure for a given partition (when ground truth is not available). However, simply optimizing modularity may result in extremely high values in α_{ab} and β_{ab}, which correspond to extremely low variances in the prior distribution for Θ^t and lead to poor estimates of Θ^t, as shown in Figure 5.3b. Furthermore, it is not applicable when the classes do not correspond to communities, i.e. when Θ^t is not diagonally dominant.



(a) EKF (a posteriori) (b) PSA

Figure 5.3: Variation of MSE (top) and adjusted Rand index (bottom) on hyperparameter settings for EKF and PSA in fourth SBM simulation scenario. $\hat{\alpha}_{ab} = 1$ in all settings for PSA.

5.4.2 Enron emails

This experiment involves the Enron email data set (Priebe et al., 2009), which consists of about 0.5 million email messages between 184 Enron employees from 1998 to 2002. I construct the network by placing directed edges between employees i and j at time t if i sends at least one email to j during time step t. I take each time step to correspond to a 1-week interval. I make no distinction between emails sent "to", "cc", or "bcc". In addition to the email data, the roles of most of the employees within the company are available. I divide employees into seven roles:

1. directors,
2. CEOs,
3. presidents,
4. vice presidents,
5. managers,
6. traders, and
7. others,

which I use as classes for a priori blockmodeling.

The mean over all time steps of the estimated matrix of edge probabilities $\hat{\Theta}^{t|t}$ using the a priori EKF is shown in Figure 5.4. Several observations can be made about the overall structure of the network over the entire duration of the data trace. First notice that the highest probabilities of emails are sent among CEOs and presidents. For a company that was known to have engaged in fraud involving the CEOs and presidents, this is not terribly surprising. Another observation is the asymmetry of emails sent to and received from the board of directors. Emails are occasionally sent to directors, but emails are very rarely received from directors. This is also to be expected, as management are held accountable by the directors, who mostly observe and do not get involved in the day-to-day operations of the company. A more surprising observation is that the managers are more likely to email

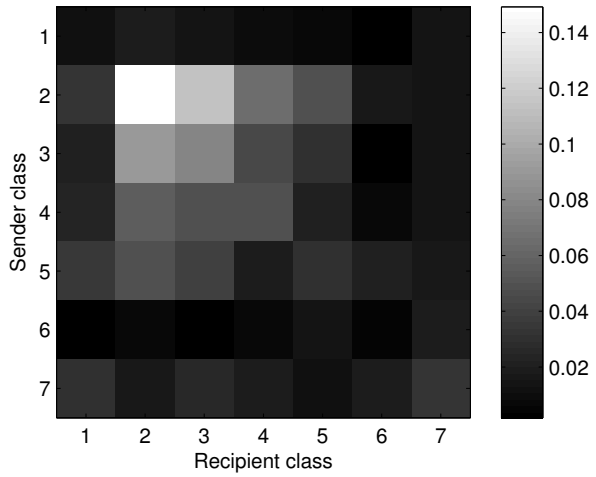


Figure 5.4: Mean estimated edge probabilities $\hat{\Theta}^{t|t}$ over all time steps for Enron data using a priori EKF. The entry in row i, column j denotes the estimated probability that an employee in class i sends an email to an employee in class j.

CEOs and presidents than they are to email vice presidents. This is surprising because managers typically sit one level below the vice presidents on the corporate ladder; thus one might expect the managers to correspond more frequently with the vice presidents than the presidents and CEOs. In short, by simply looking at the mean estimated probabilities over the duration of the trace, one can discover some interesting insights into the company structure.

While the insights gained from looking at the mean $\hat{\Theta}^{t|t}$ are interesting, they could have also been obtained by fitting a static stochastic blockmodel (SSBM) to the entire data trace aggregated into a single static network. The power of the state-space approach is that it estimates a time series of matrices $\hat{\Theta}^{t|t}$. Of particular interest are the estimated probabilities of emails originating from the CEOs, corresponding to the second row of $\hat{\Theta}^{t|t}$. These seven time series are shown in Figure 5.5. Notice that there are three spikes, which can be viewed as anomalies, that show up in several of the time series; these spikes occur shortly after several important events in the timeline of the Enron scandal, which supports their interpretation as anomalies. The week corresponding to highlighted date 2 was also found to be anomalous by Priebe et al. (2005) using scan statistics. These spikes are difficult to

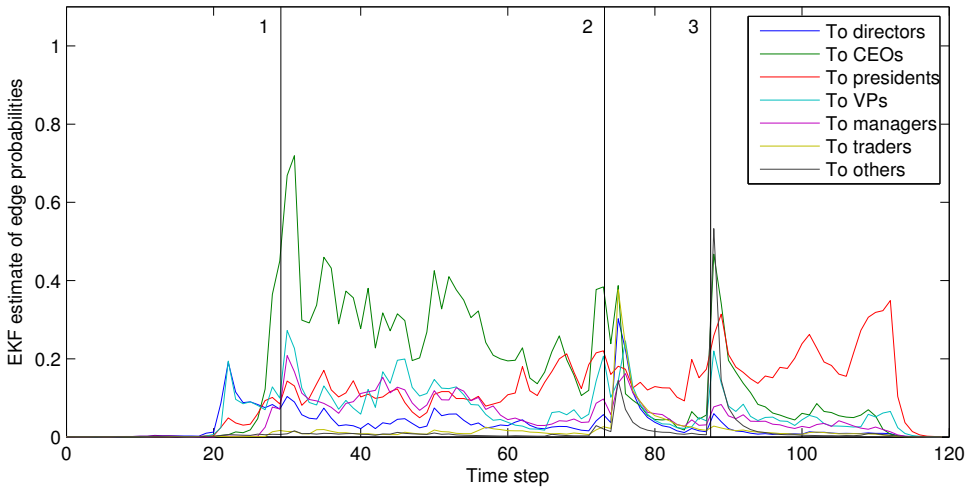


Figure 5.5: Temporal variation of EKF estimated edge probabilities from Enron CEOs $\hat{\theta}_{2b}^{t|t}$. The three highlighted dates correspond to (1) Enron issuing a Code of Ethics; (2) Enron's stock closing below \$60, a critical point in one of their partnerships; and (3) CEO Jeffrey Skilling's resignation.

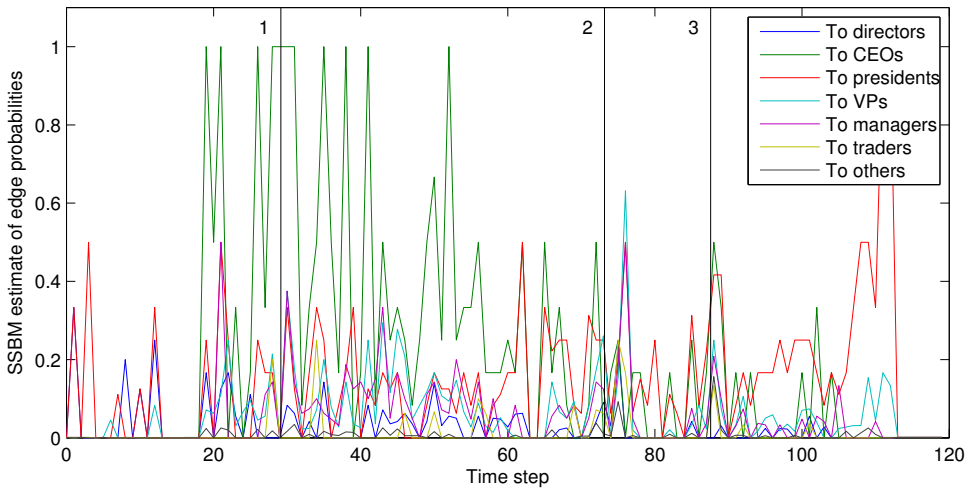


Figure 5.6: Temporal variation of SSBM estimated edge probabilities from Enron CEOs y_{2b}^{t}. The spikes in email activity at the highlighted dates is less noticeable due to the high variance.

identify when fitting an SSBM to each time step individually, as shown in Figure 5.6, where they are buried in the high variance of the SSBM estimates. This example demonstrates the superiority of the state-space approach, which constrains estimates at successive time steps to smooth out the time series in a near-optimal manner.

Next I evaluate the performance of both the a priori and a posteriori EKF inference for the task of link prediction. The link prediction problem in dynamic networks involves predicting the edges that will be present at time $t+1$ from the observations $W^{(t)}$. In the usual link prediction setting, a predictor is trained over a certain period of time then employed to predict newly formed edges in a growing network (Liben-Nowell and Kleinberg, 2007; Lichtenwalter et al., 2010). Link prediction in evolving networks differs because the predictor must simultaneously predict the new edges that will be formed at time $t+1$ as well as the current edges (as of time t) that will disappear at time $t+1$. The latter task is not addressed by most link prediction methods in the literature.

Since the SBM assumes stochastic equivalence between nodes in the same class, the EKF alone is only a good predictor of the block densities Y^t, not the edges themselves. However, the EKF (either a priori or a posteriori), which makes predictions at the block level, can be combined with another predictor that makes predictions at the individual level to form a good link predictor. A simple predictor at the individual level is the exponentially-weighted moving average (EWMA) given by $\hat{W}^{t+1} = \lambda \hat{W}^t + (1-\lambda)W^t$. By taking a convex combination of the EKF and EWMA predictors, one might expect to achieve better link prediction performance than either predictor individually. This is confirmed by the receiver operating characteristic (ROC) curves, shown in Figure 5.7, and the area under the curves (AUCs), shown in Table 5.3. Both the ROC curves and AUCs are for $\lambda = 0.5$. Notice that the a priori EKF performs worse than the a posteriori EKF; this is not surprising because the a posteriori EKF is able to find a better fit to the state-space SBM by assigning nodes to classes to maximize the posterior probability. Notice also that the EWMA outperforms both the a priori and a posteriori EKF alone in terms of AUC. Thus

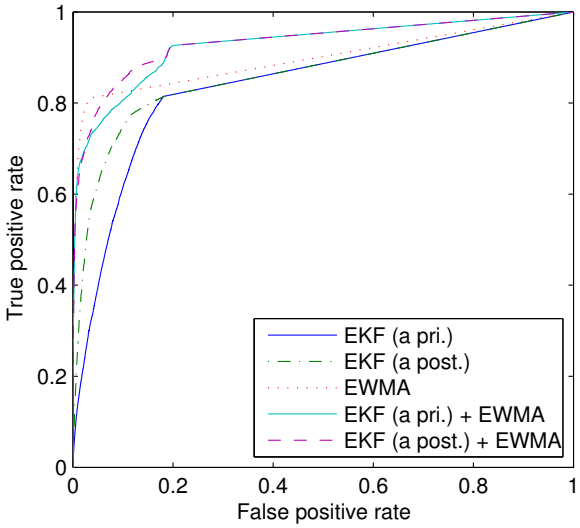


Figure 5.7: Comparison of ROC curves for link prediction on Enron data.

Method	AUC
EKF (a priori)	0.840
EKF (a posteriori)	0.863
EWMA	0.898
EKF (a priori) + EWMA	0.929
EKF (a posteriori) + EWMA	**0.934**

Table 5.3: AUC comparison for link prediction on Enron data.

it can be concluded that block-level characteristics are not enough to accurately predict individual links. However, by combining the EKF and EWMA, I obtain a significantly improved link predictor that accounts for both block-level characteristics (through the EKF) and individual-level characteristics (through the EWMA).

5.5 Summary

In this chapter I proposed a statistical model for dynamic networks using a state-space representation. The proposed model uses the stochastic blockmodel to represent individual network snapshots and a state-space model to represent the temporal evolution of the network. The model can be used in either the a priori setting, where nodes are known or assumed to belong to specific classes, or the a posteriori setting, where class mem-

berships are estimated along with probabilities of forming edges. I developed an on-line inference procedure for the proposed model that performs near-optimal state tracking in a computationally efficient manner using the extended Kalman filter. The proposed method was applied to a set of synthetic networks and a real dynamic social network and showed promising results in both tracking and link prediction.

CHAPTER VI

CONCLUSIONS

A variety of complex time-varying phenomena are well-modeled by dynamic, time-evolving networks. In this dissertation I presented computational methods for machine learning and statistical inference on dynamic networks, encompassing the tasks of visualization, clustering, tracking, and prediction. The proposed methods take advantage of the dynamic nature of the network by incorporating multiple time snapshots of the network and by modeling the temporal evolution of the network.

In Chapter II I presented an analysis of the social networks of email spammers. Specifically, I discovered communities of spammers that utilized common resources, namely spam servers, as well as communities of spammers that exhibited high temporal correlation when sending spam. Although the network was dynamic, I used an algorithm designed for static networks, namely spectral clustering, to perform the analysis one month at a time. The approach was suboptimal because it neglected the dynamic nature of the network, which motivated the development of the computational methods for dynamic networks in the following chapters.

In Chapter III I proposed a framework for visualizing dynamic networks using animated 2-D graph layouts. Effectively visualizing dynamic networks is an important step toward understanding the temporal dynamics and structural evolution of the networks. The proposed framework creates regularized graph layouts, which encourage dynamic stability of the network visualization over time, allowing a human to interpret the changes in the

network structure without being overwhelmed by large quantities of node movements.

Chapter IV discussed the problem of tracking the evolution of communities or clusters in dynamic networks. I proposed an adaptive evolutionary clustering framework that reformulated evolutionary clustering as a problem of optimal state tracking followed by static clustering. Under the proposed framework, called AFFECT, I created dynamic extensions of three static clustering algorithms, including the static spectral clustering algorithm used in Chapter II for analyzing communities of spammers. The proposed evolutionary clustering algorithms produced more stable and consistent clustering results over time.

In Chapter V I proposed a state-space stochastic blockmodel for dynamic networks along with an inference procedure using the extended Kalman filter (EKF). The proposed model is an extension of the static stochastic blockmodel to dynamic networks. It characterizes a dynamic network by assigning each node to one of k classes at each time step then specifying the probability that any node in class a will form an edge with any node in class b at any particular time step. These edge-forming probabilities are treated as time-varying states of the dynamic network. Using the proposed model and inference procedure, it is possible to perform near-optimal state tracking in a computationally efficient manner. The estimated states reveal the underlying structure of the network from the noisy observations at each time step.

There are many interesting problems for future work, particularly in developing statistical models for dynamic networks such as the one I developed in Chapter V. Specifically, it would be desirable to add individual attributes to the state-space stochastic block model to create a richer representation that could lead to better link prediction ability. Another useful extension would be to build a hierarchical blockmodel that could allow for scalability to higher numbers of classes by restricting covariances at different levels of the hierarchy.

An issue that has not been adequately addressed in dynamic network modeling involves the temporal aspect of the model. Existing work assumes a simple temporal model for the state evolution, such as a random walk. I also assume a random walk model in Chapter V.

However, such a simple model likely cannot characterize the complex temporal dynamics of networks such as social networks, which likely exhibit linear and non-linear trends, periodicity, and change points among others. Capturing such dynamics fits nicely into the state-space framework; however, many questions including model selection and evaluation remain to be answered.

Another interesting area for future work involves combining models for dynamic networks with models of influence on networks. Most existing models for dynamic networks assume that the network is passively observed. However in many situations, one might be able to locally affect the network by exerting influence on other nodes. By combining network models with influence models, one could possibly be able to predict the effects that one's actions could have on the network state in the future, which could have significant implications for sociologists, marketers, computer scientists, and the intelligence community among others.

BIBLIOGRAPHY

A. Ahmed and E. P. Xing. Dynamic non-parametric mixture models and the recurrent chinese restaurant process: with applications to evolutionary clustering. In *Proc. SIAM International Conference on Data Mining*, 2008.

A. Ahmed and E. P. Xing. Recovering time-varying networks of dependencies in social and biological studies. *Proceedings of the National Academy of Sciences*, 106(29):11878–11883, 2009.

E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing. Mixed membership stochastic blockmodels. *Journal of Machine Learning Research*, 9:1981–2014, 2008.

T. W. Anderson. *An introduction to multivariate statistical analysis*. Wiley, 3rd edition, 2003.

I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, G. Paliouras, and C. D. Spyropoulos. An evaluation of naive Bayesian anti-spam filtering. In *Proc. Workshop on Machine Learning in the New Information Age, 11th European Conference on Machine Learning*, 2000.

M. Austin. Spam at epic levels, Nov. 2006. URL `http://www.itpro.co.uk/97589/spam-at-epic-levels`.

M. Baur and T. Schank. Dynamic graph drawing in Visone. Technical report, 2008.

M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear programming: Theory and algorithms*. Wiley, 2006.

M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.

S. Bender-deMoll and D. A. McFarland. The art and science of dynamic network visualization. *Journal of Social Structure*, 7(2):1–38, 2006.

S. Bender-deMoll and D. A. McFarland. SoNIA - Social Network Image Animator, 2012. URL `http://www.stanford.edu/group/sonia/`.

I. Borg and P. J. F. Groenen. *Modern multidimensional scaling*. Springer, 2nd edition, 2005.

U. Brandes and S. R. Corman. Visual unrolling of network evolution and the analysis of dynamic discourse. *Information Visualization*, 2(1):40–50, 2003.

R. H. Byrd, M. E. Hribar, and J. Nocedal. An interior point algorithm for large-scale nonlinear programming. *SIAM Journal on Optimization*, 9(4):877–900, 1999.

A. Carmi, F. Septier, and S. J. Godsill. The Gaussian mixture MCMC particle algorithm for dynamic cluster tracking. In *Proc. 12th International Conference on Information Fusion*, 2009.

X. Carreras and L. Marquez. Boosting trees for anti-spam email filtering. In *Proc. Recent Advances in Natural Language Processing*, 2001.

D. Chakrabarti, R. Kumar, and A. Tomkins. Evolutionary clustering. In *Proc. 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

M. Chandrasekaran, K. Narayanan, and S. Upadhyaya. Phishing e-mail detection based on structural properties. In *Proc. 9th New York State Cyber Security Conference*, 2006.

M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *SIAM Journal on Computing*, 33(6):1417–1440, 2004.

Y. Chen, A. Wiesel, Y. C. Eldar, and A. O. Hero III. Shrinkage algorithms for MMSE covariance estimation. *IEEE Transactions on Signal Processing*, 58(10):5016–5029, 2010.

Y. Chi, X. Song, D. Zhou, K. Hino, and B. L. Tseng. Evolutionary spectral clustering by incorporating temporal smoothness. In *Proc. 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007.

F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

J. A. Costa and A. O. Hero III. Classification constrained dimensionality reduction. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2005.

G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph drawing: Algorithms for the visualization of graphs*. Prentice Hall, 1999.

E. Dimitriadou, A. Weingessel, and K. Hornik. A combination scheme for fuzzy clustering. *International Journal of Pattern Recognition and Artificial Intelligence*, 16(7):901–912, 2002.

H. Drucker, D. Wu, and V. N. Vapnik. Support vector machines for spam categorization. *IEEE Transactions on Neural Networks*, 10(5):1048–1054, 1999.

Z. Duan, K. Gopalan, and X. Yuan. Behavioral characteristics of spammers and their network reachability properties. In *Proc. IEEE International Conference on Communications*, 2007.

N. Eagle, A. Pentland, and D. Lazer. Inferring friendship network structure by using mobile phone data. *Proceedings of the National Academy of Sciences*, 106(36):15274–15278, 2009.

C. Erten, P. J. Harding, S. G. Kobourov, K. Wampler, and G. Yee. Exploring the computing literature using temporal graph visualization. In *Proc. Conference on Visualization and Data Analysis*, 2004.

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

Y. Frishman and A. Tal. Online dynamic graph drawing. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):727–740, 2008.

T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991.

E. R. Gansner, Y. Koren, and S. North. Graph drawing by stress majorization. In *Graph Drawing*, 2004.

Z. Ghahramani and S. T. Roweis. Learning nonlinear dynamical systems using an EM algorithm. In *Advances in Neural Information Processing Systems 11*, 1998.

M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

A. Goldenberg, A. X. Zheng, S. E. Fienberg, and E. M. Airoldi. A survey of statistical network models. *Foundations and Trends® in Machine Learning*, 2(2):129–233, 2010.

A. Gretton, K. M. Borgwardt, M. Rasch, B. Schölkopf, and A. J. Smola. A kernel approach to comparing distributions. In *Proc. 22nd AAAI Conference On Artificial Intelligence*, 2007.

F. Guo, S. Hanneke, W. Fu, and E. P. Xing. Recovering temporally rewiring networks: A model-based approach. In *Proc. 24th International Conference on Machine Learning*, 2007.

C. Gupta and R. Grossman. GenIc: A single pass generalized incremental algorithm for clustering. In *Proc. SIAM International Conference on Data Mining*, 2004.

K. M. Hall. An r-dimensional quadratic placement algorithm. *Management Science*, 17(3): 219–229, 1970.

A. C. Harvey. *Forecasting, structural time series models and the Kalman filter*. Cambridge University Press, 1989.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.

S. Haykin. *Kalman filtering and neural networks*. Wiley-Interscience, 2001.

I. Herman, G. Melançon, and M. S. Marshall. Graph visualisation and navigation in information visualisation: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

Q. Ho, L. Song, and E. P. Xing. Evolving cluster mixed-membership blockmodel for time-varying networks. In *Proc. 14th International Conference on Artificial Intelligence and Statistics*, 2011.

A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

P. D. Hoff, A. E. Raftery, and M. S. Handcock. Latent space approaches to social network analysis. *Journal of the American Statistical Association*, 97(460):1090–1098, 2002.

P. Holland, K. B. Laskey, and S. Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5(2):109–137, 1983.

L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985.

J. Jung and E. Sit. An empirical study of spam traffic and the use of DNS black lists. In *Proc. 4th Internet Measurement Conference*, 2004.

T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(12):7–15, 1989.

B. Karrer and M. E. J. Newman. Stochastic blockmodels and community structure in networks. *Physical Review E*, 83:016107, 2011.

Y. Koren. On spectral graph drawing. In *Proc. 9th International Computing and Combinatorics Conference*, 2003.

O. Ledoit and M. Wolf. Improved estimation of the covariance matrix of stock returns with an application to portfolio selection. *Journal of Empirical Finance*, 10(5):603–621, 2003.

J. A. Lee and M. Verleysen. *Nonlinear dimensionality reduction*. Springer, 2007.

J. Leskovec. *Dynamics of large networks*. PhD thesis, Carnegie Mellon University, 2008.

J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Statistical properties of community structure in large social and information networks. In *Proc. 17th International Conference on the World Wide Web*, 2008.

L. Leydesdorff and T. Schank. Dynamic animations of journal maps: Indicators of structural changes and interdisciplinary developments. *Journal of the American Society for Information Science and Technology*, 59(11):1810–1818, 2008.

Y. Li, J. Han, and J. Yang. Clustering moving objects. In *Proc. 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7):1019–1031, 2007.

R. N. Lichtenwalter, N. Dame, J. T. Lussier, and N. V. Chawla. New perspectives and methods in link prediction. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2010. ISBN 9781450300551.

Y. R. Lin, Y. Chi, S. Zhu, H. Sundaram, and B. L. Tseng. Analyzing communities and their evolutions in dynamic social networks. *ACM Transactions on Knowledge Discovery from Data*, 3(2):8, 2009.

H. Lütkepohl. *Handbook of matrices*. Wiley, 1997.

J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

S. Milgram. The small world problem. *Psychology Today*, 1(1):61–67, 1967.

G. W. Milligan and M. C. Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50(2):159–179, 1985.

K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages and Computing*, 6(2):183–210, 1995.

MIT–WWW. MIT Academic Calendar 2004-2005, 2005. URL `http://web.mit.edu/registrar/www/calendar0405.html`.

J. Moody, D. McFarland, and S. Bender-deMoll. Dynamic network visualization. *American Journal of Sociology*, 110(4):1206–1241, 2005.

P. J. Mucha, T. Richardson, K. Macon, M. A. Porter, and J. P. Onnela. Community structure in time-dependent, multiscale, and multiplex networks. *Science*, 328(5980):876–878, 2010.

J. Nazario. Third bad ISP disappears — McColo gone, Nov. 2008. URL `http://ddos.arbornetworks.com/2008/11/third-bad-isp-dissolves-mccolo-gone/`.

T. M. Newcomb. *The acquaintance process*. Holt, Rinehart and Winston, 1961.

M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physics Review E*, 69(2):026113, 2004.

A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Proc. Advances in Neural Information Processing Systems 14*, 2001.

H. Ning, W. Xu, Y. Chi, Y. Gong, and T. S. Huang. Incremental spectral clustering by efficiently updating the eigen-system. *Pattern Recognition*, 43(1):113–127, 2010.

P. G. Nordlie. *A longitudinal study of interpersonal attraction in a natural group setting*. PhD thesis, University of Michigan, 1958.

K. Nowicki and T. A. B. Snijders. Estimation and prediction for stochastic blockstructures. *Journal of the American Statistical Association*, 96(455):1077–1087, 2001.

C. Parker. Boids pseudocode, 2007. URL `http://www.vergenet.net/~conrad/boids/pseudocode.html`.

C. E. Priebe, J. M. Conroy, D. J. Marchette, and Y. Park. Scan statistics on Enron graphs. *Computational & Mathematical Organization Theory*, 11(3):229–247, 2005.

C. E. Priebe, J. M. Conroy, D. J. Marchette, and Y. Park. Scan statistics on Enron graphs, 2009. URL `http://cis.jhu.edu/~parky/Enron/enron.html`.

M. Prince, B. Dahl, L. Holloway, A. Keller, and E. Langheinrich. Understanding how spammers steal your e-mail address: An analysis of the first six months of data from Project Honey Pot. In *Proc. 2nd Conference on Email and Anti-Spam*, 2005.

A. Ramachandran and N. Feamster. Understanding the network-level behavior of spammers. In *Proc. ACM SIGCOMM*, 2006.

A. Ramachandran, D. Dagon, and N. Feamster. Can DNS-based blacklists keep up with bots? In *Proc. 3rd Conference on Email and Anti-Spam*, 2006.

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

C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. In *Proc. 14th ACM SIGGRAPH International Conference on Computer Graphics and Interactive Techniques*, 1987.

G. Robins, P. Pattison, Y. Kalish, and D. Lusher. An introduction to exponential random graph (p^*) models for social networks. *Social Networks*, 29(2):173–191, 2007.

K. Rohe, S. Chatterjee, and B. Yu. Spectral clustering and the high-dimensional stochastic blockmodel. *Annals of Statistics*, 39(4):1878–1915, 2011.

J. Rosswog and K. Ghose. Detecting and tracking spatio-temporal clusters with adaptive history filtering. In *Proc. 8th IEEE International Conference on Data Mining Workshops*, 2008.

P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

P. Sarkar and A. W. Moore. Dynamic social network analysis using latent space models. *ACM SIGKDD Explorations Newsletter*, 7(2):31–40, 2005.

J. Schäfer and K. Strimmer. A shrinkage approach to large-scale covariance matrix estimation and implications for functional genomics. *Statistical Applications in Genetics and Molecular Biology*, 4(1):32, 2005.

P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. Cytoscape: A software environment for integrated models of biomolecular interaction networks. *Genome Research*, 13(11):2498–2504, 2003.

J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

T. A. B. Snijders and K. Nowicki. Estimation and prediction for stochastic blockmodels for graphs with latent block structure. *Journal of Classification*, 14(1):75–100, 1997.

J. Sun, S. Papadimitriou, P. S. Yu, and C. Faloutsos. Graphscope: Parameter-free mining of large time-evolving graphs. In *Proc. 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007.

D. L. Sussman, M. Tang, D. E. Fishkind, and C. E. Priebe. A consistent adjacency spectral embedding for stochastic blockmodel graphs. *arXiv preprint*, 2012.

L. Tang, H. Liu, J. Zhang, and Z. Nazeri. Community evolution in dynamic multi-mode networks. In *Proc. 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 58(1):267–288, 1996.

Unspam Technologies, Inc. Project Honey Pot, 2012. URL `http://www.projecthoneypot.org`.

Visone–WWW. Visone, 2012. URL `http://www.visone.info/`.

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

Y. Wang, S. X. Liu, J. Feng, and L. Zhou. Mining naturally smooth evolution of clusters from dynamic data. In *Proc. SIAM International Conference on Data Mining*, 2007.

S. Wasserman and P. Pattison. Logit models and logistic regressions for social networks: I. an introduction to markov graphs and p^*. *Psychometrika*, 61(3):401–425, 1996.

D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

A. H. Westveld and P. D. Hoff. A mixed effects model for longitudinal relational and network data, with applications to international trade and conflict. *Annals of Applied Statistics*, 5(2A):843–872, 2011.

D. M. Witten and R. Tibshirani. Supervised multidimensional scaling for visualization, classification, and bipartite ranking. *Computational Statistics & Data Analysis*, 55(1): 789–801, 2011.

D. M. Witten, R. Tibshirani, and T. Hastie. A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis. *Biostatistics*, 10(3):515–534, 2009.

E. P. Xing, W. Fu, and L. Song. A state-space mixed membership blockmodel for dynamic network tomography. *Annals of Applied Statistics*, 4(2):535–566, 2010.

T. Xu, Z. Zhang, P. S. Yu, and B. Long. Evolutionary clustering by hierarchical Dirichlet process with hidden Markov state. In *Proc. 8th IEEE International Conference on Data Mining*, 2008a.

T. Xu, Z. Zhang, P. S. Yu, and B. Long. Dirichlet process based evolutionary clustering. In *Proc. 8th IEEE International Conference on Data Mining*, 2008b.

T. Yang, Y. Chi, S. Zhu, Y. Gong, and R. Jin. Detecting communities and their evolutions in dynamic social networks—a Bayesian approach. *Machine Learning*, 82(2):157–189, 2011.

S. X. Yu and J. Shi. Multiclass spectral clustering. In *Proc. 9th IEEE International Conference on Computer Vision*, 2003.

J. Zhang, Y. Song, G. Chen, and C. Zhang. On-line evolutionary exponential family mixture. In *Proc. 21st International Joint Conference on Artificial Intelligence*, 2009.

J. Zhang, Y. Song, C. Zhang, and S. Liu. Evolutionary hierarchical Dirichlet processes for multiple correlated time-varying corpora. In *Proc. 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2010.

Y. Zhao, E. Levina, and J. Zhu. On consistency of community detection in networks. Technical report, 2011. arXiv:1110.3854v1 [math.ST].